Exploratory Testing

Controle over het testen van informatiesystemen

Eugene Derksen en Patrice Hulzebos

Exploratory Testing
Controle over het testen van informatiesystemen

Omslagontwerp en boekverzorging | Eugene Derksen, Velp
Illustraties | Celeste Hurenkamp en Inge van Vught, Utrecht

Inhoudsopgave

Voorwoord

Er is al veel geschreven over testen, heel veel… Maar er is nog nooit een geslaagde poging gedaan om exploratory testing in Nederland bekendheid te geven. Ik ben dan ook blij dat Eugene Derksen en Patrice Hulzebos als ervaren en gewaardeerde test-experts hier verandering in willen brengen door dit boek en dat ze mij hebben gevraagd om het voorwoord voor mijn rekening te nemen.

Zelf ben ik al langere tijd een Context Driven Agile Tester en maak hierbij volop gebruik van exploratory testing, maar ik merk dat het elke keer opnieuw een forse uitdaging is om product owners, architecten, projectmanagers, ontwikkelaars en soms zelfs testers uit te leggen dat exploratory testing een gestructureerde testtechniek is die in weinig tijd heel veel resultaat geeft. Het imago van exploratory testing is nog altijd dat het een ongestructureerde techniek is. Alsof exploratory testing iets is als ´een beetje rondklikken´. Eugene en Patrice willen dit imago met dit boek veranderen. Naast een duidelijk beschrijving van wat exploratory testing nu echt is gaan zij nog een stap verder en beschouwen zij exploratory testing als een testmethode, niet zozeer als een testtechniek. Dit doen zij door testmanagement, coaching en rapportage binnen exploratory testing expliciet te maken zonder daarbij de uitgangspunten van exploratory testing te verliezen.

Dit boek is een geslaagde poging om exploratory testing in Nederland voor een grotere doelgroep te ontsluiten. Het wordt hiermee ontdaan van het imago dat het niet gestructureerd zou zijn en dat het slechts een techniek is die je zou kunnen toepassen, net als bepaalde TMap testtechnieken. Exploratory testing is ´ a way of testing´ en dit boek maakt dat duidelijk en is daarmee een aanrader voor iedereen die een raakvlak heeft met software testen.

Carlo van Driel
Directeur De Agile Testers

Inleiding

Onze maatschappij is in toenemende mate afhankelijk van informatiesystemen. De kwaliteit van deze systemen is daarmee ook steeds belangrijker geworden. Recente onderzoeken wijzen echter uit dat de kwaliteit van informatiesystemen - ondanks groter wordende testinspanningen - niet verbetert. Dit stelt ons de vraag hoe wij als testers deze kwaliteit kunnen verbeteren. Binnen traditionele testmethoden wordt een aanzienlijk deel van de tijd aan allerlei activiteiten onder de verzamelnaam testen besteed. Volgens de theorie van deze methoden wordt voor ongeveer 75% van de tijd gespecificeerd en gedocumenteerd, waardoor de effectieve tijd voor testuitvoering wordt beperkt tot zo'n 25%.

Voorbereiding kan kwaliteitsverhogend zijn als het gaat om toetsen van ontwikkel- en systeem-documentatie, maar er wordt feitelijk niets mee getest. Zo wordt slechts een klein deel van het totale testbudget gespendeerd aan het daadwerkelijk verbeteren van de kwaliteit van het informatiesysteem. Stel je eens voor welke winst er te behalen is door efficiënter specificeren en documenteren!

'Slechts een klein deel van het totale testbudget wordt gespendeerd aan daadwerkelijk testen.'

Traditionele testmethoden geven met uitgebreide testspecificaties het gevoel van zekerheid dat er goed en gestructureerd wordt getest. Dit is echter een schijnzekerheid. De hoeveelheid testdocumentatie heeft geen enkele relatie met kwaliteit van de testuitvoering. Wij willen de lezer uitdagen om de schijnzekerheid los te laten en de stap te zetten naar echt testen.

Met dit boek willen wij aantonen dat er nog andere methoden zijn om tot kwalitatief zeer goede tests te komen. Andere methoden die

in plaats van of naast de traditionele methoden gebruikt kunnen worden. Wat wij willen is testers een vernieuwende optie bieden om zich meer te onderscheiden in hun vakgebied en hun testaanpak beter af te kunnen stemmen op de vraag vanuit de klant en de markt.

Exploratory testing is een iteratieve testmethode met als uitgangspunt dat leren, specificeren en uitvoeren van de testen zoveel mogelijk gelijktijdig plaatsvindt. In plaats van specificaties bestaat de testbasis voor exploratory testing uit ervaring, domeinkennis, testkennis, creativiteit en logisch denkvermogen. Exploratory testing besteedt een substantieel groter deel van het testbudget aan daadwerkelijke testuitvoering en minder budget aan het testproces en ondersteunende documentatie. Het niet vooraf opstellen van testspecificaties bespaart tijd en geld. Het stelt je in staat flexibeler om te gaan met wijzigingen op het laatste moment. Deze wijze van testen sluit daarom zeer goed aan bij Agile ontwikkelmethoden zoals bijvoorbeeld Scrum en daarmee bij de huidige vraag en context van systeemontwikkeling.

De eerste paar hoofdstukken in dit boek beschrijven een stuk geschiedenis en achtergrond van zowel systeemontwikkeling als testen. Het testmanifest in het daaropvolgende hoofdstuk beschrijft de uitgangspunten van de testvisie die beschreven wordt in dit boek. Hierna wordt exploratory testing uitgebreid beschreven, te beginnen bij het ontstaan, de rollen, het werkproces en hoe dit alles op efficiënte en gecontroleerde wijze kan worden gemanaged.

Reviewen is een erg krachtig middel om kwaliteit te verhogen en om die reden wordt daar expliciet een hoofdstuk aan gewijd. Het beschreven exploratory testing proces voldoet ook aan de principes van Lean en past geweldig binnen Scrum dat in de daaropvolgende hoofdstukken wordt beschreven. Exploratory testing is een goede manier om te testen binnen hedendaagse systeemontwikkeling.

Niet alles wordt uitputtend beschreven in dit boek. De toepassing van exploratory testing is context afhankelijk. Wij beperken ons daarom tot hoofdlijnen en laten de specifieke invulling van deze methode aan de lezer en diens context en creativiteit. Wij willen stimuleren dat er met een frisse blik naar het vakgebied testen wordt gekeken.

Geschiedenis van systeemontwikkeling en testen

Om een helder beeld te krijgen van waar we nu staan op het gebied van testen, is het goed enige achtergrond te hebben over de veranderingen in systeemontwikkeling van de afgelopen decennia.

Watervalmethoden
De eerste methoden voor systeemontwikkeling werden rond 1970 geïntroduceerd, allen vallend in de groep van watervalmethoden. Deze methoden schrijven voor om een systeemontwikkelproces in duidelijke fasen af te bakenen. De volgordelijkheid is belangrijk, een volgende fase wordt pas gestart als de vorige is afgerond. Na het afronden van een definitiestudie, wordt er een basisontwerp of globaal ontwerp van het te ontwikkelen systeem gemaakt. Dit wordt vervolgens volledig uitgewerkt in detailontwerpen. Daarna wordt het systeem in zijn geheel gebouwd. Aansluitend daarop volgt dan de testfase en op moment dat de kwaliteit goed genoeg wordt geacht, volgen de integratiefase en de acceptatiefase om het systeem uiteindelijk in productie te nemen in de laatste fase.

Later bleek steeds meer dat het moeilijk was om met weinig klantbetrokkenheid in het proces een voor de klant bruikbaar product af te leveren. Daarnaast kwamen er tijdens de realisatie en de tests pas (technische) problemen boven tafel die grote veranderingen in eerder afgeronde fases vereisten, zoals de definitiestudie of globaal ontwerp. Laatste belangrijke nadeel was dat watervaltrajecten vaak lang duurden, waardoor de wensen en eisen die in het begin waren opgesteld door de tijd achterhaald waren en vaak tot grote wijzigingen konden leiden. Om het risico van onder meer deze nadelen te verminderen, volgden er alternatieve methoden.

RAD
Vanaf 1980 ontstonden er meer iteratieve ontwikkelmethoden onder de naam Rapid Application Development (RAD). RAD wordt

gezien als de verzamelnaam voor deze ontwikkelmethoden en is daarnaast zelf ook een in 1991 geformaliseerde methode. Een belangrijk verschil van deze methoden ten opzichte van watervalmethoden is het hanteren van de principes van timeboxen en prototyping. In het begin van het ontwikkeltraject worden er prototypes gemaakt van de later op te leveren applicatie. Deze prototypes dienden in een eerder stadium om mogelijke (technische) beperkingen aan het licht te brengen en dienden daarnaast ook als hulpmiddel bij de communicatie met de klant en toekomstige gebruikers. Risico's op veel rework later in het traject werden daarmee verkleind. Dit was een duidelijke verbetering ten opzichte van de watervalmethoden. Een ander belangrijk verschil is dat de software iteratief ontwikkeld wordt, wat inhoudt dat de diverse fasen herhalend worden doorlopen met korte doorlooptijden. Bij die iteraties werd het principe van timeboxen gebruikt, waardoor een RAD-traject ook qua tijd beter beheersbaar is ten opzichte van een watervaltraject. De iteraties werden aan de start voor het gehele traject al ingepland. Door dit timeboxen voorkom je de standaard valkuil van te veel verzanden in details. Volgens de 80-20 regel, kost de laatste 20% functionaliteit 80% van de tijd. Door het timeboxen voorkom je dat je hierin structureel qua planning uit de bocht vliegt en ligt de focus op de 80%.

RAD leverde weliswaar meer stabiliteit op het gebied van planning en tijd, maar op het gebied van functionaliteit begonnen er soms omissies te ontstaan. De 80% gerealiseerde functionaliteit leverde vaak nieuwe inzichten op, maar hiervoor was geen speelruimte om dit later in het traject te kunnen realiseren. Ook de 20% 'ontbrekende' functionaliteit bevatte vaak toch essentiële en noodzakelijke functionaliteit. RAD maakte ontwikkeltrajecten qua tijd en geld beter stuurbaar, maar het ontbrak aan een mechanisme voor herijking na iedere iteratie en het handelen hierop doordat de vastgestelde iteraties dit onmogelijk maakten. Het uiteindelijk gevolg van deze problemen was, dat er nog steeds een beperkt bruikbaar product ontstond.

Agile methoden

In februari 2001 kwamen 17 softwareontwikkelaars samen in Utah om andere mogelijkheden voor softwareontwikkeling te bespreken. Hiervoor stelden zij het Agile Manifesto op, dat als basis dient voor alle Agile softwareontwikkelmethoden. Deze ontwikkelmethoden moeten het hoofd bieden aan de problemen voortvloeiend uit de lineaire- en op RAD gebaseerde ontwikkelmethoden en de steeds complexer wordende systemen met een snel veranderende klantvraag.

De Agile softwareontwikkelmethoden ontwikkelen direct werkende software. Ook binnen Agile wordt er iteratief software ontwikkeld. Dit gebeurt in multidisciplinaire zelfsturende teams die ook zelf verantwoordelijk zijn voor het planningsaspect. Dit levert over het algemeen betere planningen op waaraan het team zich ook committeert. Zo wordt er meer resultaat behaald met de diverse mensen die betrokken zijn bij het softwareontwikkeltraject. De inhoud van een iteratie wordt pas bepaald vlak voor de start van de iteratie op basis van de dan geldende inzichten, inhoud en belangen op basis van een samenwerking tussen het team en de klant. De klant is veel nauwer betrokken bij het gehele ontwikkeltraject, waardoor er een constante afstemming en terugkoppeling is over het ontwikkelde eindproduct. Deze manier van werken levert de flexibiliteit op die de ontwikkeling van de huidige complexe systemen nodig heeft. Klantbetrokkenheid en de herijkingen na elke iteratie bieden het hoofd aan de vele nieuwe inzichten gedurende het traject en leiden tot een voor de klant veel beter bruikbaar systeem.

Verderop in dit boek wordt er nog dieper ingegaan op softwareontwikkelmethoden. Wat voor nu in ieder geval helder is, is dat softwareontwikkelmethoden zijn geëvolueerd. Ze hebben zich aangepast aan complexere te ontwikkelen systemen, meer behoefte aan klantbetrokkenheid en het beter kunnen benutten van mensen en middelen. Niet alle ontwikkelingen die hierboven zijn geschetst zijn per definitie een verbetering in elke situatie voor

elk te ontwikkelen systeem. Wat wel belangrijk is, is dat er diverse opties zijn waaruit je een keuze kunt maken.

Maar hoe is het testen van software geëvolueerd in deze zelfde periode?

TMap

Activiteiten onder de verzamelnaam testen werden vanaf ongeveer 1970 als een aparte fase in systeemontwikkeling gezien. Het testen gebeurde in het begin voornamelijk door de ontwikkelaar zelf, die testen er 'een beetje bij deed'. Het duurde nog jaren voordat testen als een afzonderlijke discipline gezien werd. In 1995 is TMap op de markt verschenen. TMap heeft een belangrijke bijdrage geleverd aan standaardisering van het testvak en het ontstaan van testen als een aparte professie. TMap was de eerste echte testmethode die op dat moment een prima aanpak omschreef voor testen binnen de watervalmethoden. Nederland liep hiermee op gebied van methodisch testen voorop in de wereld.

'TMap is de eerste echte testmethode en heeft daarmee een belangrijke bijdrage gehad aan standaardisering binnen het testvak.'

TMap hanteert zijn eigen fasering welke naadloos aansluit op de fasering van watervalmethoden. Daarmee heeft TMap ook vergelijkbare nadelen. Testspecificering wordt pas gestart indien de detailontwerpen gereed zijn, testuitvoering indien de bouw klaar is. Gevonden defects, specificatiewijzigingen of verkeerd ingeschatte verwachtingen zorgen er veelal voor dat testspecificaties moeten worden aangepast en tests opnieuw moeten worden uitgevoerd.

TestFrame
Mede door grootschalige trajecten rondom de millenniumbug en de invoering van de Euro, heeft geautomatiseerd testen aan populariteit gewonnen. TestFrame is ontstaan als framework voor geautomatiseerd testen. Later is het TMap faseringsmodel en de beschreven testspecificatietechnieken toegevoegd waardoor een afzonderlijke testmethode TestFrame ontstaan is. Ook voor TestFrame gelden daarmee de nadelen die voor TMap ook gelden. Wellicht nog iets meer, aangezien het aanpassen en werkend houden van een geautomatiseerde test in de regel meer tijd kost dan het aanpassen van een gedocumenteerde testspecificatie. Tegenwoordig wordt TestFrame weinig tot niet meer gebruikt, maar het heeft wel de basis gelegd voor Data Driven Testing. En dat zie je veel terug in de huidige test-tooling voor geautomatiseerd testen.

TMap Next
In 2006 kwam de opvolger van de TMap methode, TMap Next. Grootste veranderpunten zijn de uitbreidingen tot het benoemen van TMap tot een adaptieve methode en het Business Driven Test Management (BDTM). Hiermee is geprobeerd om de voor alleen waterval geschikte methode ook bruikbaar te maken voor meer iteratieve methoden en de klant meer te betrekken bij het testen. Daarnaast zijn er documenten verschenen zoals TMap Next binnen Scrum om de geschiktheid voor agile methoden aan te geven. Ondanks deze aanpassingen blijft TMap in essentie een watervalmethode, ook al zijn deze door het iteratieve karakter meer verworden tot kleine kortcyclische watervalletjes. Het sterk planmatige en gefaseerde karakter blijft behouden.

SmarTEST
In 2007 is SmarTest op de markt geïntroduceerd. Deze maakt in plaats van het bekende V-model gebruik van een W-model, waarin het onderste deel van de V eigenlijk geïtereerd wordt, waardoor er een W ontstaat. Het itererende deel bestaat uit ontwerp, bouw en ontwikkeltesten. Toch houdt ook dit model vooral de

acceptatietesten buiten een iteratie en laat de systeemtest buiten het iteratieve deel van ontwerpen, bouwen en ontwikkeltesten.

TMap HD

In 2014 is TMap HD verschenen als 'opvolger' van TMap Next. Hoewel TMap Next nog steeds als bruikbaar wordt omschreven voor meer lineaire (waterval) trajecten, is TMap HD meer Human Driven. Het wordt gepositioneerd naast TMap Next, als onderdeel van de gehele TMap suite. TMap HD is een set van gedefinieerde bouwblokken welke kunnen worden aangevuld met eigen gecreeërde bouwblokken. En uit deze complete set van bouwblokken kan dan naar behoefte worden geput om op die manier een passende testmethode op te bouwen. Deze bouwstenen zijn gebaseerd op 4 principes: vereenvoudigen, integreren, industrialiseren en mens centraal.

Testmethoden

Het mag duidelijk zijn dat het overzicht van testmethoden in dit hoofdstuk niet een uitputtende lijst is en dat er de afgelopen jaren ook zeker nog andere methoden zijn ontwikkeld. Deze zijn echter veelal op kleinere schaal gebruikt en om die reden niet expliciet vermeld in bovenstaand overzicht.

TMap (Next) is veruit de meest gebruikte methode in Nederland. Sommige bedrijven hebben zelfs hun eigen van TMap afgeleide methode gecreëerd met wat accentverleggingen ten opzichte van het origineel. Ook zijn er een aantal bedrijven die de SmarTEST methode gebruiken in hun testtrajecten. TestFrame is een aanpak die je nog zelden tegenkomt.

Wat is testen?

Testen is geen synoniem voor nakijken. Nakijken is een proces van valideren en verifiëren waarbij output gecontroleerd wordt tegen vooraf vastgelegde voorspellingen. Testen is een proces van verkennen, ontdekken, onderzoeken, leren en analyseren. Dit met als doel relevante informatie te verzamelen over kwetsbaarheden en risico's van, en bedreigingen voor, een informatiesysteem. Het managen van verwachtingen tijdens de systeemontwikkeling is daarbij een proces van continu afstemming zoeken en in contact blijven met de klant. Testen je als volgt kunnen definiëren: 'Testen is het managen van risico's en verwachtingen rond het ontwikkelen en onderhouden van informatiesystemen'.

'Testen is het managen van risico's en verwachtingen rond het ontwikkelen en het onderhouden van informatiesystemen.'

Een informatiesysteem beheert gegevens en ondersteunt informatiestromen. Het omvat daarmee meer dan alleen software. Een informatiesysteem kan bestaan uit hardware, software, gegevens, mensen en procedures en andere zaken die een rol spelen bij de informatievoorziening. Testen beperkt zich daarmee niet alleen tot het testen van software.

Traditionele testmethoden
Volgens de testmethode TMap Next wordt testen als volgt gedefinieerd: 'Softwaretesten is een proces dat inzicht geeft in en adviseert over de kwaliteit van het systeem en de daaraan gerelateerde risico's.'

Binnen het raamwerk van ISTQB wordt softwaretesten als volgt beschreven: 'Testen is het proces van alle activiteiten uit de

levenscyclus, zowel statisch als dynamisch, die te maken hebben met de planning, voorbereiding en evaluatie van software-producten en hieraan gerelateerde werkproducten om vast te stellen dat ze voldoen aan gespecificeerde eisen, om aan te tonen dat ze voldoen aan de doelen en om fouten te vinden.'

Beide definities hebben het expliciet over het testen van software en niet over een informatiesysteem. Het ontbreekt hier misschien wel aan het belangrijkst deel, namelijk de context waarbinnen het systeem moet functioneren wat het maakt tot een informatie-systeem. Goede software maakt nog geen goed informatiesysteem.

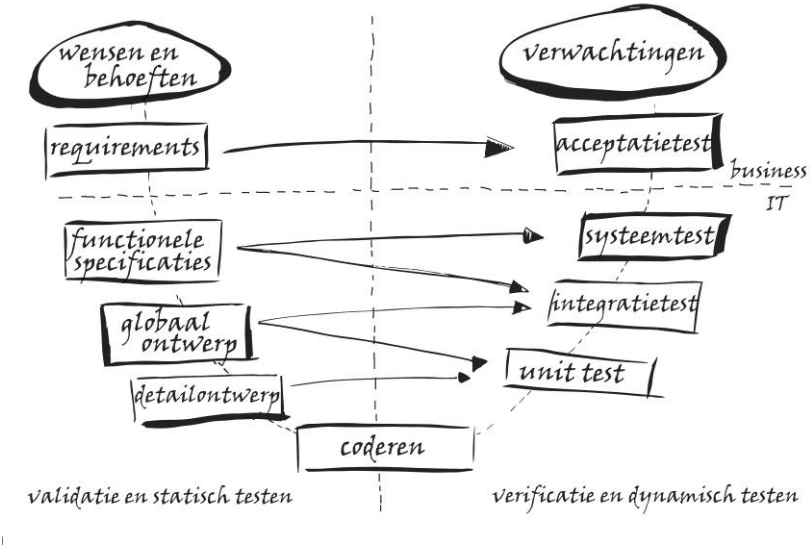
'Goede software maakt nog geen goed informatiesysteem.'

Traditionele testmethoden kenmerken zich veelal door gebaseerd te zijn op het V-model of een variant hierop. In het V-model worden de fasen en producten in het systeemontwikkelingstraject afgezet tegen de verschillende testsoorten. Het gaat in op zowel het statisch testen (reviews en inspecties) als ook het dynamisch testen waarbij programmatuur wordt uitgevoerd. Door het hanteren van het V-model wordt er nadrukkelijk onderscheid gemaakt in de verantwoordelijkheden van business en IT.

Het V-model geeft aan dat er verschillende (uitgangs)documenten noodzakelijk zijn om de testen op te kunnen baseren en te specificeren. Indien deze documenten niet aanwezig of kwalitatief beperkt zijn, is er voor de toepassing van deze methoden al een flink probleem. Tevens onderscheidt het V-model verschillende fasen die volgordelijk moeten worden uitgevoerd. De voorgaande fase moet zijn afgerond voordat met de nieuwe fase begonnen wordt.

Een vroege variant van het hedendaagse V-model is terug te vinden in het boek van Bill Hetzel 'The Complete Guide to Software Testing', voor het eerst uitgegeven in 1984. In figuur 1 staat een voorbeeld van een wat recenter V-model.

Het V-model dwingt een aantal voorafgaande processtappen en documentatie af, voordat de testuitvoering kan starten. Elke fase borduurt voort op de producten uit een voorgaande fase. In iedere volgende fase borduur je ook voort op fouten in die producten die worden veroorzaakt door vertaling en interpretatie.



Figuur 1: V-model

Omdat je steeds verder in detail gaat zullen de fouten als een olievlek verspreiden en als goed worden geïnterpreteerd tijdens bijvoorbeeld reviews. Tevens worden per fase ook nieuwe fouten geïntroduceerd. Gevolg hiervan is een opeenstapeling van fouten in volgende fasen. In testfasen zal het erg lastig zijn om deze fouten te

vinden mede omdat er mogelijk op basis van foute uitgangs-documentatie getest wordt.

Omdat er geen toetsing aan de wensen en behoeften van de klant plaatsvindt gedurende alle IT-fasen, zullen veel van dergelijke essentiële fouten pas worden onderkend in een acceptatietest.
Hoe verder je in het V-model afdaalt, hoe groter de kans dat producten uit de voorgaande fasen afwijken van de wensen en behoeften van de klant. De klant is immers nauwelijks betrokken bij deze dieperliggende fasen. Deze betrokkenheid is er pas weer in de laatste fase van het V-model waarbij de essentiële fouten pas daadwerkelijk worden onderkend. Boehm zegt dat zo vroeg mogelijk gevonden fouten het goedkoopst zijn om te herstellen. Hanteren van het V-model leidt per definitie niet tot het vroegtijdig vinden van de werkelijk relevante fouten. Immers, je bent met het kloppend maken van de softwareproducten in de fasering bezig en niet met de wensen en behoeften van de klant. En juist de laat ontdekte fouten zijn het meest duur om te herstellen.

Hoe dieper in het model, hoe minder de kennis rond de context van het informatiesysteem is. De gevonden fouten zijn daarmee ook vaak minder relevant dan problemen die tijdens de acceptatiefase aan het licht komen. Verder maakt het V-model een duidelijk onderscheid tussen business en IT. De klant zou echter doorlopend betrokken moeten zijn bij de systeemontwikkeling. Aan het begin van een traject eisen en wensen opstellen en aan het einde een acceptatietest uitvoeren is niet voldoende. De klant moet ook betrokken zijn bij het proces en de processtappen aan IT-zijde. Zowel de klant als ook IT hebben wensen, veranderende eisen en problemen. Door wederzijdse betrokkenheid zijn de verwachtingen beter af te stemmen en leer je meer over elkaars uitdagingen, processen en kennis. Dit voorkomt dat aan het eind van het ontwikkeltraject een onoverbrugbaar verschil is ontstaan tussen de verwachtingen van de klant en de gerealiseerde software door IT. Het behoeft geen uitleg dat het overbruggen van dit verschil aan

het einde van het ontwikkeltraject een zeer kostbare zaak is en daarnaast ook een gespannen relatie oplevert tussen klant en IT.

Testprincipes

In de loop van de tijd zijn er door diverse mensen en instanties principes opgesteld omtrent testen, waarop eigenlijk iedere aanpak gebaseerd zou moeten zijn. Ze worden hier besproken voor de volledigheid en verderop in dit boek zullen we hier nog op terug komen. Er zijn 7 principes voor testen geformuleerd. Deze principes zouden de basis moeten vormen voor elke test en bijbehorende aanpak.

Principe 1: Testen toont verschillen/aanwezigheid van fouten aan

Testen vermindert de waarschijnlijkheid dat er bugs in het informatiesysteem achterblijven, maar testen kan nooit aantonen dat een applicatie 100% foutloos werkt.

Principe 2: Alles testen is onmogelijk

Het testen van alle mogelijke situaties is onmogelijk. Daarom dienen de belangrijkste situaties wel getest te worden.

Principe 3: Test zo vroeg mogelijk

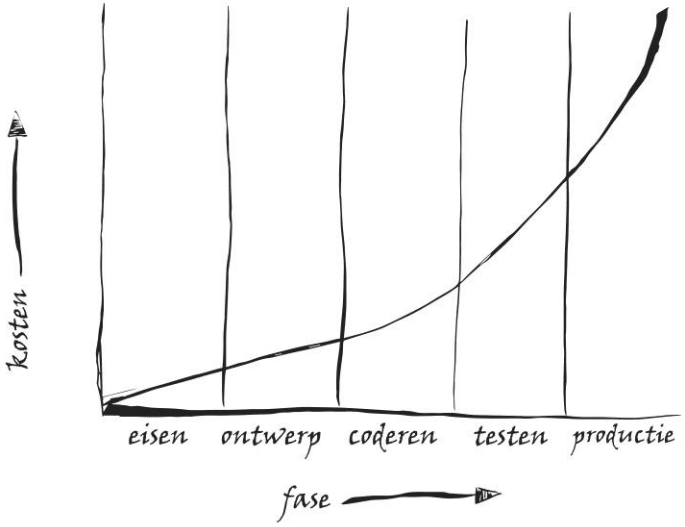
Testen moet zo vroeg mogelijk in de softwareontwikkeling plaatshebben. De belangrijkste bevindingen dienen zo vroeg mogelijk gevonden te worden, zodat ze zo goedkoop mogelijk zijn om op te lossen. De Curve van Boehm (figuur 2) is de meest bekende uitwerking van dit principe, gepubliceerd door Barry Boehm in zijn boek ‘Software Engineering Economics’ uit 1981.

Principe 4: Defect clustering

Een beperkte set componenten en/of applicatiedelen bevat het merendeel aan defects en veroorzaakt de meeste problemen in productie.

Principe 5: Pesticide paradox

Als je telkens dezelfde set testen herhaalt dan levert dat op een zeker moment geen defects meer op. Het informatiesysteem zal immuun worden voor de tests.



Figuur 2: Curve van Boehm

Principe 6: Testen is context afhankelijk

De testsoort, het testontwerp, de testuitvoering en de resultaten van de test worden bepaald door de context van het informatiesysteem.

Principe 7: De ontbreken-van-defects denkfout

Het testen van een applicatie garandeert niet dat de applicatie voldoet aan de wensen van de belanghebbenden. Het ontbreken van bevindingen zegt niets over de kwaliteit van het informatiesysteem. Een informatiesysteem is pas goed wanneer het bruikbaar is voor de klant.

Testen loopt achter

Na jarenlang in Nederland voorop te hebben gelopen op testgebied met een duidelijke methode voor testen binnen lineaire systeem-ontwikkelingstrajecten, zijn we hier veel te lang in blijven hangen. De wereld van systeemontwikkeling is in hoog tempo veranderd en de ontwikkelingen binnen het testen zijn hierbij erg achter gebleven. Door de achterstand op testgebied zitten we met een groot probleem, waardoor testen ineens een beperkende factor is geworden en veel minder toegevoegde waarde heeft binnen systeemontwikkeling. De problemen zijn de testmethode, de testers en de huidige toepassing van testautomatisering.

‘Door de achterstand op testgebied zitten we met een groot probleem, waardoor testen ineens een beperkende factor is geworden.’

Testmethode

We beschikken in Nederland niet over een geschikte testmethode voor agile systeemontwikkelingstrajecten. Alle testmethoden die in het hoofdstuk tot dusver zijn beschreven, zijn allemaal gebaseerd op het V-model. Deze methoden zijn niet flexibel en vereisen een aantal zaken op gebied van documentatie alvorens er met testen gestart kan worden. In een Agile traject werkt dit vertragend voor de rest van het team. Je creëert wachttijd en daarmee frustreer je het kort cyclische en de op snelle feedback gebaseerde aanpak. Daarnaast heb je documentatie nodig als testbasis, welke in een Agile systeemontwikkelingstraject vaak minder of niet voorhanden is en zeker niet klaar is op het moment dat de software getest moet worden.

De meest gebruikte methode, TMap, behoort toe aan een ICT-dienstverlener die naast de methode zelf ook training en certificering levert. Er is jarenlang verkondigd dat TMap Next prima geschikt was voor systeemontwikkeling binnen een agile omgeving vanwege het adaptieve karakter. Er zijn speciale en zeer omvangrijke papers geschreven om dat aan te tonen. Nu, met het verschijnen van TMap HD wordt door de organisatie achter TMap gezegd dat TMap Next alleen geschikt is voor lineaire ontwikkelmethoden en TMap HD geschikt is voor Agile methoden. Kortom, als er grote commerciële belangen spelen, gaat dit helaas vaak ten koste van een oprecht verhaal en komt het vaak neer op 'wij van wc-eend adviseren wc-eend'. Daarnaast zorgt een dergelijk groot commercieel belang er ook voor dat de positie van marktleider wordt gebruikt om andere initiatieven de kop in te drukken, veelal onder de noemer 'ongestructureerd.'

'Kortom, als er grote commerciële belangen spelen, gaat dit helaas vaak ten koste van een oprecht verhaal en komt het vaak neer op 'wij van wc-eend adviseren wc-eend'.

Dit heeft er mede toe geleid dat er op het gebied van testen de laatste 15 jaar eigenlijk weinig vernieuwends is gebeurd, helaas. De wet van de remmende voorsprong laat zich hier gelden. TMap is van een kookboek verworden tot een grote lijst met ingrediënten waar je dan als kok zelf uit kunt kiezen wat je nodig hebt en dat noemen we dan adaptief. Maar nu hebben we alleen geen kookboek meer...

Testers

Met een ongeschikte methode onder de arm, frustreren testers het Agile ontwikkeltraject. Ze blijven vanuit de onzekerheid over hoe te testen, zich koste wat kost vasthouden aan deze methode, ook

omdat ze geen andere opties zien. Hiermee maken ze zichzelf onderdeel van het probleem.

‘Met een ongeschikte methode onder de arm, frustreren testers het Agile ontwikkeltraject’.

Daarnaast zijn een groot aantal testers die geschikt zijn om te werken binnen het V-model, niet geschikt om goed te kunnen functioneren binnen een Agile ontwikkelteam. Dit heeft onder andere te maken met andere gevraagde vaardigheden op het gebied van communicatie, aanpak en houding welke eerder niet strikt noodzakelijk waren.

Bedrijven zijn veelal op zoek naar gecertificeerde testers. TMap gecertificeerd, Certified Agile Tester, etc. Echter, de meeste certificaten zeggen niets over hoe goed een tester in zijn vak is. Elke leraar wiskunde of elektromonteur kan een TMap foundation certificaat halen. Je hoeft hiervoor alleen begrippen en rijtjes te kunnen onthouden. Het is hetzelfde als een flinke hoeveelheid Franse woorden en zinnen leren en daar dan examen voor doen. Ben je dan iemand die goed Frans spreekt?

Een certificaat, of het ontbreken daarvan, zegt helaas erg weinig over jouw kunde als tester. Over je analytisch vermogen, goede communicatieve vaardigheden, werktempo, leercurve, etc. Het vermogen van een tester om in een testtraject de essentiële bugs te vinden, daar gaat het om!

Door de huidige certificeringsdrift en de manier waarop bedrijven daar mee om gaan is het voor eigenlijk iedereen mogelijk zich te laten ‘omscholen’ tot een gecertificeerd tester. Daarnaast was de vraag naar testers ook jarenlang erg hoog, waardoor ook bedrijven deze omscholingstrajecten flink hebben aangeboden. Maar

daarmee zijn er ook veel testers actief die niet de competenties hebben die nodig zijn voor een goede (Agile) tester. Een kwalitatief hoogwaardige review kunnen uitvoeren of precies die ene belangrijke bug vinden die niet uit de standaardtestgevallen naar boven komt, levert dan vaak een probleem op. En dat is nu wel net een belangrijk onderdeel van de potentiële meerwaarde van testen.

‘Het vermogen van een tester om in een testtraject de essentiële bugs te vinden, daar gaat het om!’

Testautomatisering

Binnen organisaties wordt meer en meer gebruik gemaakt van testautomatisering. Deze ontwikkeling zie je ook terug als standaard binnen Agile ontwikkeltrajecten en zelfs als vervanger van de handmatige functionele tests. Dit is een slecht gekozen oplossing voor de problemen die testen binnen Agile geeft. Deze oplossing maakt het probleem eigenlijk alleen maar erger. Doordat er zo veel geautomatiseerd wordt, creëer je dezelfde problemen als met het schrijven van uitgebreide testspecificaties in Agile. De scripts moeten worden gecreëerd, onderhouden bij elke aanpassing en bovenal ook nog werkend worden gehouden. Dit vergt dermate veel inspanning en levert vaak zo veel problemen op, dat dit afleidt van datgene waar het eigenlijk allemaal om zou moeten gaan: het vinden van de essentiële bugs in een zo kort mogelijke tijd. Automatiseren verwordt tot een doel op zich.

Met deze ontwikkeling is ook de huidige testpopulatie aan het verschuiven. Waar vroeger een groot deel van de testers gezien kon worden als functionele tester of testanalist, is dit percentage de afgelopen jaren aanzienlijk gedaald. De vraag naar technische testers is navenant toegenomen.

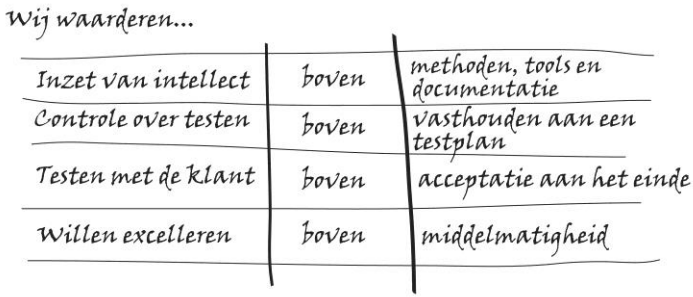
Binnen zelfsturende teams in Agile ontwikkelomgevingen kiezen bedrijven tegenwoordig ook voor ontwikkelaars die de rol van tester krijgen toegewezen. Dit met de achtergrond dat de ‘tester’ zich dan meer kan bemoeien met de technische oplossingen, code kan reviewen en als technische sparringpartner voor andere ontwikkelaars kan dienen. Ook worden testen meer en meer geautomatiseerd uitgevoerd, waardoor de coderingsvaardigheden van een ontwikkelaar ook gewenst zijn.

Bovengenoemde drie oorzaken zijn de reden waarom het mooie testvak met zo veel potentiële waarde lijkt te verdwijnen en niet de waarde heeft die het zou moeten hebben en ook kan hebben.

Testmanifest

In de vorige hoofdstukken zijn de ontwikkelingen in de markt en de stand van zaken op het vakgebied testen toegelicht. Het is duidelijk dat er een vraag is die door het hedendaagse testen onvoldoende wordt beantwoord. Het probleem zit hem hierbij in de testmethoden, de tester en de testautomatisering waarbij vastgehouden wordt aan achterhaalde uitgangspunten.

In het licht van deze tijd is er voor testen dringend behoefte aan een set actuele uitgangspunten die testen weer tot meerwaarde maakt. Deze uitgangspunten zijn verwoord in onderstaand Testmanifest.



Figuur 3: Testmanifest

Inzet van intellect
Testers verrichten nu voornamelijk lopende band werk. Functionele specificaties worden met een testtechniek gevormd tot een

testontwerp. Wanneer de software dan wordt opgeleverd worden de tests uitgevoerd volgens dat ontwerp. Dat kan comfortabel zijn, maar dit draagt niet bij tot toegevoegde waarde binnen het ontwikkelproces. De inzet van intellect betekent dat je als tester zelf nadenkt over hoe iets zo goed mogelijk binnen de context, met de beschikbare middelen en binnen de beschikbare tijd getest kan worden.

‘De inzet van intellect betekent dat je als tester zelf nadenkt over hoe iets zo goed mogelijk binnen de context, met de beschikbare middelen en binnen de beschikbare tijd getest kan worden.’

Onder inzet van intellect wordt verstaan, dat de tester:

- continu het exacte doel en de context van de test scherp stelt;
- de testaanpak bepaalt op basis van het doel en de context;
- creatief is en onder alle omstandigheden het doel weet te bereiken;
- alle mogelijke inspanningen doet om het doel zo snel mogelijk te bereiken.

Door het gebruik van intellect, weet de tester methoden, tools en documentatie te benutten. Dit in plaats van dat de tester zich laat (af)leiden door methoden, tools en documentatie.

Controle over testen

Feit is dat er gedurende een project veel wijzigt. Door aan het begin van een project een testplan op te stellen – met daarin tot in detail beschreven wat, hoe, wanneer, door wie en onder welke condities gedaan moet worden – maak je een testplan per definitie onbruikbaar en verlies je al snel de controle. Beter is het om te kiezen voor een testplan dat flexibiliteit biedt bij steeds wijzigende

omstandigheden. Hierin staat wel de aanpak, de te testen componenten en een basisplanning (budget) beschreven, maar het plan beschrijft niet in detail wat, wanneer, door wie en onder welke condities wordt getest. Dat wordt bepaald bij het begin van de testuitvoering. Hierdoor kies je uiteindelijk altijd de op dat moment beste optie, mede omdat alle opties ook nog open zijn. Dit geeft een optimale controle over het testen.

‘De kans dat een gedetailleerd testplan enigszins de waarheid benaderd is nihil. Controle over testen staat haaks op het maken van een gedetailleerd testplan.’

Onder controle over testen wordt verstaan, dat:

- het testplan de testaanpak beschrijft;
- de aanpak in het testplan je in staat stelt flexibel te zijn;
- met die flexibiliteit kun je op elk gewenst moment de testen aanpassen aan de aspecten tijd, geld en kwaliteit;
- de testresultaten zijn direct terug te koppelen in termen van tijd, geld en kwaliteit;
- er is altijd volledig inzicht in de status van het testen.

De kans dat een gedetailleerd testplan enigszins de waarheid benaderd is nihil. Controle over testen staat haaks op het maken van een gedetailleerd testplan. Zo’n plan laat vaak erg weinig opties open op het speelveld van tijd, geld en kwaliteit. Juist door keuzes, die nog niet gemaakt hoeven te worden, ook niet te maken, houd je zoveel mogelijk opties open. En die opties geven je later veel controle op dat speelveld.

Testen met de klant

Vaak verloopt het acceptatietraject in een project erg moeizaam. Dit omdat gedurende een project het contact tussen business en IT miniem is. Hierdoor maakt IT eigen keuzes bij problemen waarvoor het zich gesteld ziet. De business daarentegen vormt haar eigen gedachten rond het uiteindelijke resultaat. Het spreekt voor zich dat deze twee werelden tijdens het acceptatietraject niet meer bij elkaar kunnen komen, met alle afgeleide problemen van dien.

Beter is het om van het begin af aan samen met de klant te testen. Met deze betrokkenheid vindt acceptatie gedurende het gehele testtraject plaats. De verwachtingen van de klant worden namelijk continu getoetst tijdens de realisatie van het informatiesysteem. Ook zijn bevindingen die normaliter pas in een separaat acceptatietraject gevonden worden, sneller en goedkoper op te lossen. Uiteindelijk zal het acceptatietraject, indien nog nodig, meer een formaliteit zijn.

‘Beter is het om van het begin af aan samen met de klant te testen. Met deze betrokkenheid vindt acceptatie gedurende het gehele testtraject plaats.’

Onder testen met de klant wordt verstaan, dat:

- de klant en IT samen verantwoordelijk voor testen zijn;
- het testproces volledig transparant is;
- de klant beslist wat er getest wordt;
- acceptatie gedurende het gehele testtraject plaatsvindt.

Testen met de klant stelt de klant in staat om zelf keuzes te maken op het speelveld van tijd, geld en kwaliteit. Bovenstaande vorm van interactie met de klant geldt uiteraard niet alleen voor het testen, maar voor het gehele systeemontwikkelingstraject.

Willen excelleren
Testen vormt tegenwoordig een vast onderdeel binnen het systeemontwikkelingsproces. Helaas nemen veel testers genoegen met alleen een kunstje doen op basis van een functioneel ontwerp en een uitgekauwd testplan. Testen is langzaam verworden tot een noodzakelijk kwaad, waarbij de tester niet de mentaliteit heeft testen een meerwaarde te laten zijn. Die meerwaarde was ooit de belangrijkste reden dat testen als volwaardig vakgebied werd geaccepteerd.

Een tester die wil excelleren, neemt verantwoordelijkheid voor het te behalen eindresultaat en is creatief en flexibel in het bereiken hiervan. Hij stapt uit de slachtofferrol waarin het niet (goed) kunnen testen altijd de schuld is van anderen en omstandigheden. Hij weet testen weer de meerwaarde te geven die het ook moet hebben.

‘De tester gaat voor maximaal resultaat en verafschuwt middelmatigheid.’

Onder willen excelleren wordt verstaan, dat de tester:
- weet wat ‘goed testen’ is;
- bijdraagt aan het ontwikkelproces en het niet frustreert;
- verantwoordelijkheid neemt voor het eindresultaat;
- anderen ook stimuleert te excelleren;
- het testvak weer sexy maakt.

Een tester die weet te excelleren heeft een positieve houding en laat zich niet beperken. Niet langer wordt het verschuilen achter allerhande randvoorwaarden en entry criteria getolereerd. De tester gaat voor maximaal resultaat en verafschuwt middelmatigheid.

Exploratory Testing

In het vorige hoofdstuk is beschreven wat binnen testen belangrijk is en dat is verwoord in het Testmanifest. Dit is het vertrekpunt voor een methode en gedachtegoed die invulling kan geven aan dit manifest.

Rond 1990 ontstond parallel aan alle andere ontwikkelingen op testgebied een radicaal andere zienswijze op het gebied van softwaretesten: Context-Driven Testing (CDT). Het uitgangspunt van deze zienswijze is dat testers moeten leren werken op basis van de context. Dit betekent werken onder continue onzekerheid en verandering. Verder geloven de aanhangers van deze zienswijze dat er geen 'best practices' voor het testen van software bestaan. Zij vinden dat testen meer een set van vaardigheden behelst die de tester in staat stelt de meest passende testtechniek te selecteren of een nieuwe wijze van testen te introduceren, afhankelijk van de context waarin een test moet worden uitgevoerd. Iedere situatie is immers uniek.

Het eerste boek waarin geschreven werd over CDT is 'Testing Computer Software' van Cem Kaner uit 1987. In 2001 is het boek 'Lessons Learned in Software Testing, A Context-Driven Approach' door Cem Kaner, James Bach en Bret Pettichord gepubliceerd. De Context Driven School of Software Testing is hier een uitvloeisel van. Zij beschrijven Context Driven Testing als volgt:

'Testers die Context-Driven Testing toepassen kiezen zelf hun testdoelen, technieken en op te leveren producten (inclusief testdocumentatie) op basis van de context en details van die specifieke situatie, inclusief de wensen van de opdrachtgever en eventueel andere belanghebbenden. De essentie van Context-Driven Testing is projectafhankelijk toepassen van vaardigheden.'

Uiteindelijk is Context-Driven Testing het beste doen met wat er tot je beschikking staat. In plaats van het toepassen van 'best practices' accepteer je dat er veel verschillende goede aanpakken bestaan. Iedere aanpak kan, afhankelijk van de context, op dat moment de beste zijn.

'Uiteindelijk is Context-Driven Testing het beste doen met wat er tot je beschikking staat.'

De zeven basiswaarden achter de Context Driven School zijn:

1. De waarde van een aanpak is afhankelijk van de context.
2. Er zijn goede aanpakken in een bepaalde context, maar er zijn geen 'best practices'.
3. Mensen, en de samenwerking tussen deze mensen, zijn het belangrijkste onderdeel voor de context van ieder project.
4. Projecten ontwikkelen op een wijze die vaak niet voorspelbaar is.
5. Het product is een oplossing. Als het product het probleem niet oplost, werkt het product niet.
6. Het goed testen van software is een uitdagend en intellectueel proces.
7. Alleen juiste oordeelsvorming en inzet van vaardigheden, gezamenlijk uitgevoerd gedurende het gehele project, stelt ons in staat de juiste dingen op het juiste moment te doen om op effectieve wijze producten te testen.

Exploratory Testing

In Amerika is exploratory testing als techniek voortgekomen uit de Context Driven School of Testing (CDT). De definitie voor exploratory testing, beschouwd als techniek, is volgens twee leden van de CDT, Cem Kaner en James Bach, als volgt:

'Exploratory testing is een testaanpak die anticipeert op de vrijheid en verantwoordelijkheid van iedere tester om doorlopend de waarde van zijn werk te optimaliseren. Dit wordt gedaan door zowel leren, specificeren en uitvoeren van testen te behandelen als onderling ondersteunende activiteiten die parallel lopen gedurende een project.'

Exploratory testing is een krachtige, uitdagende en iteratieve testaanpak waarbij het uitgangspunt is dat het leren, ontwerpen en uitvoeren van testen zoveel mogelijk gelijktijdig plaatsvindt. In plaats van testspecificaties bestaat de basis voor de toepassing van exploratory testing uit ervaring, domeinkennis, testkennis, creativiteit en logisch denkvermogen. Exploratory testing is een proces van doorlopende optimalisatie met een sterke nadruk op leren en verbeteren.

De methode exploratory testing

Exploratory testing is tot dusverre alleen toegepast als testtechniek. Deze testtechniek geeft al deels invulling aan het Testmanifest. Om deze techniek optimaal te kunnen benutten, is hier in dit boek testmanagement aan toegevoegd, om de techniek ook methodisch in te kunnen zetten. De drie kritieke succesfactoren voor het inzetten van exploratory testing als methode zijn een goede invulling van de rollen, het goed toepassen van de testtechniek binnen het proces en een efficiënt testmanagement.

In de navolgende hoofdstukken worden deze drie succesfactoren nader beschreven.

Rollen binnen Exploratory Testing

Binnen de uitvoering van exploratory testing en de beschreven testsessies zijn drie rollen te onderscheiden, de tester, de klantvertegenwoordiger en de exploratory coach. Elk van deze drie rollen heeft zijn eigen unieke bijdrage voor het krachtig maken van de methode.

Tester
De tester voert het testwerk uit. De tester heeft een goede kennis van de diverse testspecificatietechnieken en kan deze 'blind' toepassen, zonder deze geheel uit te hoeven schrijven.

De tester verdiept zich continu in de context waarbinnen de testen moeten worden uitgevoerd en bepaalt op basis daarvan de testaanpak. Tijdens de tests bedenkt de tester nieuwe testgevallen, voert deze uit en maakt hiervan notities. Hierbij onderhoudt de tester nauw contact met de klantvertegenwoordiger.

'De tester verdiept zich continu in de context waarbinnen de testen moeten worden uitgevoerd en bepaalt op basis daarvan de testaanpak.'

Bevindingen of andere belangrijke testresultaten worden continu en op adequate wijze afgehandeld door de tester.

Klantvertegenwoordiger
De klantvertegenwoordiger heeft een belangrijk aandeel in de testvoorbereiding en testuitvoering. Omdat de klantvertegenwoordiger door de actieve participatie veelvuldig in contact komt met het informatiesysteem, kan hij dit doorlopend valideren en verifiëren. Zo wordt het juiste product conform de klantwens

gerealiseerd. Deze rol kan ingevuld worden door een functioneel beheerder, gebruiker, proces-, toepassings- of materiedeskundige.

‘De klantvertegenwoordiger is iemand die veel kennis heeft van de klantprocessen en de context waarin deze processen worden uitgevoerd.’

De klantvertegenwoordiger is iemand die veel kennis heeft van de klantprocessen en de context waarin deze processen worden uitgevoerd. Het moet iemand zijn die de prioriteit kan bepalen van functionaliteit, testsituaties en bevindingen. Idealiter is een van de klantvertegenwoordigers ook gemandateerd om direct beslissingen te mogen nemen.

Exploratory Coach

De exploratory coach heeft een tweeledige rol. Allereerst is de coach ook de facilitator voor de tests. Hierbij valt te denken aan het faciliteren van testomgevingen, testtools, werkplekken, bemensing en communicatie met de omgeving.

‘De exploratory coach coacht de tester en de klantvertegenwoordiger tijdens het testproces.’

De tweede rol van de exploratory coach is de belangrijkste. Hij coacht de tester en de klantvertegenwoordiger tijdens het testproces. Dit kan bijvoorbeeld door uitdagende vragen te stellen, tips en richting te geven. De coach dient als klankbord en voor procesgerichte coaching.

Deze rol kan door verschillende mensen worden ingevuld. Bijvoorbeeld door een testcoördinator of testmanager, een collega tester of zelfs de klantvertegenwoordiger. De rol op zich behoeft geen hiërarchische status, maar wel kennis van het exploratory testing-proces.

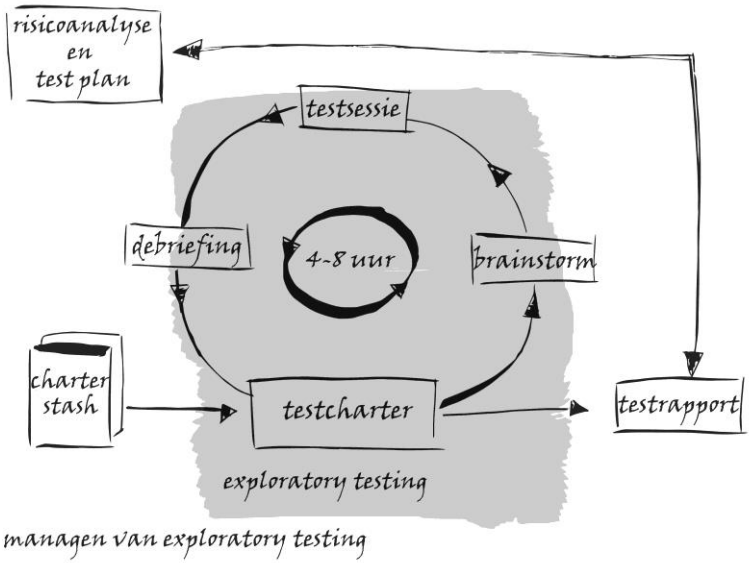
Exploratory Testing-proces

Het volgen van het proces is een van de belangrijkste factoren die exploratory testing krachtig maakt. Het proces biedt ruimte voor belangrijke zaken zoals context denken, kwaliteitsgerichtheid, creativiteit, leren en plezier in het werk. Beter nog, het proces geeft hier handvatten voor. De volgende figuur geeft het gehele proces van exploratory testing grafisch weer, bestaande uit het managementproces en het proces omtrent de testsessie. Dit hoofdstuk beschrijft het standaardproces omtrent de testsessie. Afhankelijk van bijvoorbeeld de ontwikkelmethode of bemensing kan hiervan afgeweken worden.

‘Het proces biedt ruimte voor belangrijke zaken zoals context denken, kwaliteitsgerichtheid, creativiteit, leren en plezier in het werk.’

Testsessie
Exploratory Testing wordt uitgevoerd in testsessies, ook bekend onder de term Session Based Testing (SBT). Een testsessie kenmerkt zich als een afgebakend geheel met een duidelijk beginpunt (charter), vervolgstappen (brainstorm en testuitvoering) en afsluiting (debriefing). Het testen in kort cyclische sessies die veelal binnen een werkdag zijn afgerond, biedt dagelijks veel input en ruimte voor planning, coaching en prioritering.

Startpunt voor het proces van de testsessie is de testcharter. Hoe deze testcharter ontstaat en onderdeel uitmaakt van de charter stash, en hoe de testresultaten uit de sessie verder worden verwerkt, wordt behandeld in het volgende hoofdstuk Managen van exploratory testing.



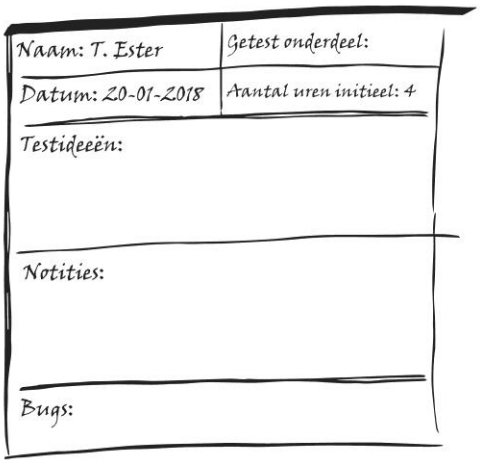
Figuur 4: Exploratory testing, het proces

Testcharter

De testcharter is het werkdocument voor een exploratory testing-sessie op een deel van het informatiesysteem. Het bevat initieel een beschrijving van het te testen gedeelte van de functionaliteit dat eerder in een testaanpak is vastgelegd. Gedurende het proces wordt de testcharter verder ingevuld en is de leidraad voor de testsessie. Aan het einde van de dag geeft de dan volledig ingevulde testcharter een overzicht weer van de testdag, eventuele te nemen vervolgstappen en een nuttige set (test-) managementinformatie. Kort, bondig, overzichtelijk en efficiënt zijn hierbij duidelijke kenmerken, dus geen uitgebreide uitwerkingen van alle testgevallen en de uitkomsten. Dit bespaart veel tijd en geld!

Een testcharter is gericht op een stuk software dat getest moet kunnen worden in ongeveer 4 tot 8 uur. Bij een kortere periode is

de effectiviteit minder doordat de brainstorm en de debriefing een relatief groter deel van de tijd in beslag zullen nemen. Ook is er te vaak input voor de onderdelen planning en voortgang, waardoor dit ook minder effectief wordt. Bij een langere periode zal de testuitvoer minder effectief verlopen, doordat het inzicht in de beschikbare tijd minder is en het deel software te groot, waardoor het overzicht op het nog te testen deel te laag is.



Figuur 5: Testcharter

Brainstorm

De brainstorm is de fase voorafgaande aan de testsessie waarbij er ideeën worden opgedaan wat en hoe de betreffende functionaliteit getest moet worden. Allereerst worden op high level de logische testpaden bepaald, bijvoorbeeld door het tekenen van een dataflow. Hierbij kan een ontwerpdocument een ondersteuning zijn voor het opstellen van de testideeën en de logische testpaden, maar dit is niet absoluut noodzakelijk.

Alle ideeën dienen als kapstok voor de testuitvoering. Daarnaast wordt er een lijst opgesteld met software-gerelateerde vragen die door middel van de testuitvoer beantwoord dienen te worden. Deze vragen kunnen hun oorsprong vinden in de context waarbinnen het testobject gebruikt zal worden, maar ook op het niveau van bedrijfsprocessen en gebruikersinteractie. Binnen de testcharter kan de klantvertegenwoordiger waar nodig ook de verschillende vragen prioriteren.

De brainstorm wordt gedaan door de tester en daarbij eenieder die zinnige input kan leveren voor de test. Bijvoorbeeld de klantvertegenwoordiger om het proces wat met de software wordt ondersteund toe te lichten of de coach om te helpen ideeën op te doen. Maar denk bijvoorbeeld ook aan een ontwikkelaar die de technische impact goed kan beschrijven, aangezien deze nog weleens deels af kan wijken van de functionele impact. Een indicatie voor de tijdsduur van een brainstorm is 1/8 deel van de benodigde tijd voor die testsessie.

Per testsessie wordt door de tester bekeken wat beste aanpak is, gebaseerd op het belang en de risico's van dit deel van de software voor de klant en de ervaringen tot dan toe. Hierin zit een essentieel verschil met de huidige, sterk gefaseerde methoden. Daarin wordt voorafgaande aan de test in een plan al bepaald wat de aanpak, de verschillende testfases, de diepgang en de keuze voor testspecificatietechnieken is.

Testuitvoering

Tijdens de testuitvoering worden de in de brainstorm opgestelde testvragen beantwoord en de logische testgevallen uitgevoerd, met daarnaast voldoende mogelijkheid om afgeleid te worden op zoek naar potentiële bevindingen, risico's en bedreigingen. Streven is tijdens een sessie om de tijd van een tester zo veel mogelijk te investeren in testuitvoering en zo minimaal mogelijk in documenteren. Wees creatief in minimaal documenteren.

De testuitvoer wordt bij voorkeur gedaan op basis van pair testing. Dit is een situatie waarbij idealiter de tester en de klant-vertegenwoordiger samen testen, waarbij de tester de 'knoppen bedient' en de klantvertegenwoordiger aantekeningen verzorgt en materiekennis aandraagt. Voordelen van pair testing zijn dat er meer plezier is waarbij je elkaar stimuleert, de drempel om een van beiden te storen hoger ligt en het eenvoudige principe van twee zien en weten meer dan een van toepassing is. En in deze specifieke combinatie is er zowel veel testkennis als materiekennis betrokken bij de tests. De testuitvoer zou ongeveer 6/8 deel van de beschikbare tijd in beslag moeten nemen.

'Wees creatief in minimaal documenteren.'

De testuitvoering bestaat uit een continu proces van het doorlopen van drie stappen: leren, specificeren en uitvoeren.

Leren
Gedurende de testuitvoering groeit het inzicht hoe de software reageert, ook ten opzichte van de context, hoe deze met bepaalde situaties om gaat en wat niet en vooral ook juist wel kan. Ook groeit het inzicht hoe de businessprocessen in elkaar steken (mede door de ondersteuning van de klantvertegenwoordiger) en welke aansluiting de software daarop heeft.

Specificeren
Het leren kennen van de software leidt tijdens de testsessie tot nieuwe ideeën voor testgevallen en geeft inzicht in bedreigingen en risico's die al testend verder in kaart gebracht moeten worden. Ook kunnen hier geleerde zaken of ontstane vragen uit de testuitvoer worden omgezet in nieuwe testgevallen.

Uitvoeren
Testgevallen en testideeën moeten uitgevoerd worden. Belangrijk hierbij is dat de reactie van de software bepaalt hoe je verder gaat, eigenlijk ben je dan weer aan het leren en specificeren. Zo blijft de cyclus van leren, specificeren en uitvoeren zichzelf continu herhalen.

Tijdens de uitvoering is het handig dat iemand relevante aantekeningen maakt, die later weer uitgewerkt kunnen worden tot nieuwe testgevallen. De aantekeningen bevatten ook notities over voortgang, zaken waar later nog naar gekeken moet worden en informatie die van belang kan zijn voor andere testcharters. Deze kunnen dan later worden opgenomen in de bijbehorende charters en meegenomen worden in de testsessie van die charters.

‘Belangrijk hierbij is dat de reactie van de software bepaalt hoe je verder gaat, eigenlijk ben je dan weer aan het leren en specificeren.’

Bij het constateren van hoog geprioriteerde bevindingen is het aan te bevelen ook direct een alternatief te onderzoeken. De testers zijn immers met het betreffende stuk software bezig en op deze manier hoeft niet later nog nagedacht te worden over work-arounds. Naast tijdswinst biedt dit een goede achtergrond om beslissingen over de werkelijke ernst van een bevinding te nemen.

De testuitvoering is ten einde wanneer de geplande tijd voor een testsessie is verstreken. Tweede optie is dat door de tester en de klantvertegenwoordiger samen is besloten dat er voldoende is getest. Tijdens de debriefing kan al dan niet worden besloten op een ander moment een vervolg te geven aan deze testsessie of een nieuwe testsessie te beginnen op basis van een nieuwe testcharter.

Bij de testuitvoering moet optimaal gebruik gemaakt worden van de 7 basisprincipes die ten grondslag liggen aan het testen. Zo is testen altijd context afhankelijk en toont het fouten aan. Door het vroegtijdig betrekken van de klant, worden grote omissies in de bruikbaarheid van het informatiesysteem al snel ontdekt. Dit scheelt aanzienlijk in de kosten voor herstel.

Indien een fout gevonden wordt, stelt de cyclus van leren, specificeren en uitvoeren ons in staat op basis hiervan met nieuwe testgevallen te komen. We weten immers dat fouten vaak geclusterd zijn. Waar een fout gemaakt wordt, leidt dat vaak tot vervolg- of samenhangende fouten.

Bij het opnieuw testen van hetzelfde testobject, al dan niet als gevolg van foutherstel, wordt per definitie de test op een andere wijze uitgevoerd. Dit verhoogt de dekkingsgraad van de tests voor dat gedeelte en voorkomt waardeverlies door steeds dezelfde test uit te voeren.

Alles testen is per definitie onmogelijk, maar door de klant-betrokkenheid kan de prioritering op basis van de testresultaten continu worden bijgesteld zodat de beschikbare middelen optimaal worden benut.

Debriefing
De debriefing is het einde van een testsessie. Bevindingen worden geregistreerd en de alternatieven worden besproken. Ook wordt van de initieel tijdens de brainstorm opgestelde testvragen bekeken of deze beantwoord zijn. Er wordt een uitspraak gedaan of dit deel software vanuit het oogpunt van de testers productie gereed is, of er nog openstaande zaken zijn en of er aanvullende charters gemaakt dienen te worden voor een nieuwe testsessie. De coach bespreekt met de tester en eventueel de klantvertegenwoordiger de testsessie. Het principe van laten zien wat je die dag hebt gedaan, motiveert testers om goed voorbereid voor de dag te

komen. Vanzelfsprekend leidt dit tot een verbetering van de kwaliteit van het werk en de effectiviteit.

Voorbeelden van interessante vragen tijdens een debriefing zijn:

- Zijn de belangrijkste bevindingen gevonden? En hoe weet je dat?
- Dienen bepaalde onderdelen nog verder getest te worden?
- Wat zijn de overgebleven risico's en bedreigingen in dit deel van de software en wat zijn de alternatieven?
- Wat is de belangrijkste of leukste gevonden bevinding van deze testdag?
- Is dit deel van de software van voldoende kwaliteit om in productie te nemen?
- Zijn er zaken die vanuit de testsessies verbeterd kunnen worden?

De debriefing zou maximaal 1/8 van de beschikbare tijd voor een charter in beslag mogen nemen.

Managen van Exploratory Testing

Het gehele testproces laat zich met exploratory testing zeer flexibel managen. Allereerst begin je met het schrijven van een testaanpak dat de kaders voor het gehele testtraject beschrijft. Daarnaast kan er op basis van de charter stash op ieder gewenst moment een heldere rapportage worden gegeven over de actuele stand van zaken in het testtraject en daarmee is de rapportage een feit.

Testaanpak
De testaanpak heeft bij exploratory testing een andere inhoud dan een testplan binnen meer traditionele methoden. Er is behoefte aan een duidelijk overzicht en inzicht in wat er getest moet worden en de relatie die dat heeft met tijd, geld en kwaliteit. Daarom wordt in het plan de nadruk gelegd op een totaalbeeld van het testobject, wat samen met de doelstelling en context een helder beeld geeft van het speelveld.

‘De testaanpak zegt niets over hoe in detail getest dient te worden.’

De testaanpak zegt niets over hoe in detail getest dient te worden. Daarmee bevat de testaanpak dus geen teststrategie. Dit is ter bepaling van de tester indien een component getest gaat worden, binnen de op dat moment actuele kaders van tijd, geld en kwaliteit.

De testaanpak bestaat uit de volgende onderdelen:
- Doelstelling;
- Functionele decompositie;
- Risicoanalyse en prioritering;
- Planning, resourceplanning en begroting.

Uiteraard kunnen er ook andere zaken als de testinfrastructuur of overlegstructuur in de testaanpak worden opgenomen. Het streven blijft een beknopt en vooral handzaam document op te stellen.

Doelstelling
Het formuleren van de doelstelling met de afbakening en de randvoorwaarden moet een helder beeld geven van het belang van het informatiesysteem voor een organisatie en de context waarbinnen dit wordt gebruikt.

Na goedkeuring van de testaanpak wordt er bij wijzigingen alleen een nieuw plan opgeleverd als de doelstelling wijzigt. De testrapportage voorziet in de status van alle andere, dynamische onderdelen van de testaanpak.

Functionele decompositie
Het te testen informatiesysteem wordt opgedeeld in componenten die logisch en functioneel samenhangen. Van belang bij deze decompositie is dat de componenten uit eenheden bestaan waarop een risicoanalyse uitgevoerd kan worden en bij voorkeur binnen een sessie van 4 tot 8 uur getest kan worden. Het is zeer wenselijk deze compositie niet (alleen) door een testmanager of –coördinator uit te laten voeren, maar hierbij ook testers en eventueel klantvertegenwoordigers te betrekken. Zij zijn immers degenen die die componenten later in een testsessie moeten testen. Daarnaast wordt er op deze wijze draagvlak voor de planning gecreëerd.

Onder een functionele decompositie verstaan wij ook het benoemen van alle overige relevante systeemeisen (non-functionals) zoals gebruiksvriendelijkheid en performance. Deze kunnen als onderdeel binnen bestaande charters worden belegd, of als afzonderlijk onderdeel gecharterd worden. Uiteraard kunnen hier later in het traject aanvullingen op worden gedaan.

Het is in de meeste gevallen zinvol om ook de requirements aan de verschillende componenten te koppelen. Dit vergemakkelijkt

risicoanalyse en prioritering. Initieel kunnen nu de eerste versies van de benodigde testcharters worden opgesteld. Deze charters kunnen vervolgens doorlopend aangevuld worden met nieuwe relevante informatie.

Risicoanalyse en prioritering
Op basis van de functionele decompositie kunnen alle componenten in een risicoanalyse samen met de klant worden ingedeeld en vervolgens geprioriteerd. Na deze analyse is de initiële prioriteit van de componenten voorlopig bepaald. In volgordelijkheid van prioriteit worden de testsessies aangevangen en uitgevoerd, uiteraard ook afhankelijk van de beschikbaarheid van een te testen component.

Planning, resource planning en begroting
Net als bij de risicoanalyse en prioritering vormen bij het plannen en begroten de componenten ook een belangrijke rol. Het aantal componenten bepaalt het initiële aantal testsessies om alles te testen.

Aan iedere testcharter is een aantal uren gekoppeld op basis van het ingeschatte risico van een component waarmee een eerste begroting een feit is. Het geplande aantal uren en de functionele decompositie is idealiter medebepaald door de testers.

Voor her- en regressietesten, wijzigingen en aanvullingen in de requirements en hiervan afgeleide functionele wijzigingen is niet direct een aantal uren begroot. Maar dit zijn wel zaken waar in de planning rekening mee moet worden gehouden. Door de initiële begroting te vermenigvuldigen met factor 1,5 wordt hierin voorzien. Ook aanvullend te charteren functionaliteit welke in de debriefing naar voren kan komen valt hieronder. De factor is een ervaringscijfer dat initieel kan worden gebruikt. Deze factor is afhankelijk van de volwassenheid van de organisatie, van de kwaliteit van de software, van de competenties van de beschikbare testers en is inclusief de uren van de exploratory coach. Uren voor

klantvertegenwoordigers zijn hierin niet meegenomen, deze kunnen bepaald worden naar gelang hun inzet. Naarmate meer ervaring is opgedaan kan deze factor op basis van resultaten eventueel worden bijgesteld.

Voor de planning en resourceplanning zijn twee variabelen van belang, het aantal testers en de geplande opleverdatum. Het aantal uren is gemakkelijk uit te zetten in de tijd. Mocht dat de opleverdatum overschrijden dan zijn er opties om het aantal benodigde testers te verhogen, tijd per charter te verlagen of om met minder geteste componenten toch in productie te gaan. Door de heldere structuur van exploratory testing kan er een bewuste keuze gemaakt worden om langer door te gaan met testen, meer budget voor testers beschikbaar te stellen of delen niet of minder uitgebreid te testen.

Testrapportage

De testrapportage is een voortgangsrapport waarbij de laatste versie van dit rapport op ieder moment als eindrapport kan dienen. De dynamische onderdelen uit de testaanpak vormen een onderdeel van dit testrapport. De initiële planning, prioritering, risicoanalyse en charter stash vormen daarmee de basis van het eerste testrapport.

‘De testrapportage is een voortgangsrapport waarbij de laatste versie van dit rapport op ieder moment als eindrapport kan dienen.’

Het is een beknopt en zeer actueel overzicht van alle testcharters, zowel geteste alsnog te testen. Voor iedere nog te testen charter wordt inzicht gegeven in:

- Naam van de component;
- Geschatte benodigde tijd;

- Risicoprofiel en prioriteit;
- Testbaarheid.

Dit is dus tevens je voortgangsrapportage. Hier staat alles in: prioriteit, voortgang, resterende uren, bestede uren, verbrande geplande uren en daarmee ook de afwijking t.o.v. de initieel afgegeven planning. Dit overzicht kan indien relevant nog worden aangevuld met bevindingen, requirements, naam van de tester(s), maatregelen, etc.

Debriefing
Tijdens elke debriefing komt er informatie binnen over geteste componenten. Zo wordt de testers door de exploratory coach in ieder geval gevraagd naar:

- Verloop: hoe is de testsessie verlopen?
- Resultaten: zijn alle testvragen beantwoord?
- Risicogebieden: is er aanvullend te charteren functionaliteit?
- Obstakels: wat beperkte de testsessie en waarom?
- Bevindingen: zijn alle bevindingen geregistreerd en zijn alle alternatieve scenario's benoemd en getest?
- Tijd: moet er aan deze charter nog meer tijd worden besteed?
- Gevoelens: mening van de tester of de geteste component van voldoende kwaliteit is om in productie gebruikt te kunnen worden.

Uiteraard gaat het laatste punt over het gevoel van de tester bij de uitgevoerde sessie heeft opgebouwd. De tester geeft slechts zijn gevoel weer. Door toch direct deze vraag te stellen wordt er over het algemeen beter nagedacht alvorens een antwoord te geven. En exploratory testing wil graag de kennis, kunde en het intellect van de tester uitdagen om tot de beste resultaten te komen.

Alle terugkoppeling verkregen in een debriefing kan verwerkt worden in de testrapportage. Hiermee wordt direct de planning en de charter stash bijgewerkt. Door deze informatie te verwerken wordt naast een actueel beeld van de status direct zichtbaar wat dit

betekent voor de planning en urenbesteding. Met deze informatie kunnen toekomstige sessies, de planning en prioritering dan weer worden bijgesteld en verbeterd, zodat toekomstig handelen altijd is gebaseerd op actuele gegevens.

Met de testrapportage is het mogelijk om altijd direct een actueel overzicht te bieden van de status van het informatiesysteem dat wordt getest. Het biedt tevens de juiste informatie om gefundeerd een uitspraak te doen over de kwaliteit van een informatiesysteem als met testen moet worden gestopt. Het zorgt er samen met de charter stash voor dat de best mogelijke test is uitgevoerd binnen de beschikbare tijd en met de beschikbare testers.

Burn down chart
Het kan handig zijn de testrapportage te voorzien van een grafische weergave in de vorm van een burn-down chart. In deze grafische weergave is het testverloop in uren of aantal testcharters uitgezet tegen de beschikbare tijd. Hiertussen wordt een nominale lijn getrokken die dan de gemiddelde benodigde voortgang aangeeft om de testen tijdig af te ronden. Dagelijks wordt het aantal afgeronde tests in dit overzicht bijgewerkt en kan er op basis van het aantal uren dat nog te gaan is, de lijn met werkelijke voortgang worden bijgewerkt in de grafiek. Zo worden afwijkingen direct zichtbaar ten opzichte van de nominale lijn. Er wordt dus vanaf testdag één gepland en bijgestuurd op geconstateerde voortgang in plaats van schattingen.

Bijsturen
Indien de lijn met de werkelijke voortgang in de burn-down-chart boven de nominale lijn van benodigde voortgang ligt (m.a.w. de werkelijke voortgang is lager dan die gemiddeld nodig zou zijn om tijdig klaar te zijn met testen), zijn er door de flexibiliteit van exploratory testing diverse bijsturingsmogelijkheden. Het speelveld van tijd, geld en kwaliteit is daadwerkelijk een speelveld geworden, waarbinnen maximale bewegingsvrijheid bestaat. Zoals bijvoorbeeld:

- Einddatum opschuiven;
- Meer resources beschikbaar stellen;
- Geplande uren van charters verlagen met X%;
- De geplande uren van de laagst geprioriteerde charters verminderen met X%;
- Charters herprioriteren.

‘Het speelveld van tijd, geld en kwaliteit is daadwerkelijk een speelveld geworden, waarbinnen maximale bewegingsvrijheid bestaat.’

Voordeel van het managen op deze manier is dat veel eerder in het traject zichtbaar wordt of het testen nog op schema ligt. En indien er veel eerder bijgestuurd kan worden, zijn de mogelijkheden hiervoor beter en krachtiger. Indien gesneden wordt in tijd van een charter, waarborgt de tester die die charter uitvoert toch dat in de beschikbare tijd het best mogelijke getest wordt. Hierdoor worden automatisch de minst belangrijke tests geschrapt. Ook bijsturen door het inzetten van extra testers is door een inwerktijd vaak alleen effectief indien dit in vroegtijdig in een traject wordt ingezet.

Rol van de testcoördinator en testmanager
Een testcoördinator of testmanager heeft binnen exploratory testing een grotendeels zelfde rol als binnen traditionele methoden. De testaanpak moet worden opgesteld en over de voortgang en potentiële risico’s moet worden gerapporteerd. Wel is het zo dat er minder specifieke managementaspecten zijn en meer coaching, sturing, facilitering en ondersteuning bij de uitvoering nodig is.

Eén van de belangrijkste verschillen tussen exploratory testing en de traditionele testmethoden is de nadrukkelijke focus die exploratory testing legt op de vaardigheden van de testers en

minder op de methode en het proces dat wordt gebruikt. Er vindt zodoende ook een verschuiving plaats van traditioneel managen van een testproject naar het coachen van de testers.

En juist op dit punt heeft een testcoördinator of -manager een zeer belangrijke rol. Door het structureel goed coachen van de testers, zal hun kennis en kunde sterk toe nemen. En deze kennis en kunde zal in de daaropvolgende tests de kwaliteit en snelheid van de testuitvoering alleen maar verhogen.

Verschillen
Het managementproces dat in dit hoofdstuk is beschreven is geen hogere wiskunde. Juist niet! Een testtraject managen met een overzichtelijk testrapport en een grafiek is precies dat wat het moet zijn: helder en effectief. Het biedt controle over testen.

‘Een testtraject managen met een overzichtelijk testrapport en een grafiek is precies dat wat het moet zijn: helder en effectief. Het biedt controle over testen.’

De winst van het managen op deze wijze ten opzichte van het traditioneel testmanagement is de inzichtelijkheid in de voortgang, eerdere en meerdere mogelijkheden voor bijsturing en geen verlies door de bijsturing. Hiermee wordt bedoeld dat er geen volledig gespecificeerde testgevallen niet worden gebruikt, waardoor met deze voorbereiding tijd is verspild, die anders aan andere tests gespendeerd had kunnen worden.

Deze manier van managen biedt de testers de mogelijkheid hun potentieel volledig te benutten en zich te ontwikkelen. Dit in plaats van ze te beperken met criteria, uitgekauwde aanpakken en een door de manager opgedrongen planning en strategie.

Reviewen binnen Exploratory Testing

Belangrijk bij exploratory testing is het helder hebben van de context. Het helder hebben van de context is net zo belangrijk bij het reviewen van documenten. Maar omgekeerd helpt het reviewen van documenten ook bij het helder krijgen van de context. Vanwege deze gecombineerde meerwaarde, beschrijven we in dit hoofdstuk hoe te komen tot een kwalitatief hoogwaardige review. Denk hierbij bijvoorbeeld aan het reviewen van functionele en technische specificaties, procesontwerpen, wetsteksten etc. Aan reviewen wordt helaas vaak weinig tijd en aandacht besteed, terwijl dit kan helpen met het leggen van een zeer goede fundering voor je systeemontwikkeling en testuitvoering.

Maar hoe doe je als tester nu een goede review?

Kaders
Let in eerste instantie op de kaders van het document. Welk doel heeft het document en wat wil de schrijver er mee bereiken? Wat zijn belangrijke kwaliteitsattributen voor dit document? Betreft het gebruikersdocumentatie of een handleiding, dan zal bruikbaarheid erg belangrijk zijn. Bij een functioneel ontwerp zal juistheid de belangrijkste factor zijn. Neem de kaders van het document als uitgangspunt voor je review.

Publiek
Daarnaast is het publiek voor wie een document is geschreven van belang bij een review. Het kan je eigen kijk op het document beïnvloeden en een rol spelen bij het verzamelen van mensen en kennis rondom de review.

Context
Naast het publiek en de kaders, welke eigenlijk ook onderdeel zijn van de context van het document, is een goede bepaling van overige contextelementen essentieel. Bepaal de relatie van het te

reviewen document met bijvoorbeeld een datamodel, werkproces, use cases, functionele ontwerpen, wetstekst, globaal ontwerp.

Informatie verzamelen
Nadat je als tester de context van het document helder hebt, kun je beginnen met het verzamelen van de informatie die nodig is voor de review. Bevraag het beoogde publiek, verzamel relevante andere documenten, verdiep je in de beschreven materie.

Reviewen inhoud
Gebruik de bepaalde context en alle verzamelde informatie actief in de review. Betwijfel elke zin en elk diagram op een positief kritische manier. Bij de beschrijving van bijvoorbeeld een proces of functie, review deze alsof het reeds gecodeerd was en je het zou testen. Levert bepaalde input de verwachte output op papier. Waar komt de input vandaan? Kan alle mogelijke input verwerkt worden? Zijn alle velden aanwezig, voldoende groot, numeriek of alfanumeriek? Kan alle mogelijke output worden doorgegeven aan een volgende functie of proces? Is niet alleen de happy flow beschreven? Krijg ik het gewenste resultaat als ik de beschreven stappen uitvoer? Sluit het detailontwerp goed aan op het globaal ontwerp? Staat er teveel beschreven en/of is er informatie redundant met andere documenten?

‘Reviewen moet iets totaal anders zijn dan een document een keer doorlezen. Voor een goede review is het belangrijk dat er tijd en moeite in wordt geïnvesteerd.’

Op deze manier zijn er tientallen vragen te bedenken, afhankelijk van welk type document er wordt gereviewed. Wat heel erg belangrijk is, is dat er heel veel waardevols uit een review gehaald kan worden. Fouten kunnen op deze manier in een vroeg stadium

worden ontdekt. Reviewen moet iets totaal anders zijn dan een document een keer doorlezen. Voor een goede review is het belangrijk dat er tijd en moeite in wordt geïnvesteerd. Een investering die zich zeker terugbetaalt!

Niet beschreven

Denk ook een tijd na over inhoud die niet in het document staat. Is het terecht dat dit niet in het document is opgenomen. Welke informatie zou je als publiek willen halen uit het document? Staat dat vermeld?

Relatie van Exploratory Testing en het Testmanifest

In het hoofdstuk Testmanifest een aantal hoofdstukken terug is het Testmanifest beschreven met daarin de punten waar het testen om draait. In deze hoofdstukken is beschreven wat exploratory testing is, welke rollen hierbij te onderkennen zijn en hoe dit goed gemanaged kan worden. Maar in hoeverre voldoet exploratory testing nu aan de principes die verwoord zijn in het Testmanifest?

Inzet van intellect boven methoden, technieken en tools
Exploratory testing wordt in dit boek beschreven als een testmethode. Toch is de inzet van het intellect belangrijker. Indien de testmethode exploratory testing niet of maar deels voldoet voor het doel van de test, dien je deze testmethode natuurlijk niet te gebruiken. Val dan terug op de uitgangspunten.

‘Exploratory testing is een goede en vaak bruikbare methode, maar moet ook zelf altijd ter discussie worden gesteld.’

Wat wel gezegd kan worden is dat het exploratory-testers veel mogelijkheden en ruimte biedt om hun intellect in te zetten en hen daarmee te laten bepalen hoe ze gaan testen en welke technieken en tools daarvoor het best geschikt zijn. Testers kunnen zelf de volgens hen en voor hen beste strategie bepalen. Exploratory testing is een goede en vaak bruikbare methode, maar moet ook zelf altijd ter discussie worden gesteld.

Controle over testen boven vasthouden aan een testplan
Bij het managen van exploratory testing wordt aangegeven dat het goed is dat er een testaanpak wordt opgesteld. De mate waarin er aan deze aanpak wordt vastgehouden is anders. Ten eerste is een

testaanpak binnen exploratory testing voor een heel groot deel dynamisch. Dit houdt in dat het vaak wordt bijgesteld en scherp gesteld aan nieuwe inzichten en reeds behaalde resultaten. Het plan is dus up-to-date. Ten tweede is het globaler van opzet, waardoor het aantal onjuist voorspelde zaken in het plan aanzienlijk lager is en andere zaken daardoor niet op onjuiste informatie zijn afgestemd.

Vasthouden aan een plan is voornamelijk vervelend indien dit vooraf tot in detail is vastgelegd en naarmate het traject vordert er steeds meer zaken afwijken van wat in het plan was beschreven. Indien toch wordt doorgegaan met het naleven van de rest van dat plan, zijn zaken niet meer op elkaar afgestemd. Indien object A getest moet worden, met x aantal mensen, in t tijd verwordt dat uiteindelijk tot een bepaalde aanpak. Indien bijvoorbeeld de beschikbare tijd veel minder blijkt te zijn dan vooraf gepland, is de aanpak dus niet meer optimaal. Het toch blijven volgen van de in het testplan beschreven aanpak is dan niet de beste oplossing en maakt dat de controle over het testen wordt verloren. Daarnaast leert de ervaring dat wijzigingen later niet meer worden opgenomen in het testplan, waardoor het gehele document waardeloos wordt.

Testen met de klant boven acceptatie aan het eind

Binnen exploratory testing is zelfs een aparte rol beschreven voor de klant. Exploratory testing prefereert testen met de klant zeer sterk en hierin is in de opzet van de methode dan ook rekening mee gehouden. Tijdens de brainstorm, testsessie en debriefing is een duidelijke en belangrijke functie beschreven voor de klant en het belang van zijn deelname al vroeg in het testtraject (en beter nog: in het gehele ontwikkeltraject) is geheel onderkend. Exploratory testing biedt ruimte voor een oplossing van de problemen die al jarenlang ontstaan in bijna ieder acceptatietraject doordat er pas helemaal aan het eind voor het eerst geaccepteerd wordt. Acceptatie is geen fase, maar een constant proces dat parallel dient

te lopen aan tenminste het testtraject maar het liefst aan het gehele ontwikkeltraject.

Willen excelleren boven middelmatigheid
Exploratory testing kan middelmatigheid natuurlijk nooit helemaal voorkomen. Exploratory testing stimuleert testers wel om uit het bestaande middelmatige denken en doen te stappen en zich te gaan ontwikkelen. Met het middelmatige denken en doen wordt bedoeld het testgevallen afleiden vanuit een detailontwerp, die vervolgens uitvoeren en dan met een zoek-de-verschillen spelletje bevindingen constateren en opvoeren. Exploratory testing beoogt dat je zelf nadenkt over datgene wat je test, hoe je dat test en legt de verantwoordelijkheid direct bij de tester.

‘Exploratory testing beoogt dat je zelf nadenkt over datgene wat je test, hoe je dat test en legt de verantwoordelijkheid direct bij de tester.’

Hierin komt duidelijk naar voren dat de tester alle ruimte krijgt zijn intellect te benutten. De tester kan per testsessie zelf de aanpak invullen ten aanzien van technieken, testbasis, tools en datgene wat gedocumenteerd wordt.

Om testsessies zo goed mogelijk te laten verlopen, is een proces van continue optimalisatie nodig. Testers die willen excelleren geven die continue optimalisatie vanzelf vorm.

Exploratory Testing en Lean

Lean is een mindset voor procesoptimalisatie, die sterk aan populariteit wint. In steeds meer organisaties worden principes uit Lean toegepast. Ook in IT-organisaties worden deze principes veelvuldig toegepast om het software ontwikkelproces te optimaliseren. We spreken daarbij over Lean Software Development.

Door een proces te matchen aan de principes van Lean, kun je constateren hoe optimaal een proces is. Onderstaand een aantal principes van Lean in relatie tot exploratory testing, waarmee duidelijk wordt dat het toepassen van de methode exploratory testing een geoptimaliseerd testproces oplevert.

1. Elimineer verspilling op gebied van:

   Talent
   Door gebruik te maken van exploratory testing als testmethode haal je veel meer uit het talent van je testers. Je geeft hen veel vrijheid en handvatten om hun creativiteit, analytisch vermogen, logisch denkvermogen, ervaring en testkennis optimaal te benutten en uit te breiden.

   Overproductie
   Te vaak wordt de vraag van de klant overschat en wordt er meer en uitgebreidere software ontwikkeld dan de klant daadwerkelijk nodig heeft. Doordat de klant eerder in het testproces wordt betrokken, kan deze overproductie eerder gesignaleerd worden.

   Correctie
   Onvoldoende testen voor in productie name en terugkerende programmeerfouten zijn hier voorbeelden van. Doordat er minder gespecificeerd wordt, wordt er meer getest voor in

productie name. Terugkerende programmeerfouten zullen ook eerder worden ontdekt, doordat in het leerproces de mogelijkheid is direct te reageren op zwakkere punten in de programmatuur.

Overbewerking
Overbewerking uit zich bijvoorbeeld in veel te uitgebreide documentatie. Exploratory testing beperkt juist de hoeveelheid testdocumentatie sterk en kan goed omgaan met minder uitgangsdocumentatie. De testmethode maakt zich daardoor veel flexibeler en efficiënter.

2. Specificeer waarde vanuit de klant
   Binnen Lean is de gedachte dat alles wat je doet, van waarde moet zijn voor de klant. Binnen exploratory testing werk je samen met de klant en test je waar mogelijk ook met de klant. Dit maakt het eenvoudig om veel te communiceren over wat de klant waardevol vindt, wat het beoogd gebruik is en waar de echte business risico's liggen.

3. Werk aan een verbetering in prestaties en werkplezier, zowel op individueel als op teamniveau
   Hierin speelt exploratory testing een grote rol. De performance van het testteam neemt sterk toe. In dezelfde hoeveelheid testtijd wordt er veel meer getest dan in diezelfde tijdsperiode met de traditionele testmethoden. Daarnaast leert de ervaring dat testers meer zinvol bezig zijn en worden uitgedaagd in hun werk.

4. Werk aan een verbetering in houding en gedrag
   Veelal is de huidige houding van testers reactief te noemen. Wat niet in de specificaties staat, wordt niet getest. Dat specificaties nog niet af, niet duidelijk zijn of maar blijven wijzigen, zijn vaak gehoorde excuses voor lange doorlooptijden bij het testen. Met exploratory testing is dit anders. Testers hebben veel minder belang bij uitgekauwde specificaties, omdat hun testbasis hier

voornamelijk wordt gevormd door kennis, ervaring en input van de klantvertegenwoordiger. Alle kennis is daarmee ter plekke beschikbaar. Wijzigingen in specificaties kosten nog steeds tijd, maar de aanpassingen in de documentatie van testers zijn minimaal. De wijziging moet vervolgens alleen nog daadwerkelijk getest worden.

Exploratory Testing en Scrum, de ideale match

In voorgaande hoofdstukken hebben we beschreven hoe het proces en het managen van exploratory testing eruit kan zien. Maar indien je de methode gebruikt binnen de systeemontwikkelingsmethode Scrum zijn er uiteraard een aantal zaken die anders zijn. Temeer omdat Scrum in een aantal van de onderdelen al voorziet.

Rollen
Indien exploratory testing wordt toegepast binnen Scrum verandert niet zo veel aan de gedefinieerde rollen op zich. Wel is er voor het ontwikkeltraject al een klantvertegenwoordiger aanwezig in de rol van product owner. De product owner zou daarmee ook kunnen fungeren als klantvertegenwoordiger bij het testen, wat een goede combinatie biedt. De product owner is op deze wijze nog nauwer betrokken bij de ontwikkelde software en de kwaliteit en bruikbaarheid daarvan.

Proces
In het proces van testuitvoering zal ook niet veel veranderen. Aanbevelenswaardig is uiteraard om testcharters rechtstreeks te koppelen aan product backlog items (PBI) om zodoende een heldere relatie te houden tussen een testsessie en bijvoorbeeld een user story. Ook is op die manier in dezelfde administratie terug te vinden wat er voor een user story is getest.

Daarnaast zullen er wellicht wat accentverschuivingen zijn in de directe betrokkenheid van de klantvertegenwoordiger bij een testsessie, aangezien de klantbetrokkenheid bij het gehele ontwikkeltraject al een stuk hoger is. Hierdoor zal er al veel informatie over bruikbaarheid en klantproces in het team en bij de tester aanwezig zijn. Je zou in dergelijke situaties de klantvertegenwoordiger bijvoorbeeld alleen bij de debriefing kunnen betrekken.

Managen
Het managen van een exploratory testing-traject wordt binnen Scrum behoorlijk anders en vooral eenvoudiger. Dit omdat de ontwikkelmethode Scrum al voorziet in een aantal zaken die in het hoofdstuk Managen van exploratory testing zijn beschreven.

Het maken van een functionele decompositie is reeds gebeurd bij het creëren van de backlog met de diverse userstories en tasks. Hierop kan prima worden ingehaakt door deze eenheden te gebruiken voor de testsessies.

Daarnaast is de backlog een geprioriteerde set met werk, waarbij een vertegenwoordiging van de business deze prioriteiten heeft bepaald en daarnaast requirements en acceptatiecriteria heeft toegevoegd aan de PBI's. Dit maakt een aparte prioritering en risicoanalyse vanuit testperspectief overbodig. Eventuele PBI-specifieke risico's zullen indien nog niet onderkend bij de brainstorm alsnog boven tafel komen.

Ook is een aparte testplanning niet meer nodig, aangezien al het werk wordt ingepland in sprints, inclusief het testwerk. Deelname van de testers aan de planningssessies borgt het planningsaspect al.

Naast dat Scrum voorziet in een burn-down-chart voor de voortgang inclusief het testwerk, hoeft er ook niet apart over het testwerk gerapporteerd te worden. Voortgang wordt dagelijks besproken in de stand-up en problemen worden gerapporteerd als impediments/blokkades bij de Scrummaster.

Wat overblijft
Er valt dus met Scrum veel werk weg bij het testen, met name op het gebied van het managen van de tests. Toch blijven er naast de invulling van de rollen en het proces rondom testuitvoering, ook nog wat zaken over die gemanaged moeten worden.

Het blijft erg belangrijk om een testaanpak op te stellen. Deze testaanpak bevat een beknopte omschrijving van het testproces zoals dat per PBI wordt doorlopen, wellicht nog met onderscheid per type PBI. Ook een beschrijving van waar eventuele testware geborgd moet worden, hoe wordt omgegaan met regressietests en impactbepaling zijn goede onderwerpen om toe te voegen. Indien gewenst kan een aanpak worden beschreven voor het omgaan met eventuele uitdagingen op het gebied van integratie en/of een aanpak voor geautomatiseerd testen, unittesten, Test Driven Development, etc. Dus een duidelijke beschrijving welke activiteiten rondom het testen plaats hebben in een sprint en eventueel daarbuiten.

‘Exploratory testing toepassen binnen Scrum levert een fantastische combinatie op, waarbij testen prima aansluit bij het algemene Scrumproces.’

Exploratory testing toepassen binnen Scrum levert een fantastische combinatie op, waarbij testen prima aansluit bij het algemene Scrumproces. Het exploratory testing-proces voorziet in een korte feedback loop over de kwaliteit van de software, vereist geen volgordelijkheid in ontwerp, bouw en test en levert een methode op die aansluit bij het Agile Manifesto.

Testautomatisering

Met de groeiende populariteit van iteratieve ontwikkelmethoden waarbij steeds kleine delen software worden ontwikkeld en toegevoegd aan een steeds groter wordend informatiesysteem, is er ook een toenemende behoefte aan testautomatisering. Want na elke iteratie wil je dat de toegevoegde software de goede werking van de bestaande software niet aantast. Het is hiervoor dus belangrijk om regressietesten uit te voeren op de bestaande software om dit te bewerkstelligen.

Indien het tot dusverre gerealiseerde informatiesysteem slechts een aantal iteraties aan software omvat, is een handmatige regressietest prima uit te voeren. Echter, op den duur zal er een moment bereikt worden waarbij erg dit te veel tijd in beslag gaat nemen, dat het interessant wordt de regressietests te gaan automatiseren.

Er zijn een aantal zaken belangrijk om te bepalen of, en zo ja, hoe en welke testen je wilt gaan automatiseren:

- Moment waarop je de integratie wilt of moet testen;
- Tijd die het kost om regressietesten uit te voeren;
- De mate waarin het informatiesysteem zich geautomatiseerd laat testen;
- De tijd die het opzetten en onderhouden van de geautomatiseerde regressietests kost;
- De staat van ontwikkeling van het informatiesysteem.

Moment waarop je de integratie moet of wilt testen
Afhankelijk van de opzet van je het ontwikkel- en testtraject worden regressietesten elke dag, elke build, elke iteratie en/of elke release uitgevoerd. Indien er een separate periode van integratietesten is gedefinieerd, bijvoorbeeld per halfjaarlijkse release, dan zal het handmatig uitvoeren van die testen tezamen op jaarbasis uiteraard veel minder tijd kosten dan wanneer je ze elke

dag uitvoert. Dit bepaalt voor een deel de return on investment en het break even point of het de investering van het automatiseren, de tooling en het onderhouden van (een deel van) de tests verantwoord is.

Tijd die het kost om de regressietesten uit te voeren
Indien de integratie als onderdeel van je iteratie is gedefinieerd (ongeacht of de software ook na iedere iteratie wordt gereleaset) zal de test vaker worden uitgevoerd en dus tijdswinst geboekt indien deze wordt geautomatiseerd. Maar behalve de totale tijd die een en ander kost, is ook doorlooptijd mogelijk een belangrijke factor. Iedere iteratie van twee weken (10 werkdagen) een dag lang handmatig regressietesten levert dus bijna 10% minder resultaat op omdat er tijdens de integratietest beter geen nieuwe software meer toegevoegd kan worden. Indien een geautomatiseerde regressietest op jaarbasis qua testuren wellicht duurder is, kan het toch wenselijk zijn deze te automatiseren indien deze bijvoorbeeld maar een uur doorlooptijd heeft. Winst in doorlooptijd kan dus zeker een belangrijke afweging zijn in de keuze voor een geautomatiseerde regressietest.

De mate waarin het informatiesysteem zich laat automatiseren
Dit is wellicht een evident punt, maar wel het vermelden waard. Niet elk systeem is gemakkelijk te automatiseren. Indien je als tester de wens hebt, maak dan het kwaliteitsattribuut testbaarheid ook voor geautomatiseerde testen bespreekbaar. Maar ook systemen met veel batchbewerking kunnen wellicht minder geschikt zijn. Daarnaast zijn een stabiele testomgeving en eventueel stabiele externe koppelingen ook van groot belang, dit om te voorkomen dat tests falen door andere factoren dan fouten in de software. Anders wordt er veel tijd verspild aan het keer op keer controleren van stukgelopen tests, met daarbij ook het risico dat de controle uiteindelijk verslapt.

De tijd die het onderhouden en opzetten van de geautomatiseerde regressietests kost
Naast licentie- en/of aanschafkosten van tooling om je geautomatiseerde regressietests in te onderhouden, moet niet worden onderschat hoeveel tijd het aanmaken en onderhouden van geautomatiseerde tests kan kosten. Indien integratie onderdeel uitmaakt van je de iteratie, dienen voor nieuwe delen software dus ook binnen de iteratie de regressietesten te zijn aangepast of aangevuld. Dit zal over het algemeen extra inspanning kosten bovenop de handmatig uit te voeren tests. Een softwareaanpassing in het hart van het systeem bijvoorbeeld kan ook voor de regressietests heel erg veel impact hebben. Dit zijn punten die meegenomen moeten worden in de initiële afweging om te kiezen voor geautomatiseerde regressietests en die daarna onderdeel moeten zijn van de impactbepaling bij planningssessies.

De staat van ontwikkeling van het informatiesysteem
Voor de afweging of het rendabel is om regressietests te gaan automatiseren, is ook de fase in de levenscyclus waarin het informatiesysteem zich bevindt van belang. Een systeem waarvan de ontwikkeling net is gestart, is vaak nog onderhevig aan fundamentele aanpassingen, die ook fundamentele en daarmee zeer tijdrovende aanpassingen in de regressietesten teweegbrengen. Wees dus geduldig met het kiezen van je het beginpunt voor testautomatisering. Het is veelal echter wel verstandig om in een vroeg stadium wat proeven te doen met testautomatisering om zaken als testbaarheid te onderzoeken en ervaring op te doen om zodoende later een goede opzet van de geautomatiseerde regressietesten te kunnen maken.

Keerzijde
Let wel dat je het automatiseren van regressietests niet te veel romantiseert. Het is een goed hulpmiddel, maar een belangrijk aspect van geautomatiseerde tests is dat deze precies doen wat er geautomatiseerd is. Controles op functionaliteiten A, B en C controleren ook daadwerkelijk alleen A, B en C. Functionaliteit D,

wordt dus niet gecontroleerd. Dit klinkt allemaal logisch, maar indien je deze gedachte doortrekt dan kun je concluderen dat geautomatiseerde regressietests vaak louter verwachte problemen detecteren. Deze zijn verwacht omdat er bij de opzet van de test controles voor zijn gemaakt. Indien functionaliteit D er door een andere aanpassing plots bij is gekomen, zal de geautomatiseerde regressietest dit niet constateren. Wees je dus ook vooral heel erg goed bewust wat de test niet voor je test. Een geslaagde regressietest betekent dus niet automatisch een goed werkend systeem.

‘Let wel dat je het automatiseren van regressietests niet te veel romantiseert. Het is een goed hulpmiddel, maar een belangrijk aspect van geautomatiseerde tests is dat deze precies doen wat er geautomatiseerd is.’

Bovenstaande betekent ook dat een geautomatiseerde regressie-test geen vervanger is voor handmatige tests. Functionaliteit D wordt namelijk wel opgemerkt door een tester die dat deel van het systeem onderhanden heeft bij een handmatige test. Ook het gehele intellectuele en interactieve karakter van een handmatige test, kan niet worden geautomatiseerd. Dit is een onontbeerlijk onderdeel van goede tests.

‘Ook het gehele intellectuele en interactieve karakter van een handmatige test, kan niet worden geautomatiseerd.’

Handmatig kwaliteit vaststellen en geautomatiseerd kwaliteit vasthouden is over het algemeen een goede indicatie hoe je je testaanpak kunt definiëren. Een nieuw deel software dus handmatig testen en daarbij gebruik maken van het intellect van de tester, de input vanuit de brainstorm en een goede controle op de onverwachte fouten. Indien deze tests zijn afgerond kun je geautomatiseerde regressietests voor deze software maken die de hoofdlijnen test. Hiermee mitigeer je het risico dat deze component die goed werkt, later nog ongewild en ongemerkt wordt aangetast door een andere wijziging.

‘Handmatig kwaliteit vaststellen en geautomatiseerd kwaliteit vasthouden.’

Welke testen je voor een regressietest automatiseert, is ook weer afhankelijk van tijd en risico. Automatiseer je bijvoorbeeld alleen de happy-flows of ook foutsituaties. Zoom je in op het gehele systeem en/of op kleine onderdelen van een systeem. Verdiep je ook in de testen die gedaan worden in de vorm van geautomatiseerde unittests. Dit alles tezamen bepaalt hoe je geautomatiseerde vangnet eruit komt te zien.

Cultuurverandering

Door het toepassen van exploratory testing zullen er veel zaken veranderen aan de invulling van rol van tester. Deze veranderingen zijn ook absoluut nodig om een start te maken met de hoognodige innovatie in het testvak. De tester zal in zijn rol meer en meer gaan handelen in lijn met het Testmanifest. De tester onderscheidt en ontwikkelt zich daarmee voortdurend op onderstaande punten:

- Kan snel beredeneren en acteren en is daarbij creatief;
- Denkt in mogelijkheden, en liefst zelfs in een veelvoud van mogelijkheden;
- Is contextbewust en benut dat om meer succesvol te zijn;
- Kan zich laten afleiden zonder het doel uit het oog te verliezen;
- Weet de aanwezigheid van de klant goed te benutten;
- Is positief en enthousiast en weet dit op anderen over te brengen;
- Is kritisch maar constructief;
- Streeft voortdurend naar verbetering en wil excelleren, accepteert geen middelmatigheid.

De traditionele tester
De traditionele tester stelde vooraf in alle rust testspecificaties op. Met een detailontwerp werden middels een testtechniek keurig testgevallen opgesteld. Aangevuld met een uitvoervoorspelling was er dan altijd iets om op terug te kunnen vallen. Dit gaf hen een gevoel van rust en veiligheid. Een probleem hierbij is dat het testontwerp minimaal dezelfde fouten bevat als het globaal- en detailontwerp, eventueel aangevuld met fouten die de tester zelf heeft gemaakt in de specificatie. Daarnaast is een ontwerp altijd een versimpelde weergave van de realiteit en daardoor dekken de testgevallen maar een klein deel van de realiteit.

Door de huidige complexiteit van systeemontwikkeling wijzigen specificaties en doelstellingen veelvuldig en wordt een grote mate

van flexibiliteit gevraagd. Naast eerdergenoemde problemen vormt een tester die eerst de hele testdocumentatie moet bijwerken alvorens de wijziging daadwerkelijk te kunnen testen zelf een beperkende en vertragende factor.

De nieuwe tester

Van testers wordt een andere manier van werken en een andere houding gevraagd. Voor deze testers is geen uitdaging groot genoeg en de tester weet van aanpakken. Ze laten de klant graag participeren in hun tests, testen buiten de gebaande paden en hebben veel parate testkennis die ze in hoog tempo en onder druk kunnen toepassen. Van wijzigingen raken ze niet in paniek, maar verwerken deze direct en moeiteloos in hun test. Ze maken een ontwerp tot input voor hun test, maar beperken zich niet alleen tot het testen van die hierin beschreven functionaliteit, maar gaan verder. Ze kunnen op elk moment bepalen wat ze willen gaan testen, zonder hiervoor terug te hoeven vallen op een eerder opgesteld document.

‘Voor het succesvol uitvoeren van exploratory testing (en eigenlijk voor testen in het algemeen) is een andere mindset bij de testers nodig.’

Voor het succesvol uitvoeren van exploratory testing (en eigenlijk voor testen in het algemeen) is een andere mindset bij de testers nodig. Deze mindset laat zich typeren door verantwoordelijkheid, flexibiliteit en context denken. Daarnaast spreekt voor zich dat de tester de theorie van het testvak beheerst en continu aan alle testvaardigheden wil blijven werken. Ook communicatief en analytisch is deze tester uiteraard zeer vaardig. De tester die deze aspecten beheerst, is ‘de nieuwe tester’.

Verantwoordelijkheid
De nieuwe tester wil verantwoordelijk zijn voor zijn eigen resultaten. Verder wil hij geïnspireerd worden om uitzonderlijke resultaten te behalen en om zijn werk nog beter te kunnen uitvoeren.

Flexibiliteit
De nieuwe tester weet dat de wereld continu verandert en vindt veranderingen vanzelfsprekend. De tester wordt niet zenuwachtig bij het ontbreken van specificaties in zijn testbasis en start de testsessies zonder een verwacht resultaat voorhanden te hebben. De nieuwe tester is in staat ter plekke en onder meer tijdsdruk zowel te specificeren, testen uit te voeren en resultaten te analyseren.

Context denken
De nieuwe tester inventariseert en begrijpt de context waarin hij acteert en is zich doorlopend bewust van het uiteindelijk doel van het te ontwikkelen product. De nieuwe tester staat voor het belang van de eindgebruiker en doelen van en risico's voor de business.

Een groot deel van de testers die zich met bepaalde persoonlijke eigenschappen of juist het ontbreken hiervan konden functioneren in een traditioneel testproces, kunnen veel moeite gaan hebben om te functioneren als nieuwe tester. Testers die goed gedijen in een omgeving met structuur, regelmaat en specificaties, zullen zich waarschijnlijk ook niet prettig voelen in een meer dynamische omgeving. In feite verandert het profiel van een tester in grote mate. Dit heeft onlosmakelijk tot gevolg dat een aanzienlijk deel van de huidige populatie testers niet geschikt zal zijn als nieuwe tester.

Nawoord

In dit boek is exploratory testing als testmethode beschreven die helpt de focus te leggen op de essentie van testen. Het helpt in het creëren en onderhouden van een eenvoudig, maar uiterst efficiënt testproces. Een testproces dat alleen bestaat uit activiteiten op het gebied testuitvoering of activiteiten die daar een directe bijdrage aan leveren. Testuitvoering is immers de enige echte kwaliteit verhogende activiteit. De methode levert daarmee een testproces op dat de beste kwaliteit levert binnen het spanningsveld van tijd en geld.

Naast een geoptimaliseerd testproces vormt exploratory testing ook een geweldige match met Agile ontwikkelmethoden als Scrum en is het een methode die aansluit bij het Agile Manifesto.

Met dit boek willen wij het vak testen uit de vicieuze cirkel halen waar het in is beland. We keren terug naar de essentie van testen en hebben op basis hiervan een set ondersteunende activiteiten ingericht om op die manier tot een testmethode te komen, die past binnen iteratief en agile werken, maar ook zeer goed past bij lineair ontwikkelen. Puur testen, zonder rompslomp.

Veel succes met uw eigen contextafhankelijke implementatie!

Veel gestelde vragen

De tekst in dit boek is niet uitputtend. Voor veel gestelde vragen hebben wij onderstaand een overzicht van vragen en antwoorden opgenomen.

Hebben exploratory-testers training nodig?
Zeker! Naast het feit dat exploratory-testers continu al nieuwe kennis en ervaring opdoen, moeten ze in beginsel goed zijn opgeleid. Ze moeten op de hoogte zijn van de meest gangbare testprincipes en testtechnieken aangezien ze die real-time moeten kunnen toepassen in de testsessies. Iedere verdere training draagt bij aan de kennis en kunde van de tester om betere tests uit te kunnen voeren.

Kan exploratory testing worden uitgevoerd met minder ervaren testers?
Ja, maar het stelt dan wel hogere eisen aan de exploratory coach. Hij moet voldoende senior zijn om de testers goed aan te kunnen sturen en goed te coachen. Junior testers maken exploratory testing minder efficiënt, maar dat geldt in dezelfde mate voor de traditionele testmethoden. Ervaring is weliswaar belangrijk, maar de houding en mindset van de junior tester is bepalend. Met een goede coach kunnen junior testers nergens zo snel ervaring opdoen als binnen een exploratory testing-traject.

Is exploratory testing in alle situaties goed toepasbaar?
Ja, in verreweg de meeste situaties kan exploratory testing goed als methode worden ingezet om snel de meest belangrijke fouten te vinden. Bij hedendaagse systeemontwikkelmethoden biedt het zelfs erg grote voordelen ten opzichte van traditionele testmethoden omdat je door exploratory testing als methode toe te passen goed met onzekerheden en wijzigingen overweg kan. Ook als er weinig tijd is om te testen, er niet of nauwelijks specificaties voorhanden zijn of snel terugkoppeling nodig is op een nieuw product, nieuwe functie of aanpassing is exploratory testing goed inzetbaar.

Zijn er situaties waarin exploratory testing minder goed toepasbaar is?
Ja, er zijn situaties denkbaar waarbij exploratory testing beter niet toegepast kan worden. Bijvoorbeeld bij het testen van functies waarvoor batch processen noodzakelijk zijn en de resultaten van zo'n batch op zich kunnen laten wachten. Op dat moment kan een sessie niet voldoen aan het snelle en flexibele karakter. Ook het testen van systemen waarbij complexe berekeningen worden uitgevoerd, zijn verwachte uitkomsten haast noodzakelijk. Verder is het raadzaam voor de meest belangrijke onderdelen ook wat uitgebreidere logische testspecificaties op te stellen voor de uitvoering met exploratory testing.

Kan exploratory testing gebruikt worden voor het testen van complexe systemen?
Ja, juist bij complexe systemen. Daar bestaat het gevaar dat met het opstellen van testgevallen de samenhang en context van sommige functies verloren gaat. Exploratory testing test per definitie vanuit de context met idealiter gebruikers als sparringpartner waarbij dit verlies tot een minimum beperkt wordt. Daarnaast kennen complexere systemen vaak meer functionele wijzigingen in het ontwikkeltraject, wat een flexibele methode als exploratory testing nodig maakt.

Hoe gaat exploratory testing om met hertesten?
Voor het uitvoeren van hertesten is tijd ingeruimd. Voor met de hertest aan te vangen neemt de tester kennis van de eerder uitgevoerde testcharter. Voor de hertest zelf wordt gestart met een nieuwe testcharter. Het uitvoeren van een hertest op een andere wijze stimuleert dat de oplossing van de bevinding wordt gehertest in relatie met de overige programmatuur. Hiermee is exploratory testing sterk in het ontdekken van de door de aanpassing nieuw geïntroduceerde bevindingen.

Hoe gaat exploratory testing om met regressietesten?
Eigenlijk hetzelfde als met hertesten. Door een goede decompositie is inzichtelijk te maken welke delen van de software een verband houden met de aangepaste onderdelen. Afhankelijk daarvan zullen een of meerdere paren met nieuwe testcharters delen van de software opnieuw testen.

Wordt exploratory testing nu al veel gebruikt als testtechniek?
Het leren, specificeren en uitvoeren wordt haast onontkoombaar toegepast bij elke testuitvoering. Bijna elke tester past wel een aantal testgevallen aan of bedenkt een aantal nieuwe testgevallen naar aanleiding van wat hij of zij tijdens of voor de testuitvoering geleerd heeft over het te testen onderdeel. Ook wordt exploratory testing nog weleens toegepast in geval van tijdnood. Feitelijk wordt het dus hier en daar toegepast als ondersteunende techniek. Maar in die toepassing ook vaak ten onrechte verward met error guessing.
Het gebruiken van exploratory testing als core testtechniek met een efficiënt ingericht ondersteuningsproces en testsessies zie je tot dusverre weinig.
Er zijn toch veel bedrijven en projecten die vooralsnog de TMap leer volgen omdat er in Nederland ook vrij weinig alternatieven voor handen lijken. Daarnaast biedt het uitgebreid specificeren een eerder beschreven schijnzekerheid over de kwaliteit van je softwareoplossing die door veel bedrijven toch als prettig wordt ervaren.
Wel zijn er steeds meer bedrijven die worstelen met het vormgeven van testen in met name Agile ontwikkeltrajecten en die vanuit die achtergrond zie je wel steeds meer initiatieven gebaseerd op exploratory testing ontstaan.

Vraagt exploratory testing meer tijd dan traditioneel testen?
Nee, zeker niet. Exploratory testing kan de beschikbare tijd anders indelen waardoor de beschikbare tijd voor de daadwerkelijke testuitvoering omhoog gaat. Eigenlijk biedt exploratory testing zelfs

een mogelijkheid aanzienlijk op het budget te kunnen bezuinigen zonder de kwaliteit van de software nadelig te beïnvloeden.

Is exploratory testing wel transparant?
Exploratory testing is veel transparanter dan traditionele testmethoden. Er is op ieder gewenst moment een rapport te leveren met de exacte status van het testproject, inclusief de tot dan toe ontdekte bevindingen met hun alternatieven, de besteding tot dan toe, de niet geteste onderdelen met hun prioriteit, etc. Daarnaast is er op het gebied van planning veel meer transparantie. In de traditionele methoden werd er voor de test van het gehele systeem een planning opgesteld. Dit planningsgebied is vaak zo groot dat het niet overzien kan worden en er vaak te optimistisch gepland wordt en de planning daarmee ver van de uiteindelijke waarheid verwijderd is. Door met exploratory testing het gehele systeem in behapbare brokken te verdelen die zich veel beter laten plannen heb je met de som van alle componenten een veel betrouwbaarder planning.

Is de dekkingsgraad bij exploratory testing wel voldoende?
Exploratory testing heeft een betere dekkingsgraad dan traditionele testmethoden. Met name omdat ook de minder of niet beschreven delen van het informatiesysteem worden getest op basis van contextkennis, businesskennis en de creativiteit van de tester. Bij traditionele methoden leverden de technieken vooraf testspecificaties van waaruit een dekkingsgraad zou kunnen worden afgeleid. Maar zoals in dit boek al meerdere malen beschreven, er is veel meer te testen dan wat beschreven is in specificaties. Dus daarmee is een dekkingsgraad bepalen op basis van deze specificaties ook pure nonsens en een schijnzekerheid.

Hoe implementeer ik exploratory testing in mijn organisatie?
Door gewoon te starten! Exploratory testing kan op ieder gewenst moment geïntroduceerd en gestart worden. Stel een testaanpak op met een duidelijke doelstelling (context), een functionele decompositie met een risicoanalyse en prioritering. Koppel hier de

planning, resourceplanning en begroting aan en bespreek dit met de opdrachtgever. Er is geen documentatie of testware uit het verleden voor nodig. Wel is initiële procesbegeleiding voor de testers aan te bevelen.

Hoe houd ik het overzicht in een groot project?
Door het inrichten van de charter stash en een continu proces van debriefen en herijken heb je ten alle tijden een totaaloverzicht van alle te testen en geteste functionaliteit met een overzicht aan mogelijke gebreken.

Hoe gaat exploratory testing om met testprocesverbetering (waaronder TPI)?
Met exploratory testing voer je constant de beste test uit in de daarvoor beschikbare tijd. Met iedere testsessie leert de tester meer van testen, het systeem en de werking hiervan. In de hierop volgende debriefing wordt de testsessie geëvalueerd en waar nodig wordt de tester verder gecoacht en uitgedaagd om te verbeteren. De principes achter Lean gaan ook op voor exploratory testing. Testprocesverbetering is dan ook niet het steeds verder uitbreiden en verder dichttimmeren van de verschillende testen en het testproces, maar eerder procesoptimalisatie waarbij het denken en de mindset van de tester voorop staat om tot de best mogelijke tests te komen in de beschikbare tijd en geld.

Over de auteurs

Eugene Derksen
Eugene Derksen heeft zich als testmanager sinds 1996 in een groot aantal projecten binnen verschillende sectoren ontwikkeld op het gebied van ketens en de regie hierop. De afgelopen 12 jaar vervulde hij meermalen rollen als testmanager, ketentestmanager en ketenregisseur – allen binnen de strafrechtketen, zowel bij het Ministerie van Veiligheid en Justitie als ook haar taakorganisaties. Daarnaast is hij ondernemer en oprichter van o.a. DutchPelican, een bedrijf gespecialiseerd in exploratory testing en ketentesten.

Eugene staat bekend om zijn vermogen zich buiten de gebaande paden te begeven om tot oplossingen en verbeteringen te komen. Sinds zijn kennismaking met de context driven school in 2001, heeft hij meermaals exploratory testing als afzonderlijke testmethode binnen organisaties geïmplementeerd.

Patrice Hulzebos
Patrice Hulzebos is sinds 2005 werkzaam als software tester, scrummaster, testcoördinator en product owner binnen verschillende sectoren. Sinds 2016 is hij testspecialist in dienst van DutchPelican.

In de jaren dat Patrice als tester actief is, liep hij constant tegen een gevoel van inefficiënt werken, eindeloos documenteren, nooit gehaalde planningen en klantverafschuwing aan. Dit heeft hem de aanzet gegeven zijn eigen visie en werkwijze te veranderen gebaseerd op de principes van Agile ontwikkelen en exploratory testing. Na zijn ideeën en werkwijzen geventileerd en toegepast te hebben binnen zijn eigen werkomgeving, wil Patrice met dit boek een groter publiek aanspreken, omdat hij gelooft dat zijn ideeën en mening een goede bijdrage leveren aan de ontwikkeling van het testvak.

Dankwoord

Dit boek is niet ineens tot stand gekomen, maar is in een periode van 5 jaar geschreven en ook herschreven. Wij willen de volgende mensen bedanken voor hun bijdrage en geduld: Lilian Nijboer, Huib Schoots, Greet Zwaan, René Bouma, Jan Derksen en uiteraard Carlo van Driel die het voorwoord voor zijn rekening wilde nemen en Celeste Hurenkamp en Inge van Vught voor de vormgeving van de illustraties.

Verklarende woordenlijst

Acceptatietest – Een formele test met betrekking tot gebruikersbehoeften, eisen (requirements) en bedrijfsprocessen, die worden uitgevoerd om vast te stellen of een systeem al dan niet aan de acceptatiecriteria voldoet, en om de gebruiker, klant of een andere geautoriseerde entiteit informatie te geven voor het besluit om het systeem wel of niet te accepteren.

Agile – Een groep ontwikkelmethoden gebaseerd op iteratief ontwikkelen door zelfsturende teams.

Agile Manifesto – Het Agile Manifesto (Manifesto for Agile Software Development) werd opgesteld tijdens een informele bijeenkomst van zeventien softwareontwikkelaars. Deze bijeenkomst vond plaats van 11 tot en met 13 februari 2001 in 'The Lodge' in Snowbird (Utah). Het handvest en de principes vormden een uitwerking van ideeën die halfweg de jaren negentig waren ontstaan, als reactie op methoden die men traditioneel klasseert als waterval-ontwikkelmodellen. Die modellen werden ervaren als bureaucratisch, traag en bekrompen en zouden de creativiteit en effectiviteit van ontwikkelaars belemmeren. De zeventien mensen die het Agile Manifesto samen hebben opgesteld vertegenwoordigden de diverse Agile stromingen. Na de publicatie van het handvest, richtten enkele ondertekenaars de 'Agile Alliance' op, om de principes verder om te zetten in methoden.

Brainstorm – De fase voorafgaande aan de testsessie waarbij er ideeën worden opgedaan over wat er die dag getest moet worden en vooral ook hoe dat te doen.

Burn-down chart – Grafische weergave van de testvoortgang binnen exploratory testing. De grafiek toont het aantal testcharters ten opzichte van de tijd.

Charter – Zie: Testcharter.

Charter Stash – De voorraad testcharters binnen een testproject.

Context denken – Alle functionaliteiten en constateringen hierop relateren aan de specifieke situatie waarin deze gebruikt danwel gedaan worden.

Context Driven Testing – Testers die Context-Driven Testing toepassen kiezen zelf hun testdoelen, technieken en op te leveren producten (inclusief testdocumentatie) op basis van de context en details van die specifieke situatie, inclusief de wensen van de opdrachtgever en eventueel andere belanghebbenden. De essentie van Context-Driven Testing is projectafhankelijk toepassen van vaardigheden.

Curve van Boehm – Een grafiek die de relatie aangeeft tussen de hoogte van de herstelkosten van een probleem ten opzichte van het moment waarop het geconstateerd wordt.

Debriefing – Evaluatiegesprek aan het eind van een testsessie waarin de testsessie, inclusief de gedane bevindingen, de alternatieven en de onderkende risico's geëvalueerd worden, waarna decharge van het tweemans testteam plaatsvindt.

Dekkingsgraad – De mate waarin een bepaald dekkingsonderdeel geraakt wordt door een testset, uitgedrukt als percentage van het geheel.

Detail Testplan – Een testplan dat zich specifiek richt op één testfase.

Dynamisch testen – Testen waarbij de software van een component of systeem wordt uitgevoerd.

Exploratory Testing – Een testaanpak die anticipeert op de vrijheid en verantwoordelijkheid van iedere tester om doorlopend de waarde van zijn werk te optimaliseren. Dit wordt gedaan door

zowel leren, specificeren en uitvoeren van testen te behandelen als onderling ondersteunende activiteiten die parallel lopen gedurende een project.

Foutclustering – (of defect clustering) Fenomeen waarbij een beperkte set componenten en/of applicatiedelen het merendeel aan defects bevat en de meeste problemen in productie veroorzaakt.

Gebruikers Acceptatietesten – Zie: Acceptatietesten.

Hertesten – Opnieuw testen van situaties die de laatste keer niet geslaagd waren om de juistheid van herstelacties te verifiëren.

Heuristics lijst – Uniek referentiekader met ervaringen van een tester die in het verleden zijn opgedaan en zijn toegespitst op testtips, trucs en veelvoorkomende fouten die hij in toekomstige tests kan benutten.

Informatiesysteem – Het geheel van hardware, software, gegevens, mensen en procedures en andere zaken die een rol spelen bij de informatievoorziening.

Integratietest – Test om fouten in interfaces en interacties tussen geïntegreerde componenten of systemen bloot te leggen.

ISTQB – International Software Testing Qualifications Board, een organisatie die internationale richtlijnen opstelt voor certificering van testers en bijdraagt aan een eenduidige voorziening van vakinformatie voor testers.

Lean – Methode voor procesoptimalisatie.

Lineair Systeemontwikkelingsproces – Een methode voor softwareontwikkeling, waarin de ontwikkeling regelmatig vloeiend naar beneden loopt (als een waterval). De ontwikkeling doorloopt

een aantal fasen, namelijk: definitiestudie of analyse, basisontwerp, technisch ontwerp of detailontwerp, bouw, testen, integratie en beheer en onderhoud.

Non-functional Testing – Het testen van een component of systeem ten aanzien van niet functionele kwaliteitsattributen, zoals betrouwbaarheid, efficiëntie, bruikbaarheid, onderhoudbaarheid en portabiliteit.

Master Test Plan – Een testplan dat betrekking heeft op meerdere testsoorten.

Pair testing – Testvorm waarbij twee personen samenwerken in het opsporen van fouten. Meestal delen ze één computer die ze afwisselend bedienen terwijl ze testen.

Pesticide paradox – Fenomeen dat wanneer je dezelfde testpaden blijft doorlopen je op den duur geen nieuwe bevindingen meer doet terwijl die er wel zijn.

Rapid Application Development (RAD) – Een methode voor softwareontwikkeling in projectvorm. RAD is een concept dat organisaties in staat stelt softwaresystemen sneller en van betere kwaliteit te (laten) ontwikkelen.
De belangrijkste elementen van RAD zijn:

- Het verzamelen van eisen aan de hand van workshops of focusgroepen;
- Prototyping en vroegtijdige herhaalde gebruikertesten van modellen;
- Het hergebruik van softwarecomponenten;
- Snelle ontwikkeling door uitstel van ontwerpverbeteringen tot de volgende versie;
- Minder formaliteit in verslagen en andere team communicatie.

RAD maakt vaak gebruik van object georiënteerd programmeren. De meest populaire objectgeoriënteerde programmeertalen zijn

C++, Java en C#. Het concept kan ook worden toegepast op de ontwikkeling van hardware.

Regressie testen – Het testen van een eerder getest programma na een wijziging, om aan te tonen dat er geen fouten zijn geïntroduceerd of ontdekt in ongewijzigde gebieden van de software als gevolg van die wijzigingen. Het vindt plaats wanneer de software of de omgeving is gewijzigd.

Requirement – Een voorwaarde of mogelijkheid van een gebruiker om een probleem op te lossen of een doel te bereiken waaraan een systeem of subsysteem moet voldoen, waarbij aan een contract, standaard, specificatie of een ander formeel opgelegd document wordt voldaan.

Scrum – Een Agile softwareontwikkelingsmethode met een vaste set van regels om met multidisciplinaire teams in korte sprints (iteraties) werkende software op te leveren.

Session based test management – Een methode om session based testing (bijv. exploratory testing) te managen en te beheersen.

Session Based testing – Een testaanpak waarbij testactiviteiten worden gepland als ononderbroken sessies van testontwerp en uitvoering, vaak toegepast in combinatie met exploratory testing.

SmarTEST – Nederlandse methode voor het testen van software. De SmarTEST-aanpak wordt gekenmerkt door de vijf principes van slim testen: Strategisch, Mensgericht, Adaptief, Risicogedreven en Transparant.

Statisch testen – Testen van een component of systeem op specificatie- of implementatieniveau, zonder die software uit te voeren, bijvoorbeeld door een review of statische code analyse.

Systeemontwikkeling – Ontwikkeling van systemen (software) waarbij gebruik gemaakt wordt van systeemontwikkelmethoden.

Systeemontwikkelmethode – Een methode die gebruikt wordt bij het ontwikkelen van systemen, veelal software. In geval van software wordt ook wel softwarelevenscyclus of softwareproces genoemd. Zie ook: Lineair Systeemontwikkelingsproces, Rapid Application Development (RAD) en Scrum.

Systeemtest – Het proces van het testen van een geïntegreerd systeem om te verifiëren dat het aan de gespecificeerde eisen voldoet.

Testaanpak – De vervanger van het testplan voor exploratory testing, waarin bijvoorbeeld een functionele decompositie, prioritering en een testplanning beschreven kunnen staan, maar ook het testproces, bevindingenprocedure, opslag van testware enzovoorts.

Testcharter – Werkdocument voor een individuele testsessie binnen exploratory testing op een deel van de te testen software.

TestFrame® – Testmethode voor het gestructureerd testen van software. Bij TestFrame® is de vastlegging van de testgevallen dusdanig dat geautomatiseerde testuitvoering hierbij goed aansluit.

Testidee – Een idee over hoe een test uitgevoerd zou kunnen worden gebaseerd op de kennis, kunde en ervaring van de tester.

Testmanifest – Het hedendaagse testen met de huidige testmethoden en testers waarbij krampachtig vastgehouden wordt aan achterhaalde uitgangspunten, geeft onvoldoende antwoord op de vraag naar kwaliteit en toegevoegde waarde in projecten, met name binnen Agile ontwikkelteams. Dit maakte dat twee testers in 2016 het Testmanifest hebben opgesteld met hierin een set actuele

uitgangspunten die beschrijven hoe testen en de tester weer tot meerwaarde gemaakt kunnen worden.

Testobject – Een component of systeem dat getest moet worden.

Testplan – Een document dat de afbakening, de aanpak, de hulpmiddelen en de planning van de testactiviteiten beschrijft. Het beschrijft o.a. de testelementen, de te testen aspecten, de test-taken, wie welke taak uit zal voeren, niveau van onafhankelijkheid van de tester, de testomgeving, de testspecificatietechnieken en de ingangs- en uitgangscriteria en de beweegredenen voor die keuze, en de risico's die noodscenario's behoeven. Het is het eindresultaat van het testplanningproces.

Testscript – Deze term wordt gewoonlijk gebruikt om te refereren aan een testprocedure specificatie, in deze term meestal van een geautomatiseerde testprocedure.

Testsessie – Periode van 4 tot 8 uur waarin een van tevoren bepaald deel van de software wordt getest.

TMap (Next)® – Testmethode voor het gestructureerd testen van software.

Unit test – Het testen van afzonderlijke softwarecomponenten (component testing).

V-model – Een kader om de activiteiten van de softwareontwikkel-levenscyclus van specificatie van eisen tot en met het onderhoud te beschrijven. Het V-model illustreert hoe de testactiviteiten in elke fase van de softwareontwikkellevenscyclus kunnen worden geïntegreerd.

Valideren – Bewijs door onderzoek en door aanleveren van objectief bewijsmateriaal dat aan de eisen ten aanzien van een specifiek voorgenomen gebruik of van een toepassing is voldaan.

Verifiëren – Bewijs door onderzoek en door aanleveren van objectief bewijsmateriaal dat aan de gespecificeerde eisen is voldaan.

Watervalmethode – Zie: Lineair systeemontwikkelingsproces.

Literatuur

Software Engineering Economics – Barry Boehm, 1981

The Complete Guide to Software Testing – Bill Hetzel, 1984

Testing Computer Software – Cem Kaner, 1987

Session Based Test Management – Jonathan Bach, 2000

Lessons Learned in Software Testing – Cem Kaner, James Bach en Bret Pettichord, 2001

The Testing Practitioner – Erik van Veenendaal, 2002

Exploratory Testing Explained – James Bach, version 1.3, 2003

TestFrame – Chris Schotanus, 2009

TMap Next ® – Tim Koomen, Leo van der Aalst, Bart Broekman en Michiel Vroon, 2009

Scrum en Exploratory Testing – Eugene Derksen, 2010

Exploratory Testing Explained – James Bach, version 3.0, 2010

Standaard verklarende woordenlijst van software testtermen (Vertaling Engels – Nederlands) – Working Party-Glossary van de BNTQB, versie 2.0, 2008

Voor verdere achtergrondinformatie zijn de volgende websites geraadpleegd:

www.satisfice.com

www.context-driven-testing.com

www.en.wikipedia.org/wiki/Exploratory_testing

www.nl.wikipedia.org/wiki/Agile-softwareontwikkeling

www.agilemanifesto.org

www.poppendieck.com