United Certifications - Testing Fundamentals voor software engineers (ontwikkelaars en beheerders)

Syllabus

Versie 1.1 30-10-2023

Copyright

Woord van dank

Ik wil een aantal mensen in het bijzonder bedanken bij het tot stand komen van dit werk. Voor mij was het al lang een wens om mijn ervaring te kunnen bundelen of vastleggen in een soort van verhaal. Dit was me niet gelukt zonder hulp van de volgende mensen. Dank aan allen voor het op de één of andere manier mede mogelijk maken van het tot stand komen van dit verhaal: Kyle Siemens, Rogier Ammerlaan, Daniel van der Zwan, Annelies van Rijn, Valentijn Duijser, John Wittmaekers, Anne Laura Drost-Douma, David de Roo, Sjoerd Walinga, Jose Correia, Khadidja Zegrir, Linda van Straaten en natuurlijk alle andere collega's van Concept7 voor de hulp bij het maken van deze syllabus.

Versie

Versie	Datum	Opmerkingen
1.0	04-08-2023	Eerste druk
1.1	30-10-2023	Reviewopmerkingen verwerkt

Inhoudsopgave

Inleiding

Doel van dit document

Deze syllabus vormt de basis van de Testing Fundamentals voor software engineers certificering. In dit document wordt beschreven wat je nodig hebt om te kunnen slagen voor je examen. Er rust copyright op dit document en het trainingsprogramma, inclusief het examen. Het examen bevat alleen vragen over concepten en kennis die in dit document uitgelegd worden.

De verschillende onderdelen van de training zijn ook beschikbaar op de officiële website van Certified Testing Fundamentals voor software engineers. Deze onderdelen bestaan uit:

- Een complete lijst met training aanbieders en beschikbare cursusdata. Een cursus is aanbevolen, maar niet verplicht om het examen te kunnen doen.
- De syllabus (dit document) om te downloaden.
- Een compleet oefenexamen van 40 vragen en een document met antwoorden om te gebruiken als voorbereiding op het echte examen.

We streven ernaar om de documenten beschikbaar te hebben in meerdere talen. Blijf de website volgen voor verdere ontwikkelingen.

Doel van deze syllabus

Het doel van deze cursus is gericht op het leren van enkele specifieke test skills aan ontwikkelaars en beheerders. De training is geschikt voor zowel junior, medior en senior ervaren mensen.

De Certified Testing Fundamentals voor software engineers bevat alleen de basisvaardigheden. Het schetst een praktische en pragmatische aanpak, en reikt daarbij handvatten aan, die in de meeste alledaagse projecten direct toegepast kunnen worden.

Testen is geen exacte wetenschap. Er zijn talloze manieren om het doel, dat is in het algemeen een applicatie is die vrij is van ernstige en kostbare fouten, te bereiken. Dit is geen cursus die alle facetten van testen beschrijft. Het leert je ook niet alle onderdelen van testen die er bestaan.

Er zijn voldoende andere trainingen, die testen uitgebreid en in detail beschrijven. Die trainingen voldoen beter als je een test engineer wilt worden, of veel wilt leren over testen.

Resultaat na het volgen van deze training (bedrijfsdoelen)

BD 1	Leer als ontwikkelaar of beheerder praktische zaken over testen en kwaliteit.
BD 2	Begrijp als ontwikkelaar of beheerder de basis van testen en kwaliteit.
BD 3	Begrijp wat de randvoorwaarden en uitgangspunten zijn voor het uitvoeren van een goede test.
BD 4	Leer als ontwikkelaar of beheerder een aantal verschillende handvatten om de

	kwaliteit te verbeteren.
BD 5	Leer als ontwikkelaar of beheerder een aantal verschillende manieren om een testscript te maken.

Leerdoelen

Leerdoelen (LD) zijn korte omschrijvingen van wat je dient te onthouden na het lezen van de betreffende tekst. **[I]** Er zijn 3 niveaus:

- K1: Onthouden
- K2: Begrijpen
- K3: Toepassen

De volgende tabel geeft een overzicht van alle leerdoelen van deze training.

LD1	Onthoud wat er bedoeld wordt met testen. (K1)
LD2	Onthoud wat er bedoeld wordt met kwaliteit. (K1)
LD3	Onthoud dat je op verschillende manieren naar kwaliteit kunt kijken met behulp van de kwaliteitsattributen. (K1)
LD4	Begrijp de noodzaak van testen. (K2)
LD5	Begrijp het verschil tussen de focus van ontwikkelaars en de focus van testers. (K2)
LD6	Pas een aantal belangrijke testpractices toe. (K3)
LD7	Onthoud een aantal specifieke testtermen. (K1)
LD8	Begrijp de testpractice van geen aannames doen. (K2)
LD9	Begrijp de practice dat testen logisch nadenken is. (K2)
LD10	Begrijp het belang van regressietesten. (K2)
LD11	Onthoud de voor- en nadelen van testautomatisering. (K1)
LD12	Begrijp de practice dat alles testen onmogelijk is. (K2)
LD13	Begrijp het belang van een teststrategie. (K2)
LD14	Pas risicoanalyse toe in je testtraject. (K3)
LD15	Begrijp wat Pareto's analyse inhoudt. (K2)
LD16	Begrijp hoe root cause analyse je kan helpen je software ontwikkelproces te verbeteren. (K2)
LD17	Pas de practice van specifiek zijn toe. (K3)
LD18	Begrijp de practice van zo vroeg mogelijk testen. (K2)
LD19	Soorten omgevingen waarop je kunt testen. (K2)
LD20	Begrijp het belang van een goede testomgeving. (K2)
LD21	Begrijp het belang van test data. (K2)
LD22	Begrijp de practice van klein beginnen en je testen steeds uitbreiden. (K2)
LD23	Onthoud testlevels and testtypes. (K1)
LD24	Begrijp hoe testlevels en testtypes zich verhouden tot een teststrategie. (K2)
LD25	Onthoud de basis van een testproces. (K1)
LD26	Begrijp het nut van vastleggen van je testen. (K2)

LD27	Leer een testscript maken door het uittekenen van het proces/programma. (K3)
LD28	Leer een testscript maken met behulp van de semantische test. (K3)
LD29	Leer functionaliteit testen met behulp van een beslistabel. (K3)
LD30	Leer je testen verdiepen door het gebruik van grenswaardenanalyse. (K3)
LD31	Leer je testen verdiepen door gebruik van equivalentieklassen. (K3)
LD32	Leer hoe je kunt testen met behulp van een checklist. (K3)
LD33	Leer de testtechniek pairwise testing. (K3)
LD34	Begrijp het belang van communicatie in je activiteiten als tester. (K3)
LD35	Leer de basis van testen op security. (K2)
LD36	Leer de basis van testen op usability. (K2)
LD37	Leer de basis van testen op performance. (K2)

Voorkennis

Er is geen specifieke voorkennis vereist. Het is handig als je wat ervaring hebt met één of meerdere van de volgende gebieden: softwareontwikkeling, projectmatig werken, beheren van software, accepteren van software, testen van software. Maar dit is geen harde vereiste om deze training te kunnen doen en het examen met goed gevolg af te kunnen leggen.

Algemene opmerking

Om de tekst goed te laten lopen worden soms synoniemen gebruikt. Wees je er bewust van dat de volgende begrippen als volgt uitgelegd kunnen worden.

- '**Software**,' waar 'Product, Service en/of System' wordt bedoeld.
- '**Software engineers**,' waar ontwikkelaar of beheerder wordt bedoeld.
- '**SDLC**,' is de afkorting voor Software Development Life Cycle (software ontwikkel proces).

Standaarden voor testtechnieken

In deze syllabus worden diverse testtechnieken uitgelegd. Er is bewust voor gekozen om dit beperkt en eenvoudig te houden. De basis van enkele technieken wordt aangeleerd. Dit omdat deze syllabus alleen de fundamentals van testen beschrijft. Voor aanvullende testtechnieken en meer gevorderd gebruik verwijzen wij naar de ISO standaard **[XX]**, en/of andere bronnen.

Toepasbaarheid in Agile en DevOps

In deze syllabus wordt in mindere mate gerefereerd aan diverse systeemontwikkel methodieken (Waterval, Agile, DevOps). Dit is een bewuste keuze. We willen hier graag de essentie van testen neerzetten en die essentie verschilt naar onze mening niet met een andere systeemontwikkelingsmethodiek. Er zullen verschillen zijn in de manier waarop je in een project de activiteiten die beschreven zijn in de syllabus uitvoert, in de tijd, of op de omgeving waar je ze uitvoert. Ook zul je merken dat in andere types van organisaties of met andere soorten software de accenten anders komen te liggen. Soms omdat de organisaties al ge-evolueerd zijn naar een hoger volwassenheidsniveau. Maar de basis van software testen en het belang van deze basisvaardigheden zullen niet veranderen door de gekozen

systeemontwikkelingsmethodiek. Omdat deze syllabus zich richt op ontwikkelaars, beheerders en anderen die beginnen met testen is ervoor gekozen de essentie weer te geven en niet te uitgebreid in te gaan op zaken die verder van de basis staan.

Woord van de hoofdauteur Mattijs Kemmink:

Ongeveer 20 jaar voordat ik deze syllabus schreef, ben ik begonnen als software tester. Eigenlijk wilde ik ontwikkelaar worden, maar de werkgever waar ik in dienst trad had een beleid om software developers altijd eerst te laten starten als tester. Ik moest destijds vragen wat testen was, want ik had niet van testen als een aparte functie gehoord. Maar ik was blij met een vaste aanstelling bij een mooie werkgever. Zo zette ik mijn eerste stappen op het pad van software testing. In de jaren die volgden had ik meerdere keren de mogelijkheid om toch ontwikkelaar te worden, maar ik kwam toch telkens terug bij mijn roots in softwaretesten.

In deze syllabus hoop ik je mee te nemen op een reis door enkele van mijn ervaringen in de wereld van softwaretesten. Daarbij leg ik een relatie tussen mijn ervaringen en de theorie uit welbekende testmethodologiën. Ik heb geprobeerd deze concepten zo praktisch mogelijk en zonder onnodige bagage te presenteren. Maar realiseer je wel dat testen nooit simpel is, het is altijd maatwerk en is altijd afhankelijk van situatie en context. Merk ook op dat deze training geen vervanging is voor de theoretische cursussen (zoals bijv. ISTQB). Daar is het vakgebied van testen en QA te groot voor. Mijn doel in deze training is om er voor te zorgen dat deelnemers beter de rol van een tester en het belang van het vakgebied begrijpen. Ik probeer mensen te helpen ervoor te zorgen dat de overall kwaliteit verbetert van de software producten, die ze maken of beheren. En ik probeer mensen de kennis aan te reiken die ik gehad had willen hebben bij de start van mijn carriére. Ik hoop dat ik jullie enkele praktische en direct toepasbare zaken leer die je kunt toepassen in je dagelijkse praktijk. In mijn carriére ben ik verscheidene ontwikkelaars en beheerders tegengekomen, die zich betrokken voelden bij en bezorgd waren over kwaliteit en die meer wilden leren over hoe de kwaliteit te verbeteren. Deze personen wisten niet precies waar ze moesten beginnen. Dat zijn de mensen die ik graag wil helpen met deze syllabus, training en certificering.

Hoofdstuk 1: Kennis maken met Testen

LD1	Onthoud wat er bedoeld wordt met testen. (K1)
LD2	Onthoud wat er bedoeld wordt met kwaliteit. (K1)
LD3	Onthoud dat je op verschillende manieren naar kwaliteit kunt kijken met behulp van de kwaliteitsattributen. (K1)
LD4	Begrijp de noodzaak van testen. (K2)

Wat is Testen eigenlijk?

Voor testen kun je veel verschillende definities geven. De definitie die we hanteren in deze syllabus is de volgende: 'Testen is een proces dat inzicht geeft in- en adviseert over de kwaliteit en de daaraan gerelateerde risico's'. **[II].**

Hierin zitten twee belangrijke zaken, namelijk kwaliteit en risico's. Op basis van de beschreven definitie kun je de conclusie trekken dat we eigenlijk spreken over kwaliteit en dat testen een middel is om hier inzicht in te geven. Er zijn echter ook andere manieren om kwaliteit te bereiken. Testen is op zichzelf een vrij reactieve maatregel, omdat het plaatsvindt als je het product al hebt. Wat vaak nog zinvoller en efficiënter is dan testen is reviewen, omdat je dit vaak al kunt doen voordat het product ontwikkeld is. Door te reviewen kun je al in een vroeg stadium kwaliteit verhogen door de opgestelde requirements te verbeteren, dan wel missende requirements op te stellen of overbodige requirements te verwijderen.

Het tweede element uit de definitie van testen dat we willen uitlichten is risico's. Die risico's bestaan al, want je geeft er inzicht in met testen. Organisaties testen dus om risico's ten aanzien van hun bedrijfsvoering te vinden, daarna uit te sluiten of te mitigeren. Als er een risicoanalyse gedaan is voor het specifieke project, dan kun je nagaan in hoeverre deze risico's nog actueel zijn, maar testen kan ook nieuwe risico's aantonen. Andere redenen om te testen kunnen zijn: aan wettelijke eisen voldoen, om ervoor te zorgen dat hun bedrijfsproces, product en IT-oplossingen kwaliteit leveren.

Wat is Kwaliteit?

Kwaliteit definiëren we in dit verhaal als volgt: 'Kwaliteit is het geheel van eigenschappen en kenmerken van een product of dienst dat van belang is voor het voldoen aan vastgestelde of vanzelfsprekende behoeften' **[III].** Dit is de definitie die we gebruiken in deze syllabus. Een mooie definitie, maar je moet hem even een paar keer lezen om hem goed te begrijpen. Kwaliteit bestaat dus uit eigenschappen en kenmerken, en kwaliteit komt overeen met verwachtingen die al dan niet vastgelegd zijn. Dat is hetzelfde gezegdin simpele bewoordingen. De lange definitie is opgenomen omdat hij wel mooi volledig is en wat dat betreft voldoet aan onzekwaliteitseisen.

Met deze definitie komen we op één van de lastige dingen van testen: de niet vastgelegde behoeften. Vaak leggen we nog wel een lijst met requirements vast, maar er zijn ook behoeften

die voor sommige betrokkenen erg voor de hand liggen maar voor andere betrokkenen niet zo. Denk als voorbeeld aan het sluiten van een applicatie met een kruisje in een Windows-omgeving of in een Apple-omgeving. Dat gaat precies andersom.

Kwaliteitsattributen

Het eerste deel van de definitie ‘eigenschappen en kenmerken’ kunnen we uiteenzetten in kwaliteitsattributen. Kwaliteitsattributen zijn verschillende manieren om naar kwaliteit te kunnen kijken. De ISO 25010 **[IV]** norm beschrijft een aantal kwaliteitsattributen die ons kunnen helpen bij het testen van software. Zonder volledig te zijn: functionaliteit, beveiligbaarheid (security), performance, gebruiksvriendelijkheid (usability), onderhoud baarheid (maintainability) en overdraagbaarheid (portability) zijn enkele kwaliteitsattributen uit de norm, die je het meest tegenkomt. We kiezen er bewust voor om nu niet alles op te nemen, want dat zou het hier te uitgebreid maken. De indeling en onderverdeling van kwaliteit in deze norm is echter zeer bruikbaar bij het testen van software. In een later hoofdstuk gaan we nog uitgebreider op enkele kwaliteitsattributen in.

Afhankelijkheid van Software

Software is tegenwoordig overal, je kan vrijwel niet meer zonder. Denk er maar eens aan om een dag zonder je smartphone te moeten leven. Dat is voor veel mensen behoorlijk wennen. Nu kun je wellicht nog wel zonder je telefoon, maar het leven met Smartphone is een stuk sneller en makkelijker. Naast je eigen afhankelijkheid van de software op je telefoon is ook de samenleving in hoge mate afhankelijk van software. Denk maar eens aan alle geautomatiseerde processen van de overheid zoals bijvoorbeeld het innen van belastingen. Als dat weer zonder computers moet plaatsvinden duurt het proces veel langer, kost het meer en is de inzet van veel extra personeel nodig. Dit is één voorbeeld, maar software regelt ook de dienstregeling van treinen en tevens de toegangscontroles en abonnementen voor diezelfde treinen. Het zal wennen zijn als dat (weer) zonder software zou moeten. Omdat we willen dat de samenleving blijft functioneren, belasting geïnd wordt, treinen volgens dienstregeling blijven rijden, toegangscontrole en incasso’s van abonnementen blijven functioneren, moeten we zorgen dat software betrouwbaar is en blijft functioneren. Daarnaast willen we dat als software functioneert het dan ook functioneert zoals bedoeld, juist omdat we erop vertrouwen. Neem bijvoorbeeld de software die ervoor zorgt dat de airbags in een auto afgaan bij een ongeval. We willen dat dit exact zo gebeurt als dat het bedoeld is. Bij te laat afgaan van de airbags zal er schade zijn, maar bij het te vroeg of onverwacht afgaan ook. In dit voorbeeld gaat het om letselschade, maar de schade kan ook imagoschade, andere materiële of immateriële schade, of financiële schade zijn. Vanwege deze en nog vele andere voorbeelden is het dus absolute noodzaak om software te testen. Om schade te voorkomen, om kwaliteit aan te tonen, om risico’s te mitigeren en inzichtelijk te maken, en om continuïteit te waarborgen.

Definities

Testen	Testen is een proces dat inzicht geeft in en adviseert over de kwaliteit en de daaraan gerelateerde risico’s.

Kwaliteit	Kwaliteit is het geheel van eigenschappen en kenmerken van een product of dienst dat van belang is voor het voldoen aan vastgestelde of vanzelfsprekende behoeften.
Kwaliteitsattribuut	Een kwaliteitsattribuut beschrijft een kenmerk van een informatiesysteem.
Risico	Een factor bestaand uit kans en impact die consequenties kan hebben.
Functionaliteit	De geschiktheid van een product of dienst; in hoeverre en op welke wijze de gewenste taak wordt uitgevoerd.
Onderhoudbaarheid (Maintainability)	De mate waarin een product of systeem effectief en efficiënt gewijzigd kan worden door de aangewezen beheerders.
Overdraagbaarheid (Portability)	De mate waarin een systeem, product of component effectief en efficiënt overgezet kan worden van één hardware, software of andere operationele of gebruiksomgeving naar een andere.
Review	Een activiteit waarbij een product of proces geëvalueerd wordt met als doel fouten te vinden of verbeteringen door te voeren.

Hoofdstuk 2: De Juiste Focus hebben om te Testen

LD5	Begrijp het verschil tussen de focus van ontwikkelaars en de focus van testers. (K2)

Als ontwikkelaar kan het lastig zijn om goed te kunnen testen. Je hoofddoel is vaak een werkend product maken op basis van de specificaties. Dit zal dan ook je focus zijn, gericht op het opleveren van een door de klant gedefinieerd en juist product. Doordat je erg gefocust bent op min of meer 1 ding, de oplossing, zul je randzaken eromheen minder snel zien.

Een tester heeft juist een heel ander doel, namelijk het op alle mogelijke manieren kijken naar de applicatie, om te kijken of deze juist functioneert. Dat is helemaal het tegenovergestelde van wat je als ontwikkelaar doet. Er is niet voor niets een functiescheiding ontstaan tussen testers en ontwikkelaars. Het is namelijk niet eenvoudig om deze twee verschillende invalshoeken tegelijk te combineren.

Er zijn echter wel wat handvatten die je kunt gebruiken om dit makkelijker te maken. Je kunt bijvoorbeeld met een collega afspreken dat je elkaars werk test, om op die manier een meer onafhankelijke blik mogelijk te maken en je los te maken van de ontwikkel-focus. Wat je ook kunt doen, is voor jezelf een aparte testdag te nemen, of een apart moment op de dag in te ruimen waarop je bewust niet begint met ontwikkelen, maar vanuit de test-focus start. De focus is een belangrijk ingrediënt hiervoor. In het komende hoofdstuk volgen een aantal concrete handvatten die je kunt toepassen vanuit die focus.

Als beheerder kun je met deze wetenschap ook beter begrijpen waarom er soms wel heel goed en soms veel minder goed getest wordt. Wat je kunt doen is een partij die ontwikkelt bevragen over hun testproces: of ze testers in dienst hebben, of hun ontwikkelaars ook testen. Als je nog in de fase van contractonderhandelingen zit kun je hier zelfs requirements voor neerleggen. Mocht je als beheerder zelf softwareontwikkeling of configuratie doen dan kun je bovenstaande tips voor ontwikkelaars natuurlijk zelf ook toepassen.

Hoofdstuk 3: De 8 Test Practices

LD6	Pas een aantal belangrijke testpractices toe. (K3)
LD7	Onthoud een aantal specifieke testtermen. (K1)

Om de materie van het testen zo toegankelijk mogelijk te maken hebben we een aantal practices geformuleerd. Deze practices zijn gebaseerd op praktijkervaringen in de testwereld. Deze practices zijn gekoppeld aan een aantal concreet toepasbare activiteiten. Deze practices met hun activiteiten kunnen je helpen bij het verhogen van de kwaliteit van software. Als je deze practices altijd hanteert, wordt de kans op fouten significant kleiner. Ook als niet-tester ben je dan toch in staat om de kwaliteit van de software, het proces en de requirements te verhogen. De practices waar we jullie langs mee wil nemen zijn:

1. Geen aannames doen.
2. Testen is logisch nadenken.
3. Alles testen is onmogelijk.
4. Wees specifiek.
5. Test zo vroeg mogelijk.
6. Begin klein en maak je testen steeds groter.
7. Leg je testen vast.
8. Begrijp het belang van communicatie in je activiteiten als tester.

In de hoofdstukken hierna worden de uitgangspunten stuk voor stuk toegelicht. Naast deze practices is er nog een belangrijk uitgangspunt dat altijd geldt: testen is altijd afhankelijk van de context. En met de context bedoelen we dan de omgevingsfactoren en randvoorwaarden. Omgevingsfactoren zijn onder andere de specifieke branche of sector waarin de test plaatsvindt, de aard van de software die je aan het maken bent, de wettelijke vereisten, voorschriften, richtlijnen en industriestandaards die gelden, maar ook het karakter van de organisatie of het project waarbinnen de testen plaatsvinden. De randvoorwaarden zijn onder andere tijd, geld en kwaliteit.

Let er op dat je zowel zelf deze practices kunt hanteren, als dat je weet dat anderen, die software voor jouw organisatie bouwen of testen, deze practices zouden moeten hanteren. Je kunt dit dus altijd toepassen naar een leverancier toe, door erom te vragen of door het als voorwaarde mee te geven.

Toepasbaarheid in Agile testing en Devops

In moderne, volwassen IT-organisaties zul je soms zien dat testen en kwaliteit geëvolueerd is en dat het lijkt of een aantal van de zaken in deze syllabus niet of minder worden toegepast. In sommige gevallen kan dit best het geval zijn. En het is een feit dat er andere tools en middelen zijn om ook toe te passen in deze omgevingen. Het doel van deze syllabus is echter duidelijk de basis, essentie van testen en softwarekwaliteit neer te zetten. Tevens willen we testen

toegankelijk maken voor een groep die nog onervaren is in het vakgebied. Ook in organisaties met een hoger volwassenheidsniveau of met een Agile of DevOps proces zou je deze basis aan de hand van de practices kunnen toepassen. Het verschil zal echter zijn dat je per sprint of per user story je test strategie zult moeten bekijken en mogelijk voor die sprint of user story specifiek zult moeten aanpassen. Zou je in je sprint werken aan stories met een hoog risico, zou je een zwaardere test techniek kunnen kiezen, die meer dekking bereikt bijvoorbeeld.

Of in het geval van DevOps zou je als voorbeeld kunnen nemen dat je de verschillende test levels in misschien wel dezelfde omgeving uitvoert. Maar dat je ze in ieder geval wel uitvoert, zodat het 'vergeten' ervan niet leidt tot een verminderde kwaliteit. Voor toepassing van andere tools en middelen in een Agile of DevOps omgeving verwijzen wij dan ook graag naar meer uitgebreide trainingen toegespitst op de methodieken. Bijvoorbeeld TMap, ISTQB, Agile United of DevOps United, die complementair kunnen zijn op deze syllabus.

Practice 1: Geen Aannames doen

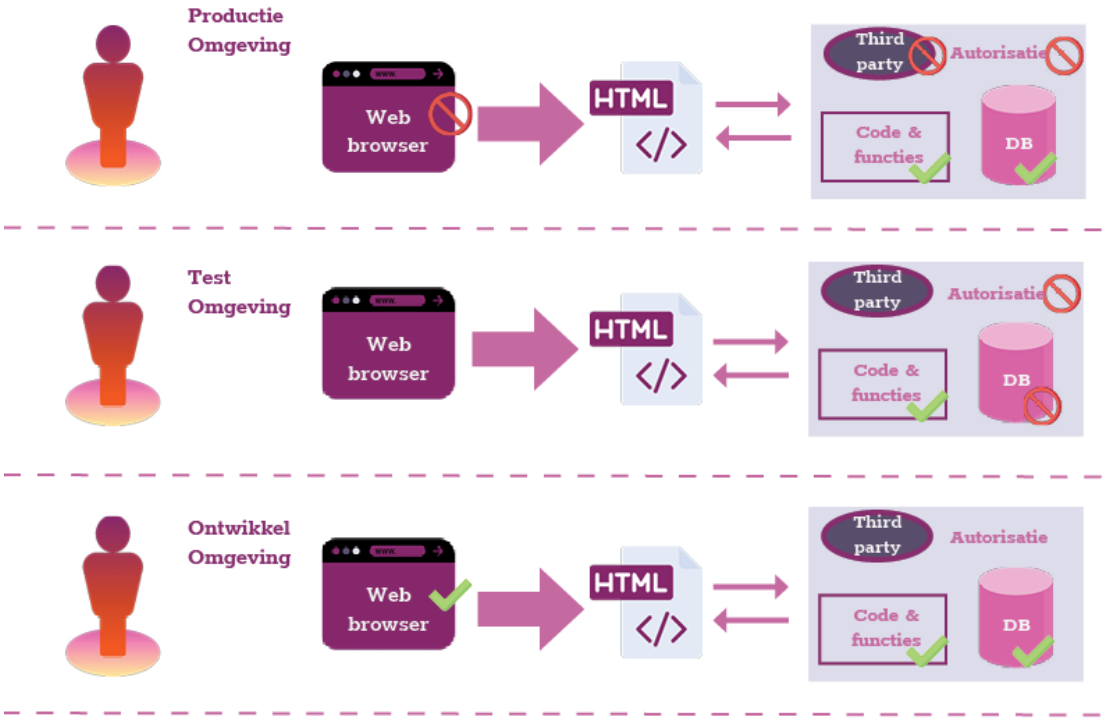
LD8	Begrijp de test practice van geen aannames doen. (K2)

De basisregel voor alle testers: geen aannames doen. Vraag het voor de grap maar eens na aan een willekeurige collega die tester is. Dat is het antwoord dat je zult krijgen. Als tester dien je een uiterst kritische houding hebben ten opzichte van hetgeen je aan het testen bent. Je moet over een gezonde portie wantrouwen beschikken. Eigenlijk moet je als tester in principe alles controleren. Je controleert of het systeem werkt zoals beschreven is, maar je kunt ook controleren of hetgeen beschreven is, ook hetgeen is waar de klant behoefte aan had. Dit eerste wordt in de testwereld verificatie genoemd. Verificatie is het proces om te controleren dat de software doet wat gespecificeerd is. Het tweede heet validatie. Validatie is het controleren of de specificatie doet wat de klantbehoefte en/of verwachting is. **[V]**

Als je wilt kun je hier heel ver in gaan, maar waar ga je stoppen? Soms zul je er inderdaad vanuit moeten gaan dat hetgeen dat beschreven is, ook is wat de klant wenste. Soms zul je ervan uit moeten gaan dat de specificaties van de leverancier kloppen. Maar ga er in de basis vanuit dat alles wat over de software beschreven is, of gezegd wordt, geverifieerd moet worden. En dat ook gevalideerd moet worden, dat wat de klant wenste, datgene is wat gebouwd is. En dat die ontwikkelaar of beheerder van de andere partij het ook op de manier ontwikkeld heeft die hij van tevoren aan je bevestigde.

Een typisch voorbeeld van een aanname is ervan uitgaan dat als iets op één omgeving werkt, het op de andere omgeving ook werkt. Dit is namelijk heel vaak niet zo. Als het gaat om configuratiemanagement wordt er bijvoorbeeld niet altijd stilgestaan wordt bij verschillende versies van webcomponenten op verschillende omgevingen.

Neem als voorbeeld een relatief eenvoudig systeem. Waarbij we een informatiesysteem hebben dat bestaat uit een database, code en functies en een third-party component die allemaal gepubliceerd worden op een webpagina. In de afbeelding hieronder (figuur: overzicht en omgevingen) is het visueel weergegeven. De groene vinkjes in de afbeelding laten zien wat er goed gaat en wat er getest is. De rode rondjes geven de defects aan.



Figuur: Overzicht van omgevingen

Als je dit systeem aan het bouwen bent, kun je dit testen op je ontwikkelomgeving en kun je dat heel grondig en volledig doen. En dat is natuurlijk ook hoe het zou moeten.

Echter als je dit gedaan hebt moet je het op de testomgeving gaan zetten. Omdat je getest hebt dat de functionaliteit werkt, ben je er zeker van dat het op test ook werkt. Maar wat als je vergeet om een component over te zetten? Of wat als je een ander adres moet gebruiken voor de third-party software (denk aan bepaalde plug-ins)? En als je een ander certificaat nodig hebt in een andere omgeving? Ook de autorisaties zullen opnieuw toegekend moeten worden. Staan deze gelijk goed? Hoe weet je dan dat alle verschillende rollen die je gedefinieerd had ook werken? Misschien had je je test gedaan met de admin-rol en werkte het prima, maar nu je met een gewone user kijkt werkt het ineens niet meer. Tevens kan de klant beschikken over een sterk verouderde of niet geteste webbrowser, waardoor er toch fouten ontstaan die je niet gevonden hebt met je test in de ontwikkelomgeving.

Dat je al die vragen kunt stellen, wil niet zeggen dat je de gehele test moet herhalen op de testomgeving en daarna nog een keer op productie. Dat hoeft niet want je hebt al aangetoond dat de functionaliteit werkt. Maar als je in de ontwikkelomgeving uitgebreid alles getest hebt, kun je op de volgende omgeving een aantal gevallen testen waarmee je aantoont dat de oplevering volledig is en dat alle componenten er staan. Belangrijk om te weten is dat als je dynamisch verifieert of de functionaliteit werkt, je de grootste kans hebt dat het goed gaat. Als dynamisch verifiëren niet kan, kun je ook statisch controleren of valideren. Daarnaast is het belangrijk om te communiceren. Bij twijfel of iets de bedoeling is, neem contact op met de tester of de opsteller van het requirement.

Voor nu is het belangrijk om te onthouden dat je **geen aannames** doet. Wat we verder nog kunnen toevoegen is dat het voor jou als ontwikkelaar of beheerder nu afgelopen is met aannames als: 'testen is moeilijk', 'testen is voor testers', 'er is geen tijd voor testen', 'ik heb alle unittesten gedaan', 'er is goede logging', 'de testers moeten ook wat te doen hebben', 'testen is saai' en 'het werkt op mijn omgeving'. Vooraf kon je je nog verschuilen achter het feit dat je het niet wist. Na het volgen van deze training kan dat echter niet meer.

Samengevat: probeer indien mogelijk alles te verifiëren. Doe dat als het even kan dynamisch, door het uit te voeren. Let er wel op dat je geen dingen dubbel doet. Je zult je code bijvoorbeeld niet meer heel uitgebreid hoeven te unittesten op de test- of productieomgeving. Als het niet mogelijk is om dynamisch te verifiëren, probeer het dan te valideren. Het belangrijkste is in ieder geval dat je communiceert over wat je wel en niet doet tijdens het testen.

Definities

Aanname	Een veronderstelling of uitgangspunt c.q. hypothese, die niet bewezen is.
Configuratiemanagement	De versies van verschillende werkproducten in het softwareontwikkelproces. Het zorgt ervoor dat je kunt traceren welke versie van een requirement bij welke versie van de software hoort.
Dynamisch testen	Testen waarbij je code van een component of een systeem uitvoert **[V]**.
Testomgeving	Een omgeving die hardware, instrumentarium, simulatoren, softwareprogramma's en andere ondersteunende elementen bevat die nodig zijn om een test uit te voeren.
Validatie	Het controleren of de specificatie doet wat de klantbehoefte/verwachting is.
Verificatie	Het proces om te controleren dat de software doet wat gespecificeerd is.
Statisch testen	Beoordelen/testen van een onderdeel van het software ontwikkelingsproces zonder code uit te voeren. Bijvoorbeeld een requirement reviewen, code reviewen, een statische analyse laten doen **[V]**.

Practice 2: Testen is Logisch Nadenken

LD9	Begrijp de practice dat testen logisch nadenken is. (K2)
LD10	Begrijp het belang van regressietesten. (K2)
LD11	Onthoud de voor- en nadelen van testautomatisering. (K1)

Softwaretesten is niet heel ingewikkeld. Enige ervaring erin helpt zeker, maar het komt eigenlijk neer op gewoon wat gezond verstand. Neem onderstaande bewering:

'ALS het regent, DAN word je nat'.

Als je dit requirement wilt ontwikkelen, kun je dit vrij snel omzetten in code. Daarna voer je hier waarschijnlijk een unittest voor uit, zoals je gebruikelijk doet als je software ontwikkelt. Hoewel dat laatste uiteraard een aanname is. Het kan natuurlijk nooit kwaad om te vragen of en hoe iemand getest heeft. Bedenk echter dat als je aan het ontwikkelen bent, je de focus hebt om zaken te realiseren, requirements om te zetten in code. Je zal je dus voornamelijk focussen op de oplossing. Als je je wilt verplaatsen in een tester moet je het requirement echter heel anders benaderen. Wat bijvoorbeeld als het niet regent? Word je dan ook nat? En als het wel regent, maar als je dan niet nat wordt? Wat moet er gebeuren in die situatie?

Als tester zou je de volgende zaken willen controleren:

Scenario	Het regent	Je wordt nat
1.	Waar	Waar
2.	Waar	Onwaar
3.	Onwaar	Waar
4.	Onwaar	Onwaar

Tabel: mogelijke uitkomsten van een bewering

Je wilt valideren dat het 2e scenario niet voor kan komen en dat de overige scenario's wel voor kunnen komen om te bevestigen dat de bewering waar is.

In het voorbeeld hierboven gaat het niet over IT. Daarom als voorbeeld nu het volgende requirement:

*'ALS **beheerder** van dit systeem DAN wil ik **gebruikers** kunnen wijzigen'*

Scenario	Ik ben beheerder	Ik kan gebruikers wijzigen
1.	Waar	Waar
2.	Waar	Onwaar
3.	Onwaar	Waar
4.	Onwaar	Onwaar

Tabel: mogelijke uitkomsten van een bewering

Als je dit voorbeeld ziet, begrijp je onmiddellijk dat het niet wenselijk is om als niet-beheerder gebruikers te kunnen opvoeren, wijzigen en verwijderen.
Je moet dat dus zeker wel testen. Probeer daarom naast je focus om het requirement te realiseren daarnaast ook te focussen op de 'Wat-Als' of 'Onwaar situaties'.

Als je als voorbeeld neemt het uploaden van een .pdf-bestand op een website wordt het nog duidelijker. Neem het volgende requirement:

*'ALS **beheerder** DAN wil ik een .pdf-bestand kunnen toevoegen aan een website.'*

Wanneer je kijkt wat je moet realiseren als ontwikkelaar is dit beperkt. Als tester wil je in dit geval echter veel meer testen, omdat je begrijpt dat het uploaden van bestanden met een andere extensie wel kwalijke gevolgen kan hebben. Ook wil je controleren dat als je de rol beheerder niet hebt je geen bestanden kunt uploaden. De volgende tabel illustreert wat we hiermee bedoelen.

	1	**2**	**3**	**4**
Ik ben beheerder	J	J	N	N
De file is een .pdf	J	N	J	N
Bestand kan geüpload worden	X			
Geen actie		X	X	X

Tabel: beslistabel voor het uploaden van een bestand

Hier zie je dat je in ieder geval graag 4 testcases wilt gaan doen. Maar een snelle inventarisatie leert dat er naast 'beheerder' en 'gebruiker' nog een aantal rollen is. Je zou pad 3 eigenlijk met al die rollen willen testen. Tevens is het zo dat er meerdere voor de hand liggende bestandsextensies zijn die je absoluut wilt uitsluiten vanwege juridische of veiligheidsredenen. Denk bijvoorbeeld aan .doc, docx, .ppt, .exe, .zip, .rar. Dit leidt tot meer situaties in pad 2 dan je op voorhand zou zeggen.

Hiernaast is alleen het controleren op extensie wellicht iets te kort door de bocht. Een pdf-bestand kan ook nog schadelijke inhoud bevatten. Je zou dus eigenlijk moeten voorstellen om het requirement uit te laten breiden met een aanvullende controle op de integriteit van het bestand. Maar dit laten we voor nu even buiten beschouwing.

Je kunt ook beargumenteren dat er geen reden is om een uitgebreide controle te doen omdat het hier een beheerder betreft. De rechten voor een beheerder zijn immers alleen voorbehouden aan een aantal personen die over het algemeen goed weten wat ze moeten doen. Echter de functionaliteit hier kan ook elders hergebruikt gaan worden, of het wachtwoord voor een beheerder kan achterhaald worden.

Regressie

Je merkt ondertussen al dat voor relatief simpele functionaliteit best veel gronden kunnen ontstaan om iets te gaan testen en dat de test best groot kan gaan worden. Stel je nu voor dat het requirement in het voorbeeld aangepast wordt nadat het testtraject voor het oorspronkelijke requirement is afgerond. De aanpassing zou kunnen zijn dat een andere bestandstype nu ook toegestaan wordt. De nieuwe functionaliteit komt er als volgt uit te zien:

	1	2	3	4	5	6	7	8
Ik ben beheerder	J	J	J	J	N	N	N	N
De file is een .pdf	J	J	N	N	J	J	N	N
De file is een .doc	J	N	J	N	J	N	J	N
Bestand kan geüpload worden	X	X	X					
Geen actie				X	X	X	X	X

Tabel: beslistabel voor het uploaden van een bestand met een nieuw requirement

Waar je dacht dat je in de oorspronkelijk situatie al veel testcases had, zie je ze nu toenemen. Je kunt niet aannemen dat de wijzigingen in de code niet onbedoeld bijwerkingen hebben gehad in de bestaande functionaliteit en je moet daarom de bestaande testcases nog een keer uitvoeren. De testen voor het controleren of bestaande functionaliteit ongewijzigd is, heten regressietesten. Het is van belang om bij elke nieuwe aanpassing goed na te denken over de functionaliteit die je op regressie zou willen testen. Regressietesten lenen zich over het algemeen erg goed voor automatisering omdat ze meerdere keren uitgevoerd worden.

Testautomatisering

Voordat tests kunnen worden geautomatiseerd, zijn er altijd handmatige tests nodig om de testcases te kunnen maken. Je kunt zodoende te weten komen wat er nodig is om geautomatiseerde testscripts te maken. **Testautomatisering** verwijst meestal naar de geautomatiseerde uitvoering van tests, maar dit is een vrij enge definitie. De definitie zou misschien kunnen worden uitgebreid tot ‘het automatiseren van delen van het testproces’, omdat het vaak wenselijk is tijdrovende taken in het hele testproces op een eenvoudige en kosteneffectieve manier te automatiseren. Denk hier bijvoorbeeld aan het deployen van software van de ene omgeving naar de andere. Hoewel er veel voordelen verbonden zijn aan het automatiseren van testtaken, zijn er ook argumenten te bedenken die tegen testautomatisering pleiten. Om iets in het software ontwikkelproces te kunnen automatiseren is bijvoorbeeld een volwassen proces nodig, evenals een investering van tijd, geld en middelen. Dit kan betekenen dat het enige tijd kan duren voordat de initiële investering is terugverdiend.

Testautomatisering kan waardevol zijn, bijvoorbeeld wanneer grote aantallen (regressie)tests in volwassen complexe softwareomgevingen gedurende langere tijd worden uitgevoerd. Andere voorbeelden van situaties die zich lenen voor testautomatisering zijn grote projecten waarin dezelfde gebieden steeds opnieuw moeten worden getest (projecten met veel iteraties).

Testautomatisering kan ook nuttig zijn bij projecten die al een eerste handmatig testproces hebben ondergaan. De implementatie van testautomatisering voor een volwassen softwareproduct kan de kostenefficiëntie van het testen verbeteren en de totale testdekking vergroten.

In het algemeen is testautomatisering goed te gebruiken als de fundamentele testprocessen volwassen zijn, of met andere woorden, als er een gevestigde teststrategie is, met verschillende soorten tests die de verschillende testdoelstellingen dekken.

Enkele typische eigenaardigheden van testautomatisering om rekening mee te houden:

- Automatisering is een middel; het wordt nog wel eens gezien als een doel.
- Bij testautomatisering ontstaat nog een applicatie die ontworpen, gebouwd, geleerd, beheerd en ondersteund moet worden.
- Met testautomatisering bouw je vaak een applicatie die afleidt van de applicatie die je wilde testen.
- Geautomatiseerde testscripts vereisen onderhoud, en het over het hoofd zien van onderhoud kan de scripts onbeheersbaar maken.
- Bedenk dat je voordat je een testgeval kunt automatiseren, deze eerst handmatig ontworpen zal moeten worden.
- Het nadenken over en het opstellen van een teststrategie geeft je ook een beter inzicht in welke delen van het systeem zich goed lenen voor testautomatisering. Naast regressietesten zijn dat ook vaak testen waarbij de applicatie getest dient te worden op verschillende besturingssystemen, webbrowsers en devices.

Er zijn veel voordelen aan het automatiseren van bepaalde tests: de sleutel tot succesvol testautomatisering is echter het vinden van de balans tussen wat het meest geschikt is voor automatisering en wat vooral ook niet. Soms is **automatisering van delen van het testproces** zelfs logischer en geschikter dan testautomatisering zelf. Testautomatisering is een uitgebreid onderwerp en als je geïnteresseerd bent, zijn er nog meer volledige cursussen over dit onderwerp, inclusief tools zoals Selenium.

Testtooling

Er zijn veel tools beschikbaar die goed kunnen ondersteunen bij verschillende onderdelen van het testproces of verschillende onderdelen van de SDLC. Bij het vastleggen van testgevallen en defects en ook voor configuratiebeheer, kunnen geschikte hulpmiddelen zeer nuttig zijn. Voor performancetesten kan zelfs worden beweerd dat tooling een noodzaak is. Houd er rekening mee dat, hoewel tools soms goede oplossingen bieden, zij vaak een breed spectrum van taken van de SDLC bedienen, zodat zij misschien niet de exacte functionaliteit bieden die aan jouw behoeften voldoet.

Als je toch tools gaat gebruiken, doe dan een pilot en een goede risicoanalyse van de tool, en overweeg zorgvuldig de (licentie)kosten en de kosten om de tooling te gaan beheren. Tools zijn niet altijd wat ze lijken en de kosten zijn niet altijd duidelijk gespecificeerd. Er zijn ook risico's

verbonden aan open-source tools, die misschien geen toegewijd ondersteuningsteam hebben. Probeer te voorkomen dat er extra uitgebreide projecten worden opgezet alleen maar om de implementatie van tools te bewerkstelligen.

Terugkomend op het idee van ‘testen is logisch denken’, dit sluit aan bij de concepten van testautomatisering en testtooling, waarbij voorafgaand aan de implementatie passende acties moeten worden ondernomen om ze succesvol te maken. Aangezien het uiteindelijke doel van elk project kwaliteitssoftware is, is het belangrijk de focus op dit doel te houden en je minder te richten op de implementatie van extra (mogelijk onnodige) hulpmiddelen en automatisering.

Definities

Actie	De gebeurtenis die plaatsvindt.
Automatisering van testen	Het automatiseren van delen van het software ontwikkel proces.
Conditie	Voorwaarde waaraan al dan niet voldaan kan worden.
Beslissing	Uitkomst van een conditie.
Testautomatisering	Het gebruik van software om testen uit te voeren of te ondersteunen.
Testtooling	Software of hardware die één of meerdere test activteiten ondersteunt.
Regressietesten	Het opnieuw testen van al bestaande functionaliteit met als doel vast te stellen of deze nog steeds zo functioneert als voorheen.
Requirement	Omschrijving van datgene wat benodigd is.

Practice 3: Alles Testen is Onmogelijk

LD12	Begrijp de practice dat alles testen onmogelijk is. (K2)
LD13	Begrijp het belang van een teststrategie. (K2)
LD14	Pas risicoanalyse toe in je testtraject. (K3)
LD15	Begrijp wat Pareto's analyse inhoudt. (K2)
LD16	Begrijp hoe root cause analyse je kan helpen je software ontwikkelproces te verbeteren. (K2)

In de software en applicaties die gebouwd worden zitten talloze keuzemomenten en mogelijkheden. Daarnaast kun je applicaties op talloze devices bekijken. Zelfs een relatief simpele **application programming interface (API)** kent al snel enkele tientallen mogelijkheden om zaken in te voeren. Het is dus onmogelijk om alle testcases die er zijn uit te voeren. Je zult daarom keuzes moeten maken en hier onderscheid in aan moeten brengen. Het lijkt misschien lastig om dit te gaan doen, maar er zijn enkele trucjes voor.

Een praktijkvoorbeeld om dit te illustreren is het wijzigen van een telefoonnummer-veld. Natuurlijk zijn er richtlijnen voor het noteren van telefoonnummers, alleen moeten die wel opgevolgd worden. Dit gebeurt natuurlijk lang niet altijd; niet door gebruikers die het met de hand invoeren en ook niet door applicatieontwikkelaars die tegen de applicatie aan ontwikkelen. Ga er dus vanuit dat je een telefoonnummer in veel verschillende formaten aangeleverd kunt krijgen. Zie (zie tabel Voorbeelden testcases telefoonnummers) voor een aantal voorbeelden:

Land	**Notatie voorbeelden**	
Nederland	+31 050 360 0233	00(31)050 360 0233
	+(31) 050 360 0233	00(31)50 360 0233
	+31 50 360 0233	00 31 050 360 0233
	+31 (0)50 360 0233	00 31 50 360 0233
Verenigde Staten	+(1)(425) 555-0100	+14255550100
Verenigd Koninkrijk	+(44)(20) 1234 5678	+442912345678
China	+(86)(10) 1234 5678	+861012345678
Singapore	+(65) 1234 5678	+6512345678

Tabel: Voorbeelden formaten van telefoonnummers

In dit voorbeeld kon in de oude situatie een API op een aantal manieren aangesproken worden, waardoor er onderscheid te maken was tussen vaste nummers en mobiele nummers, via een aantal verschillende SOAP (Simple Object Access Protocol) -berichten en via een aantal verschillende REST (REpresentational State Transfer) -berichten.

Er zaten afhankelijk van het SOAP of REST bericht, twee of vier telefoonnummervelden in deze berichten, waarbij het veld dat voor mobiel was altijd diende te starten met 06. Tevens zat er

een strenge controle op het veld: max 10 posities, altijd beginnend met een + gevolgd door een tweecijferige landcode die in dit geval ook nog de vaste waarde van 31 moest hebben. Al die controles worden losgelaten in de nieuwe situatie. Het veld mag nu 20 posities zijn. Ook de landcode en 06 worden losgelaten. Je mag dus een vast nummer in het voormalige mobiel-veld invoeren en visa versa. Als je nu kijkt naar de manier waarop je een telefoonnummer op kunt schrijven dan kom je op een oneindig aantal situaties (zie tabel Voorbeelden testcases telefoonnummers).

Merk hierbij op dat wat in de tabel weergegeven is slechts een deel van de mogelijke vaste nummernotaties is en dat mobiel nog buiten beschouwing gelaten is. Dit kan een ontwikkelaar redelijk eenvoudig oplossen door code toe te voegen in het veld die alle speciale tekens eruit haalt en een '+' toevoegt. Als tester moet je anders best veel moeite doen om alle valide input te controleren. En eigenlijk zou je dit ook nog in al die verschillende velden moeten testen. Bij elkaar kom je snel op tienduizenden testsituaties.

Het is onmogelijk om al die (functionele) situaties te testen, want dat duurt meestal te lang of kost gewoon te veel. Je zult keuzes moeten maken over wat nodig is om te testen. Het moet duidelijk zijn waaruit het testobject (telefoonnummerveld) technisch bestaat. Dat helpt om een duidelijk beeld te krijgen van wat er getest moet worden. Dit kan dan gedocumenteerd worden in een teststrategie.

Wat is een Teststrategie?

Een teststrategie kan worden beschouwd als de tactische richtlijnen, die alle ins en outs moeten uitleggen rond alles wat moet worden getest gedurende de SDLC. Een belangrijk aspect om te onthouden bij het creëren van een teststrategie is ervoor te zorgen dat het nuttige inzichten biedt in het **testobject**.

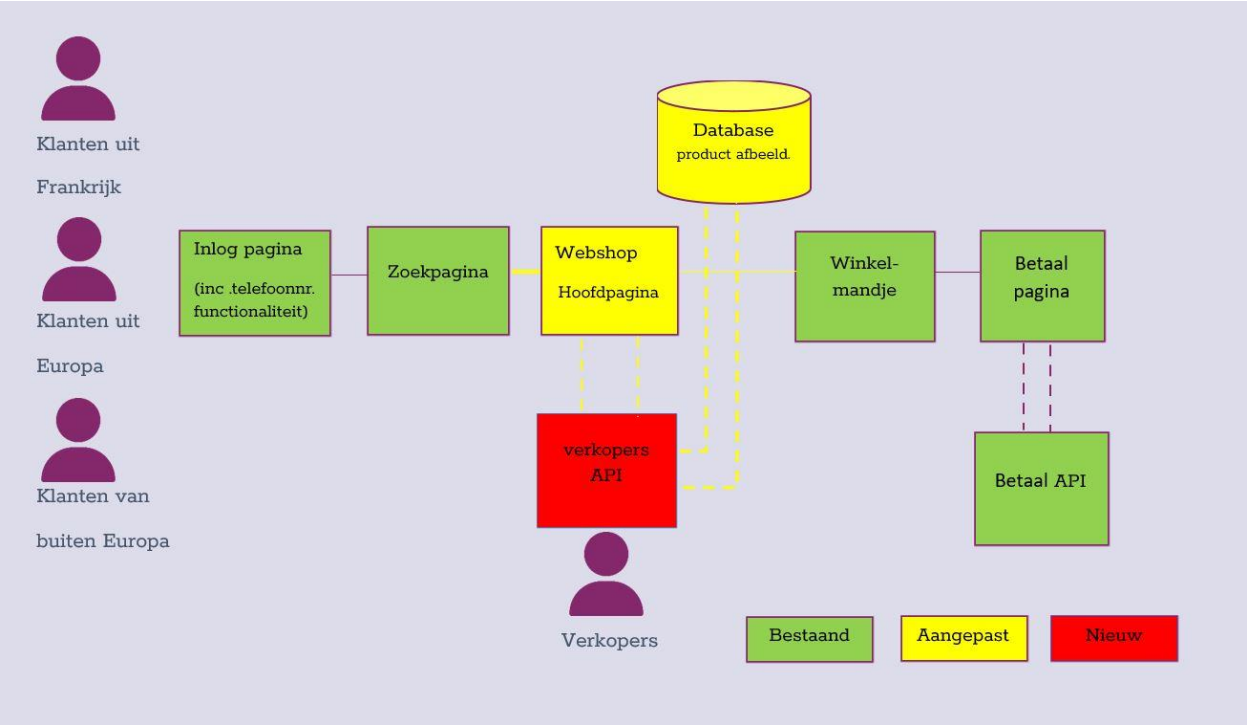
Opstellen van een Teststrategie: Informatie Verzamelen

Handige informatie bij het creëren van een teststrategie (en het bepalen van de testdoelen):

- Requirements: om als testbasis te dienen voor onze tests, bijvoorbeeld (high-level) designs, epic's, userstory's, use cases, technische documentatie, gebruikershandleidingen etc.
- Risico's: om beter te begrijpen waar potentiële bedreigingen van het systeem zitten. Deze kunnen uit bestaande risicoanalyses worden gehaald.
- Kwaliteitsattributen: om beter te kunnen bepalen op welke gebieden extra aandacht nodig is, bijvoorbeeld als financiële en persoonlijke informatie van klanten wordt verwerkt (Beveiliging), of als veel gebruikers die het systeem tegelijkertijd gebruiken om tickets te kopen (Performance), enz.
- Bestaande high-level testcases: om als voorbeeld te dienen. Deze kunnen worden overgenomen uit een bestaande regressietestset.
- Incident rapporten (servicedesk-tickets): om veel voorkomende problemen in het proces of systeem weer te geven, waar rekening mee gehouden moet worden.

Opstellen van een Teststrategie: het Systeem Schetsen

Het kan nuttig zijn om een schets op te nemen (of te tekenen) die beschrijft hoe het testobject werkt (inclusief de bestaande onderdelen en hun omgeving). Het opnemen van een schets van het systeem kan helpen om duidelijk te maken welke delen van het systeem nieuw zijn, en welke delen al bestaan. Het kan ook nuttig zijn de schets een kleurcode te geven, zoals in dit voorbeeld:



Deze schets kan worden gebruikt voor het toewijzen van testlevels en testtypes, die we zullen bespreken in ‘Practice 6: Begin klein en breid je testscope geleidelijk uit’.

Als je eenmaal een schets hebt die uitlegt hoe het systeem werkt en meer inzicht hebben gekregen in de applicatie zelf, kan het nuttig zijn om de risicoanalyse ernaast te leggen zodat je zien waar risico’s optreden, en je de focus van je teststrategie zodoende aan kunt scherpen.

Opstellen van een eenvoudige Teststrategie met behulp van Risicoanalyse

Wanneer je besluit wat je gaat testen, moet je een eenvoudige teststrategie op basis van risico's ontwikkelen. Er zijn verschillende manieren om een dergelijke strategie te ontwikkelen, die kunnen variëren in lengte en omvang. Wij zullen enkele van de basis benaderingen uitleggen die helpen om een risico gebaseerde teststrategie op te stellen.

Om een risico gebaseerde teststrategie op te kunnen stellen, is het belangrijk dat we eerst definiëren wat een risico is. Volgens TMap **[XII]** is een risico samengesteld uit de twee

elementen ‘faalkans’ en ‘schade’. De faalkans bestaat uit de frequentie waarmee het proces wordt gebruikt en de complexiteit van het proces zelf. Beschouw de volgende formule:

Risico = faalkans (frequentie + complexiteit) * schade

Laten we eens kijken hoe je risico's het beste kunnen identificeren door middel van risicoanalyse.

Risicoanalyse gebruiken om een Teststrategie op te stellen

Risicoanalyse **[XIX]** is een proces dat bestaat uit drie vereiste stappen om een basis voor een op risico gebaseerde teststrategie te construeren: identificatie, analyse en mitigatie. Een goede bron voor het **identificeren van risico's** is de opdrachtgever (de applicatie-eigenaar). Als belangrijkste stakeholder begrijpt de opdrachtgever meestal het beste de bedrijfstak waarvoor we een systeem bouwen (of aanpassen). Daarom is de opdrachtgever vaak goed op de hoogte van de risico's en bijzonderheden van de betreffende bedrijfstak en kan hij helpen beslissen wat de belangrijkste risico's zijn. Het is ook belangrijk om bij het identificeren van risico’s andere belanghebbenden van de klantzijde te betrekken, bijvoorbeeld systeembeheerders, ontwikkelaars en key-users. Zij hebben meestal relevante inzichten voor het onderhoud, het gebruik en de ontwikkeling van de functionaliteit, waardoor zij kunnen ingaan op de gebruiksfrequentie, de complexiteit of veel voorkomende problemen met de functionaliteit. Deze inzichten kunnen de waarde van de **risicoanalyse** als geheel behoorlijk verbeteren. Zodra de risico's zijn geanalyseerd, kun je je gaan richten op het beperken van de risico's. Testen op zich is een risicobeperkende maatregel. Er zijn nog andere maatregelen om de risico's te beperken, bijvoorbeeld de implementatie van monitoring na invoering in de productieomgeving.

Met het vorige voorbeeld van het wijzigen van de functionaliteit van een telefoonnummerveld kan onze klant misschien bevestigen dat 90% van zijn inkomende bestellingen uit Europa komt. Door deze waardevolle informatie weet je dat een goed werkende functionaliteit zeer belangrijk is voor Europese telefoonnummers. Daarom kun je deze informatie gebruiken om de functionaliteit en dus de risico's te verdelen. Laten wij eens kijken naar de risico's die nu zijn vastgesteld:

Nr.	Risico
1.	De telefoonnummer functionaliteit binnen Europa werkt niet.
2.	De telefoonnummer functionaliteit buiten Europa werkt niet.

Het testen van alle telefoonnummerformaten van elk land in Europa is al veel werk, dus het is goed om de scope nog verder te verkleinen. Door de klant specifiekere vragen te stellen over de belangrijkste markten binnen Europa, kun je ontdekken welke telefoonnummerformaten het meest getest moeten worden. Als de klant bijvoorbeeld zegt dat 70% van de omzet uit Frankrijk komt, kun je het risico verder onderverdelen:

Nr.	Risico
1.	De telefoonnummer functionaliteit in Frankrijk werkt niet.
2.	De telefoonnummer functionaliteit in andere landen in Europa werkt niet.
3.	De telefoonnummer functionaliteit buiten Europa werkt niet.

Om ons voorbeeld eenvoudig te houden, blijven wij bij deze geïdentificeerde risico's (in werkelijkheid kunnen er nog veel meer zijn).

Nu kun je deze geïdentificeerde risico's bekijken en goed analyseren. Risicoanalyse houdt in dat men de kans op falen en schade leert kennen.

De faalkans bestaat uit de factoren frequentie en complexiteit. Met frequentie bedoelen we de interval waarmee het programma (of proces) wordt uitgeoefend. Bij het bepalen van de gebruiksfrequentie is het een goede vuistregel om de gebruikscijfers van de specifieke functionaliteit uit de applicatie in de productieomgeving te halen. Indien deze gegevens beschikbaar zijn, geven zij een feitelijk inzicht in het gebruik van verschillende functionaliteiten. Vaak kunnen die cijfers worden verkregen door te kijken naar logging of business intelligence rapporten voor de genoemde functionaliteit in productie. Door deze gegevens toe te voegen, krijgt je een meer feitelijk inzicht in de frequentie waarmee een bepaalde functionaliteit wordt gebruikt.

Met complexiteit bedoelen we de moeilijkheid om een bepaald programma (of proces) uit te voeren, of de complexiteit van het programma zelf. Als je de resultaten van frequentie en complexiteit combineert, kom je tot een eindscore voor de faalkans. Het is belangrijk te onthouden dat risicoanalyse altijd een subjectieve manier is om iets in te schatten. Deze subjectiviteit is nodig omdat je probeert een waarde toe te kennen aan de prioriteit die aan de gegeven risico's moet worden toegekend.

Bij risicoanalyse is het gebruikelijk een driepunts-schaal te hanteren met de waarden ‘hoog’ (H), ‘gemiddeld’ (M) en ‘laag’ (L) voor alle onderdelen. Het is belangrijk op te merken dat niet alles als ‘hoog’ kan worden aangemerkt, aangezien niet alles even belangrijk is. Beschouw de volgende risicomatrix voor ons voorbeeld van telefoonnummer functionaliteit:

Risico tabel			Functionaliteit	
			Frequentie	H
			Complexiteit	M
			Totaal	M
Proces	**System**	**Schade**		
Frankrijk	Webportal	**H**		A
Elders Europa	Webportal	**M**		B
Buiten Europa	Webportal	**L**		C

Risicomatrix: Telefoon Nummer Functionaliteit

In deze risicomatrix verwijzen het onderste gedeelte ‘Proces’, ‘Systeem’ en ‘Schade’ naar de onderverdeelde risico's. ‘Frankrijk’ staat onder Schade als ‘H’, omdat 70% van de orders hier vandaan komt. Als 10% van de orders uit ‘Buiten Europa’ komt, kunnen we via wiskundige deductie schatten dat 20% van de orders uit de rest van Europa komt (‘Overig Europa’). Vandaar dat ‘Overig Europa’ onder ‘Schade’ wordt vermeld als ‘M’ met 20%, en ‘Buiten Europa’ onder ‘Schade’ als ‘L’ met 10%. Tijdens de bespreking van het proces met de belanghebbenden is duidelijk geworden dat de ‘Frequentie’ ‘H’ is, omdat het veld voor elke bestelling wordt gebruikt om de identiteit van de betreffende klant te valideren. Aangezien het wijzigen van een telefoonnummerveld een vrij gebruikelijke en eenvoudige procedure is, zou men (merk op dat dit subjectief is) kunnen stellen dat de ‘Complexiteit’ ‘M’ is. De ‘Totale’ schatting van de faalkans voor dit risico is ‘M’. In de rechterbenedenhoek van de risicomatrix verwijzen de ‘A’, ‘B’ en ‘C’ naar het belang van de functionaliteit bij het beperken van het risico. ‘A’ verwijst naar de risico's die de meeste dekking moeten krijgen om het risico te beperken dat het bedrijf de meeste schade zou toebrengen, ‘C’ verwijst naar de risico's die de minste dekking vereisen (omdat het totale risico voor het bedrijf lager is), en ‘B’ wordt gebruikt voor de risico's die er tussenin liggen.

Testdoelen

Testdoelen (Engels: test objectives of test goals) kunnen een bepaalde functionaliteit zijn, een onderdeel van een softwaresysteem, of zelfs een kwaliteitsattribuut dat speciale aandacht moet krijgen tijdens het testen; daarom is er geen exacte wetenschap of regel om ze af te leiden.

Zoals je in het voorbeeld van de telefoonveldfunctionaliteit hebben gezien, kun je tijdens de risicoanalyse zien dat de verdeling van het risico in verschillende onderdelen een aantal waardevolle gezichtspunten over de functionaliteit heeft opgeleverd. Dit zijn ‘Bestellingen uit Frankrijk’, ‘Bestellingen uit andere delen van Europa’, en ‘Bestellingen van buiten Europa’. Hier hebben we alleen deze opgesomd, maar aangezien het voorbeeld een webshop betreft, waar je

artikelen kunt bestellen, is het duidelijk dat je ook enkele andere onderdelen van de webshop als potentiële (waardevolle) testdoelstellingen kunt benoemen. Enkele voorbeelden zouden kunnen zijn: 'de gebruiksvriendelijkheid van de interface', 'het winkelwagentje', 'de kassa', 'de betaalmogelijkheden', 'de zoekmachine', 'de performance', 'de API voor externe verkopers', 'de API-documentatie'. Dit zijn allemaal voorbeelden die belangrijk kunnen worden als je dieper in dit voorbeeld zou duiken.

De Risico Gebaseerde Teststrategie

Als je eenmaal weet waar de verschillende dekkingsniveaus nodig zijn, kun je het systeem opdelen in verschillende aandachtsgebieden (testdoelen) en op basis daarvan een teststrategie opstellen. Het creëren van een teststrategie gaat over het creëren van een gestructureerde aanpak van je testen, door het gebruik van verschillende testtechnieken die verschillende niveaus van dekking kunnen toepassen op de verschillende testdoelstellingen.

In het voorbeeld van telefoonnummerfunctionaliteit hebben we de testdoelen beperkt om het voorbeeld zo eenvoudig mogelijk te houden, maar als je een echte teststrategie voor dit voorbeeld zou creëren, moet je er rekening mee houden dat er veel meer testdoelen kunnen zijn waar je rekening mee moet houden.

Als bij het opstellen van de teststrategie en bijvoorbeeld bij het bekijken van incidentrapporten blijkt dat de lijst van testdoelen zeer lang is, of dat het erg uitdagend of schijnbaar onmogelijk wordt om ze in de teststrategie op te nemen, dan is de 'Pareto-analyse' een voorbeeld van een methode die nuttig kan zijn om je hierbij te helpen.

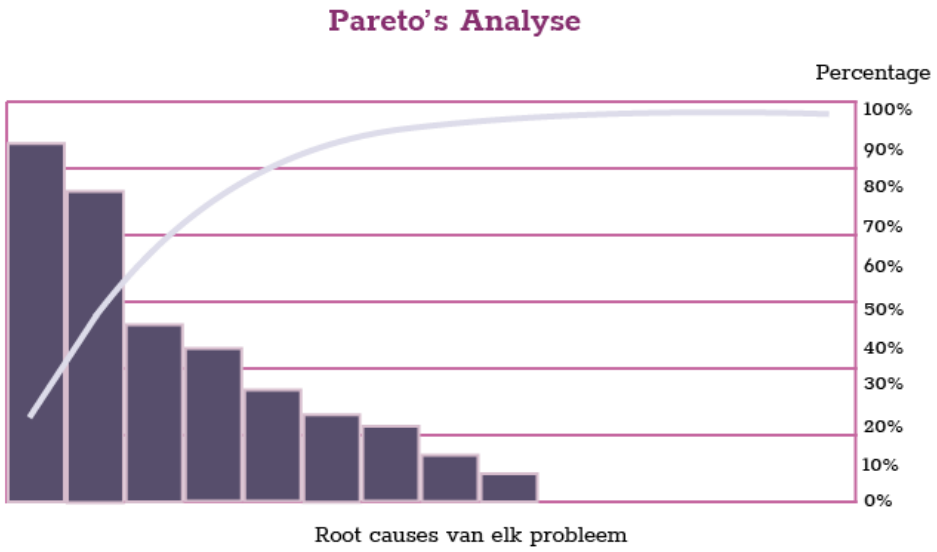
Pareto's Analyse (Voorbeeld van een risicoanalyse-methodiek)

Een van de dingen die je kunt inzetten om risicogebaseerd testdoelen te bepalen is Pareto's analyse. **[VI]** Pareto's analyse is gebaseerd op het idee dat 80% van de opbrengsten van een project bereikt kan worden door 20% van het werk te doen. Of omgekeerd, 80% van de defects kan gerelateerd worden aan 20% van de oorzaken.

Stappen om een Pareto's analyse te doen:

1. Identificeer en beschrijf problemen die zich hebben voorgedaan
2. Identificeer de root cause (onderliggende oorzaak) van elk probleem
3. Ken een relatieve score toe aan elk probleem
4. Groepeer gelijksoortige oorzaken en tel de score voor die groepen bij elkaar op
5. Je hebt nu in kaart welke soort problemen het meest voorkomen en kunt daar gericht actie op gaan nemen

Een Pareto's analyse zal tot een onderstaand soort lijndiagram leiden:



Root Cause Analyse

Root cause analyse is het opsporen van de onderliggende oorzaak van defects, incidenten of problemen. Door tijdens en direct na het incident te kijken naar de oorzaak en deze oorzaak direct vast te leggen bij het incident, kun je een krachtig instrument ontwikkelen om je software development proces te verbeteren. Onderstaand een overzicht van mogelijke root causes, die je kunt gebruiken. Door root causes vast te leggen en periodiek een rapportage hierover te maken kun je zien welke type fouten het meest voorkomen en hoe je de oorzaak daarvan weg zou kunnen nemen.

Categorie oorzaak	Hoofd oorzaak	Omschrijving
Code	Fout in code	De functionaliteit werkt niet vanwege een fout in de code.
	Anders gebouwd dan	De gebouwde functionaliteit wijkt af van de omschrijving in het ontwerp.
	Meer gebouwd dan	De gebouwde functionaliteit stond niet beschreven in het ontwerp.
	Tekst & Layout	Fouten in spelling, interpunctie, design, uitlijning en gebruiksvriendelijkheid.
	Vergeten uit te voeren	Er is vergeten een deel van de functionaliteit te ontwikkelen.
	Onbedoelde wijziging	Door gewijzigde code is onbedoeld iets gewijzigd aan de functionaliteit.
	Merge/integratie fout	Vanwege een integratie van verschillende systemen is een fout ontstaan.
Ontwerp	Ontwerp niet	Het ontwerp is niet tijdig bijgewerkt.
	Interpretatie fout	De beschrijving van de functionaliteit in het ontwerp is verkeerd geinterpreteerd door
	Ontwerp hiaten	In het ontwerp is een nodig onderdeel van de functionaliteit niet beschreven.
	Onduidelijke	De omschrijving van de functionaliteit is niet duidelijk genoeg in het ontwerp.
	Incorrecte specificatie	Er staat een fout in de specificatie.
Omgeving	Hardware	De bevinding is veroorzaakt door hardware problemen.
	Software	De bevinding is veroorzaakt door software problemen.
	Configuratie	De bevinding is veroorzaakt door verkeerde configuratie van (onderdelen van) de
Test	Incorrecte testdata	De testdata die gebruikt is voor het uitvoeren van het testgeval was niet correct.
	Incorrect testgeval	Het uitgevoerde testgeval is niet correct.
	Testfout	De testcases zijn niet juist uitgevoerd.
Onbekend		Oorzaak nog niet duidelijk.

Ongeacht hoe je (of je team) hebt gekozen je risicoanalyse en teststrategie te ontwikkelen, is het belangrijk te onthouden dat alles testen onmogelijk is. Daarom moet je er zeker van zijn dat je het grootste deel van je testinspanningen richt op de belangrijkste en meest waardevolle onderdelen van de software.

Definities

API	Application Programming Interface is een set regels en hulpmiddelen waarmee verschillende softwareprogramma's met elkaar kunnen communiceren en samenwerken
Complexiteit	Mate van ingewikkeldheid.
Schade	Het nadelige gevolg, zowel materieel als immaterieel, van een gebeurtenis.
Faalkans	De statistische kans dat iets fout gaat.
Frequentie	Hoe vaak iets gebeurt of voorkomt binnen een bepaalde tijd of in een zekere ruimte.
REST	REpresentational State Transfer is een vorm van een API waarbij computers op een eenvoudige, gestandaariseerde manier met elkaar kunnen communiceren.
Risico	Een factor opgebouwd uit de componenten faalkans en

	schade, die kan uitmonden in toekomstige negatieve gevolgen.
Risicoanalyse	Het proces van risico identificatie en beoordeling.
Root cause analyse	Een analysetechniek met als doel om de hoofdoorzaak van fouten (defects) te identificeren.
SOAP	Simple Object Access Protocol is een gestandaardiseerde methode waarop computers met elkaar kunnen communiceren.
Testbasis	De basis waarop tests worden gebaseerd, waaronder documentatie, een bestaand systeem, eisen, of de kennis van iemand die weet hoe het werkt.
Testobject	Het te testen systeem of product.
Testdoel(en)	Verschillende delen van het te testen systeem.
Teststrategie	De tactische richtlijnen die alle ins en outs uitleggen van alles wat getest moet worden tijdens de SDLC.

Practice 4: Wees Specifiek

LD12	Pas de practice van specifiek zijn toe. (K3)

Bij het vastleggen van testcases is het erg belangrijk om specifiek te zijn. Programmeren is in tegenstelling tot sommige aspecten van testen wel een exacte wetenschap. Iets is 1 of 0 zoals gebruikelijk in de informatica. Als je dus op die techniek je testcases bouwt dan moet je ook zo exact zijn. En omdat je ook zo exact dient te zijn met het maken van je testcases, moet je dat ook zijn bij het maken van requirements of ontwerpen. Deze zijn immers je test basis; de bron waarop je je testen baseert.

Specifiek zijn zorgt ervoor dat je keuzes inzichtelijk maakt. Het zorgt er ook voor dat je het over de juiste dingen hebt. Ook al zijn de requirements die je hebt gekregen misschien niet specifiek, je kunt ze zelf wel specifiek maken doormiddel van de vertaling naar testcases . Als je hierbij vragen hebt, kun je deze altijd voorleggen aan de opdrachtgever. Op die manier heb je al vrij eenvoudig de kwaliteit verbeterd.

Een truc die je hiervoor kunt gebruiken is SMART. Dit staat voor Specifiek Meetbaar Acceptabel Realistisch Tijdsgebonden. De tabel bij de definities sectie van dit hoofdstuk geeft een uitleg van de onderdelen van SMART **[VII]**.

In sommige gevallen kan het moeilijk of bijna onmogelijk zijn om ‘vage’ requirements of testgevallen verder te verduidelijken. Als dit het geval is, kun je het risico beschrijven van het requirement dat je niet goed kunt testen, of het testgeval dat je niet goed kunt beschrijven. Met andere woorden, geef precies aan wat je niet kunt testen of meten omdat je onvermogen om dat te doen het risico vergroot. Als je bijvoorbeeld moet vaststellen dat een

schermachtergrond ‘paars’ blijft wanneer een bepaalde actie plaatsvindt en je hebt geen specifieke RGB-kleurenparameters gekregen, hoe kun je dan vaststellen dat de schermachtergrond de juiste kleur paars heeft? Als je alle tinten paars zou aanvaarden (omdat je de exacte kleurparameters niet aan de klant hebt gevraagd), voldoe je misschien wel aan de oorspronkelijke eis, maar is het specifieke ervan gemist, omdat je het hele spectrum van paarse kleuren aanvaardt, en het specifieke paars is waarschijnlijk belangrijk voor de bedrijfsidentiteit van de klant.

Voorbeeld Business Requirement:

Requirement:
‘Er zullen werkstromen aangemaakt worden voor de verschillende brieven en e-mails. Deze werkstromen kunnen uitgevoerd worden op basis van selecties die gemaakt worden d.m.v. geavanceerd zoeken.’

Het probleem hierbij is dat je niet weet:

- Hoeveel werkstromen worden er aangemaakt?
- Hoeveel brieven zijn er?
- Hoeveel e-mails zijn er?
- Hoeveel selecties zijn er?
- Wat is een werkstroom?
- Welke werkstromen leiden tot welke brieven?
- Welke werkstromen leiden tot welke e-mails?
- Leiden alle werkstromen alleen tot brieven?
- Leiden alle werkstromen alleen tot e-mails?
- Waar kan ik de werkstromen opstarten?
- Hoe gaat het opstarten van de werkstromen in zijn werk?
- Wat is de relatie tussen de selecties en geavanceerd zoeken?

Door deze vragen te stellen, krijgt je een veel beter inzicht in wat het requirement vraagt, zodat je beter kunt begrijpen wat het requirement inhoudt en wat je moet testen. Dit helpt om het requirement te laten voldoen aan de SMART-normen zoals hierboven uitgelegd.

Onderstaand wat de verbetering was in dit specifieke voorbeeld.

- In de applicatie kan een aantal werkstromen aangemaakt worden. Een werkstroom is een onderdeel van het standaardpakket en kan geconfigureerd worden voor specifiek gebruik. Er zijn drie werkstromen: werkstroom A, werkstroom B en werkstroom C. Werkstroom A is voor de acceptatiebrief of e-mail (indien e-mail bekend is), werkstroom B is voor de afwijzingsbrief en werkstroom C is voor het versturen van de rekening.
- Deze werkstromen kunnen door de rol ‘Behandelaar Klantcontact’ uitgevoerd worden vanaf scherm ‘CER012’. Op dit scherm worden alle werkstromen getoond. Door middel van het drukken op de knop ‘Start werkstroom’ wordt de werkstroom geactiveerd. Er komen drie knoppen. Eén knop voor elke werkstroom. De werkstromen mogen maximaal een minuut actief zijn.

- Middels het drukken op de knop 'Start werkstromen' wordt een voorgedefinieerde selectie in geavanceerd zoeken geactiveerd. Deze selectie zijn: voor werkstroom A: NAW GEGEVENS of EMAIL en NAAM; BRIEFTEKST18; ONDERTEKENING4, werkstroom B: NAW GEGEVENS; BRIEFTEKST14; ONDERTEKENING2, werkstroom C: NAW GEGEVENS; BRIEFTEKST11; FINANCIEELE GEGEVENS; ONDERTEKENING1.
- Indien een werkstroom niet uitgevoerd kan worden of indien een werkstroom langer dan een minuut actief is, dient een foutboodschap met de volgende tekst: 'werkstroom X kon niet uitgevoerd worden' getoond te worden aan de behandelaar die de werkstroom opstartte. Hierbij moet op de positie van de X de naam van de betreffende werkstroom getoond te worden.

De bovenstaande tekst kost misschien meer tijd om te genereren, maar geeft specifieke informatie die veel gemakkelijker te begrijpen en te ontwikkelen is. Dit maakt het ook veel gemakkelijker om te weten wat er getest moet worden. En het bespaart hoogstwaarschijnlijk projectontwikkelingstijd, omdat de kans groter is dat het product precies voldoet aan de eisen van de klant.

Wees Specifiek: Wanneer je een Defect beschrijft

Bij het opstellen van een defect is het belangrijk zo specifiek mogelijk te zijn, zodat de defects gereproduceerd of opnieuw getest kunnen worden door andere mensen.

Een goed gespecificeerd defectsrapport moet bevatten:

- een duidelijke titel
- een SMART beschrijving van de stappen om het defect te reproduceren
- verwacht resultaat
- daadwerkelijk resultaat
- bewijs van het defect: logging, screenshots, enz.
- relevante informatie: een uniek nummer dat de bijbehorende testcase(s) identificeert, de naam van het scherm, traceerbaarheid (of verwijzing) naar de gegeven eis, in welke omgeving het defect zich voordoet, het versienummer van de gebruikte software
- ernst van het defect: 'Blokkerend', 'Ernstig', 'Niet ernstig', 'Cosmetisch'
- prioriteit van het defect: 'Laag', 'Gemiddeld', 'Hoog'

Aangezien er meestal meerdere partijen betrokken zijn bij het maken en testen van nieuwe software gedurende de SDLC, is het belangrijk om goede communicatiestandaarden te hebben, zoals specifiek zijn. Dit helpt de teamleden waardevolle tijd (en middelen) te besparen, en taken sneller uit te voeren, terwijl het ook een postieve invloed op de motivatie tijdens het hele project heeft.

Definities

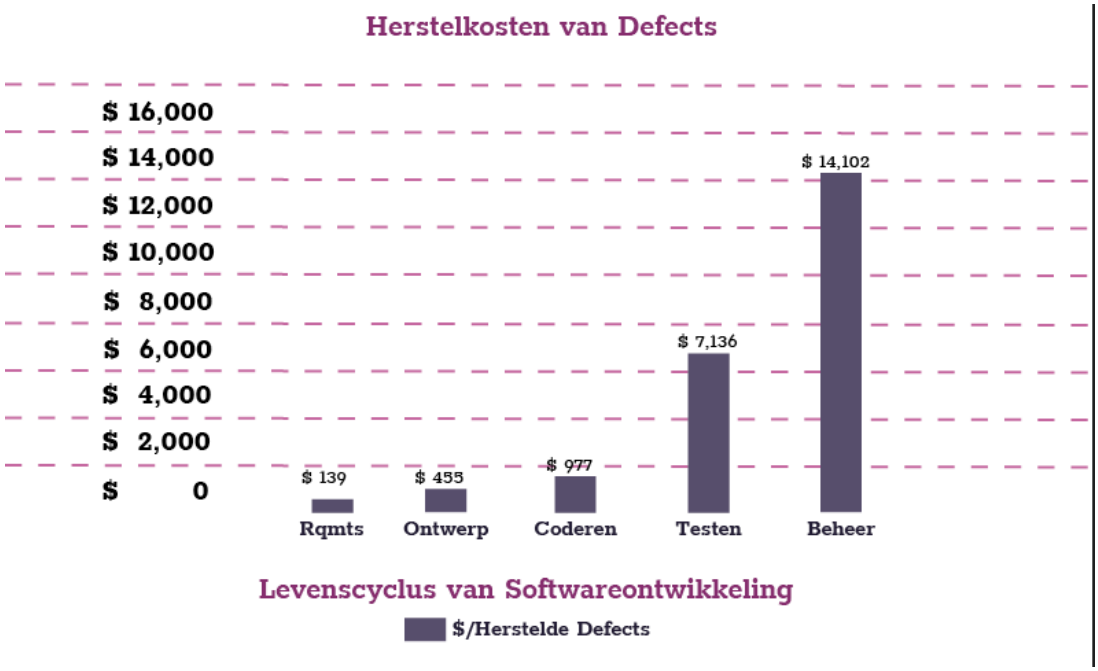
Specifiek	Wees zo gedetailleerd mogelijk. Vermijd algemeenheden. Vermijd

	verwijswoorden als deze, dit, die, dat. Omdat als een tekst gekopieerd of aangepast wordt het niet duidelijk kan zijn wat er bedoeld wordt. Formuleer positief en enkelvoudig. Denk ook om het formuleren van fout paden.
Meetbaar	Het requirement of de testcase moet te meten zijn. Vermijd dus vage bepalingen als ‘snel’, ‘direct’, ‘veel’, ‘gemakkelijk’, of ‘gelijk’. Maar koppel er een te meten eenheid aan. Op die manier kun je echt aangeven of het klopt. Bijvoorbeeld: ‘De data in scherm CER012 moet na het openen van het scherm of het opvragen van een record, binnen 1 seconde volledig geladen zijn.’ of ‘het proces moet dagelijks circa 100.000 transacties kunnen verwerken’.
Acceptabel	Een requirement of testcase moet altijd geaccepteerd worden door de opdrachtgever en/of andere stakeholders. Ook moet iets ethisch en moreel acceptabel zijn en voldoen aan wettelijk vereisten of voorschriften.
Realistisch	Een requirement of testcase moet altijd reëel zijn. Het moet realistisch geformuleerd zijn. Een haalbaar doel met de beschikbare tijd, financiën en resources.
Tijdsgebonden	In veel requirements ontbreekt een bepaling van tijd. Door deze wel op te nemen en concreet te maken, kun je beter meten en dus beter testen of een requirement voldoet. Neem dus een afgebakende tijdseenheid op, bijvoorbeeld seconden, minuten of dagen.

Practice 5: Test zo Vroeg Mogelijk

LD18	Begrijp de practice van zo vroeg mogelijk testen. (K2)
LD19	Soorten omgevingen waarop je kunt testen. (K2)
LD20	Begrijp het belang van een goede testomgeving. (K2)
LD21	Begrijp het belang van testdata. (K2)

De practice om zo vroeg mogelijk te testen levert aanzienlijke tijd- en kostenvoordelen op. Deze praktijk vindt zijn oorsprong in onderzoek van Barry Boehm uit 1977 **[IX]**. Uit dit onderzoek blijkt namelijk dat de kosten van een defect exponentieel toenemen naarmate een defect later gevonden wordt. Ergo, hoe eerder het defect gevonden wordt in de SDLC hoe goedkoper het is om dit op te lossen. Dit wordt geïllustreerd in onderstaande grafiek.



Figuur: Herstelkosten van defects

Deze wetenschap zorgt ervoor dat je zo vroeg mogelijk defects wilt gaan voorkomen. Er zijn een aantal manieren waarop je dit kunt doen. Kun je het de software nog niet uitvoeren (dit noemen we dynamisch testen), dan kun je wellicht een review toepassen op requirements of code.(statisch testen) dit kun je alleen doen of met meerdere personen in een meeting, ook kun je als ontwikkelaar bijvoorbeeld een walkthrough doen met gebruikers. Dit zodat zij je werk vroegtijdig kunnen beoordelen en eventueele aanpassingen kunnen benoemen. of vroeg een demo van je product geven.

Het belang van reviews en zo vroeg mogelijk testen

Het is belangrijk om statisch zoveel je kunt te testen. Wacht hier niet mee. Reviewen van requirements is namelijk zeer kosteneffectief (hoewel het misschien leuker is om direct te beginnen met ontwikkelen). Je kunt er hier immers al voor zorgen dat de requirements correct en volledig zijn voordat je gaat ontwikkelen. Het voorkomt dus het bouwen van requirements die niet correct of onvolledig waren.

Verder is een goede en complete unittest of een code review door een collega op ontwikkelomgeving i volgens het onderzoek van Boehm goedkoper en sneller dan pas iets testen op een latere omgeving. Je zorgt er namelijk voor dat iets met minder defects opgeleverd wordt en bespaart zodoende tijd in het deployen naar een volgende omgeving.

Ook nieuwer onderzoek dan dat van Boehm toont dit aan. Wees je hier dus als ontwikkelaar of beheerder bewust van. Andere middelen die je hier zou kunnen toepassen is het laten reviewen van requirements door gebruikers, het uitvoeren van een geautomatiseerd statische review op de code. Ook unitintegratie testen met collega ontwikkelaars kunnen bijdragen aan dit principe. Als voorbeeld wil ik hier noemen de integratie tussen Front- en Backend van een webapplicatie. Die vaak al op de ontwikkelomgeving gedaan kan worden, maar in de praktijk pas later plaats vindt.

Een ander belangrijk punt dat hieraan te koppelen is, is in mijn ogen het meest onderschatte onderdeel van het vakgebied testen, namelijk het zorgdragen voor een goed functionerende en beheerde testomgeving. Dit bespreken we in de volgende paragraaf.

Testomgevingen

Testomgevingen zijn omgevingen waarin getest wordt. Meestal zijn er vier verschillende soorten testomgeving te onderscheiden. De omgeving die je altijd als eerste tegenkomt is de ontwikkelomgeving.

De **ontwikkelomgeving** is het eerste niveau en wordt gebruikt door de ontwikkelaars om de software te ontwikkelen. Op de ontwikkelomgeving voer je veelal technische testen als unittesten uit. Het ontwikkelen en het testen op deze omgeving wordt normaal gesproken door de ontwikkelaars gedaan en vaak is er geen of weinig integratie met externe software componenten of partijen.

Het tweede niveau is gewoonlijk de **testomgeving**. De testomgeving is bedoeld voor technische testen zoals systeemtesten en integratietesten. Hiervoor heeft de omgeving meer externe componenten nodig, zodat de integratie kan plaatsvinden. Deze omgeving wordt voornamelijk gebruikt door testers, maar ontwikkelaars en beheerders kunnen ook toegang hebben.

Het derde niveau is meestal de **acceptatieomgeving**. De acceptatieomgeving is een relatief complete omgeving waar bijna alle relevante componenten uit het productielandschap aanwezig zijn. Het hoofddoel van deze omgeving is het accepteren van de nieuwgebouwde of

gewijzigde software. Dit houdt dus in dat testen hier voornamelijk uitgevoerd worden door eindgebruikers. Omdat het vaak de meest complete omgeving is en de laatste omgeving voor de productieomgeving, wordt de omgeving ook wel eens door beheerders gebruikt om installatietesten of andere beheergerelateerde testen uit te voeren.

Het laatste niveau is de **productieomgeving**. Het is niet aan te bevelen om hier te gaan testen, maar er zijn wellicht een aantal uitzonderingen waarbij je er niet onder uit kan. Bijvoorbeeld als je een compleet nieuw systeem aan het bouwen bent waar je een performancetest op uit wilt voeren. Bij performancetesten is het lastige dat je technische gelijkheid aan productie moet hebben om een goede meting te kunnen doen. De berekening van de belasting is gerelateerd aan de aantallen en hoeveelheden van de technische componenten gebruikt in de omgeving (hardware, software, switches, CPU, memory). De omgeving waar deze componenten in de juiste verhouding aanwezig zijn is de productieomgeving. Omdat technische gelijkheid van cruciaal belang is bij performancetesten is het vaak moeilijk om dit met dezelfde precisie in andere omgevingen te doen. Als het gaat om testen in de productieomgeving, is het heel belangrijk om uiterst voorzichtig te zijn met de risico's die dat oplevert. Het is altijd aan te raden om alle tests in de omgevingen vóór de productieomgeving uit te voeren.

De vier omgevingen (Ontwikkeling, Test, Acceptatie en Productie) worden vaak het OTAP-principe genoemd. Het **OTAP-principe** houdt in dat software wordt gemaakt en vervolgens in elke omgeving in volgorde wordt uitgerold (Ontwikkeling -> Test -> Acceptatie -> Productie), waarbij de software pas in productie gaat als hij in elke vorige omgeving is geweest. Door dit principe te volgen, kunnen we er zeker van zijn dat alle softwareversies in de gegeven omgevingen hetzelfde zijn (met uitzondering van wijzigingen die worden ontwikkeld of getest).

OTAP in Agile en DevOps

Bovenstaande alinea is beschreven als een situatie waarin veel organisaties met een initieel volwassenheidsniveau op IT en/of testgebied naar toe zouden kunnen groeien. Het vergroot je mogelijkheden om veel en volledig te kunnen testen vroeg in het softwareontwikkeltraject. Dat wil echter niet zeggen dat het OTAP model een verplichting is. Als er voldoende mogelijkheden zijn om de testlevels die je definieert in je teststrategie uit te voeren op minder omgevingen, zoals je wel ziet in Agile teams, dan kan dat ook een prima oplossing bieden.

Er zijn ook organisaties die een DevOps werkwijze hanteren. DevOps is een werkwijze waarbij Development en Operations binnen één team wordt gedaan. Ook gebruikers zijn vertegenwoordigt in de teams. Hierdoor zijn er soms situaties waarbij de OTAP niet meer volledig ingezet wordt. Denk bijvoorbeeld ook aan App ontwikkeling waar nieuwe features voor een beperkt aantal gebruikers op productie uitgerold worden en de features in geval van bevindingen eenvoudig teruggedraaid kan worden. Het is hierdoor niet altijd noodzakelijk een volledige OTAP straat te hebben. De keuze voor een OTAP straat zal ook altijd een goede afweging tussen kosten, baten en risico zijn.

Begrijp het Belang van een Goede Testomgeving

Één of meer goed ingerichte en goed beheerde testomgevingen kunnen ervoor zorgen dat de kwaliteit van je software verbetert, door het mogelijk te maken snel, accuraat en effectief te testen. Met ander woorden, je kan veel aannames wegnemen die je anders pas op de productieomgeving zou tegenkomen.

Een testomgeving is belangrijk omdat er verschillende praktijken samenkomen. In de testomgeving worden eisen, uitgewerkte code, testgevallen, testgegevens en andere noodzakelijke software samengevoegd. Als de testomgeving niet bestaat, of als onderdelen ontbreken, kan de software niet worden getest en moeten er dus aannames worden gedaan (bijvoorbeeld dat sommige onderdelen werken). Dit is in strijd met 'Practice 1: Doe geen aannames'.

Om zo vroeg mogelijk te testen, wil je ervoor zorgen dat je testomgeving zo compleet mogelijk is, om te voorkomen dat je in een laat stadium moet testen in een omgeving van een hoger niveau. Omgevingen op een hoger niveau zijn uitgebreider in de SDLC. Daarom hebben deze omgevingen op een hoger niveau vaak meer afhankelijkheden, wat hun beschikbaarheid (qua tijd) voor testen in een later stadium ernstig kan beperken. De beperkte beschikbaarheid van omgevingen op een hoger niveau kan het gevolg zijn van vele factoren, zoals interne projectafhankelijkheden of gebruik door andere projecten. Zelfs het feit dat het tijdschema dichter bij de release krapper wordt, kan een belangrijke factor zijn.

Er zijn soms begrijpelijke financiële redenen om geen aparte testomgeving te creëren. Het is echter belangrijk in gedachten te houden dat alle defects die niet of later worden gevonden ook geld kosten. Sommige van deze kosten zijn misschien niet onmiddellijk zichtbaar (in verband met extra fixes en roll-outs), maar kunnen veel later komen wanneer de software in productie is en reputaties, verkoop en algemene kwaliteit zijn geschaad. Wanneer de risico's van een ontbrekende testomgeving duidelijk worden gemaakt, zijn veel klanten bereid meer te investeren, zelfs zonder een volledig sluitende business case. Het is sterk aan te raden de risico's zo concreet mogelijk te maken en deze urgentie goed te beschrijven.

Stubs en Drivers gebruiken

Om het probleem op te lossen dat niet alle componenten beschikbaar zijn in omgevingen op een lager niveau, kunnen we **stubs en drivers** gebruiken om die componenten te simuleren. Door simulatie van de ontbrekende systeemonderdelen kun je in een vroeg stadium uitgebreider testen en dus eerder in de SDLC defects vinden. Stubs en drivers zijn, afhankelijk van de situatie en de technologie, relatief eenvoudig te maken en kunnen daarom relevant zijn voor de business case bij het beoordelen van de besparingen op de correctiekosten van defects in een later stadium.

Enkele overwegingen ten aanzien van testomgevingen:

- Het ontbreken van een testomgeving leidt altijd tot een risico. Ook als slechts een deel van de testomgeving ontbreekt, kunnen risico's ontstaan, omdat je delen niet kunt testen.
- Risico's moeten duidelijk worden gemaakt aan de eindverantwoordelijke (de opdrachtgever).
- Testomgevingen moeten zo goed mogelijk lijken op de productieomgeving.
- Vergeet niet om stubs en drivers te gebruiken in plaats van ontbrekende componenten.
- Respecteer het OTAP-principe, en dwing het waar mogelijk (technisch) af.
- Zorg voor configuratiebeheer en onderhoud de omgevingen op de juiste wijze. Het is ook belangrijk om configuratiebeheer van relevante componenten uit te voeren.
- Automatiseer waar mogelijk altijd de overdracht van software van de ene naar de andere omgeving. Dit kan onnodige handmatige handelingen voorkomen die de kwaliteit en voorspelbaarheid van de implementaties kunnen schaden.
- Het kan een voordeel zijn om het aanmaken van omgevingen te automatiseren. Op die manier hoef je infrastructuurcomponenten maar één keer te beheren, waardoor je gemakkelijk nieuwe omgevingen kunt aanmaken met specifieke wijzigingen of releases die gemakkelijk kunnen worden weggegooid. Dit is vooral aan te bevelen wanneer je met cloudomgevingen werkt.

Als het gaat om testomgevingen, zijn er eindeloos veel problemen en risicovolle situaties die je kan tegenkomen. We kunnen ons een testomgeving voorstellen als een laboratoriumopstelling: een product (Software) wordt gemaakt in een zeer nauwkeurig proces (SDLC). Men moet beschikken over de juiste hoeveelheid originele stoffen die op de juiste wijze moeten worden gebruikt: ze moeten op precies de juiste temperatuur worden verhit, verdampt, enz. In het proces is het niet mogelijk om een deel van de oorspronkelijke grondstof eruit te halen omdat dit het eindproduct zou beïnvloeden.

Testdata

De kwaliteit van de beschikbare testgegevens bepaalt in grote mate de kwaliteit van je tests, dus het is van essentieel belang om over testgegevens van goede kwaliteit te beschikken. Waar mogelijk is het gunstig om productiegegevens te gebruiken. Als je echter met productiegegevens werkt, moet je rekening houden met lokale wetgeving, zoals de GDPR in Europa.

Vanuit een testperspectief is testen met productiedata gunstig omdat het veel variaties van functionele testcases oplevert. Het gebruik van een grotere verscheidenheid aan gegevens kan een grondige en efficiënte manier zijn om de software te testen. Of productiegegevens nu mogelijk zijn of niet, het is belangrijk ervoor te zorgen dat alle functionele varianten aanwezig zijn in de testgegevens, bijvoorbeeld als je de software van een autoverzekering test, zijn gegevens met alle variaties in adres, type auto, schadevrije jaren, enz. essentieel.

Om effectief en efficiënt te zijn, is het belangrijk om zo vroeg mogelijk te testen, zodat we defects eerder tegenkomen en voorkomen dat ze later in de SDLC opduiken en duurdere defects worden.

Definities

OTAP Principe	Het is een principe waarbij de software alle omgevingen doorloopt alvorens in gebruik genomen te worden. OTAP staat voor Ontwikkel, Test, Acceptatie, Productie.
Review	Een vorm van statisch testen waarbij een (tussen)product uit de SDLC door een of meer personen wordt beoordeeld.
Stubs en drivers	Een tijdelijk gesimuleerde component dat gebruikt wordt om een andere softwarecomponent te testen. Een stub komt na een component, een driver ervoor.
Testdata	De gegevens die gebruikt wordt om testen uit te voeren.
Testomgeving	Een gevalideerde, bruikbare en stabiele omgeving, zoveel mogelijk lijkend op de uiteindelijke productieomgeving, die gebruikt wordt om testcases uit te voeren of om bugs te reproduceren.

Practice 6: Begin Klein en maak je Testen steeds Groter en Groter

LD22	Begrijp de practice van klein beginnen en je testen steeds uitbreiden. (K2)
LD23	Onthoud testlevels and testtypes. (K1)
LD24	Begrijp hoe testlevels en testtypes zich verhouden tot een teststrategie. (K2)

Het concept 'klein beginnen en geleidelijk uitbreiden' wordt ook beschouwd als een best practice voor projectmanagement. Dit geldt ook als we kijken naar de voordeligste manier om software te testen. De praktijk gaat ervan uit dat je een klein deel (unit) van de software volledig hebt gecontroleerd om er zeker van te zijn dat het goed werkt. Eenheden waarvan is bevestigd dat ze goed werken, kunnen worden geïntegreerd met een andere eenheid van de software, op voorwaarde dat de andere eenheid al onafhankelijk is gecontroleerd, waardoor je bereik groter wordt. Dit heeft als voordeel dat defects die voortkomen uit de integratie van de twee eenheden het gevolg moeten zijn van de integratie zelf. Dit vereenvoudigt het onderzoek bij het zoeken naar de oorzaak van gevonden gebreken. Deze praktijk versterkt de 'Practice 5: Test zo vroeg mogelijk', omdat je eerder en vollediger kunt testen als je het systeem opdeelt in kleinere te testen eenheden.

In het algemeen zijn er vier testlevels:

- Testlevel 1 - unittesten
- Testlevel 2 - integratietesten
- Testlevel 3 - systeemtesten
- Testlevel 4 - acceptatietesten

Testlevel 1 - Unittesten

De praktijk van het testen per eenheid heet **unittesten**, en is volgens internationale teststandaarden (zoals de ISTQB) het eerste **testlevel**. Het is gebruikelijk dat unit testing wordt uitgevoerd door ontwikkelaars in de ontwikkelomgeving. Unit testing wordt dynamisch uitgevoerd en gebeurt tegen de voltooiing van de ontwikkeling van de unit, terwijl de software nog in ontwikkeling is. De reikwijdte en dekking van unit testen zijn dus altijd beperkt tot de unit zelf.

Testlevel 2 - Integratie Testen

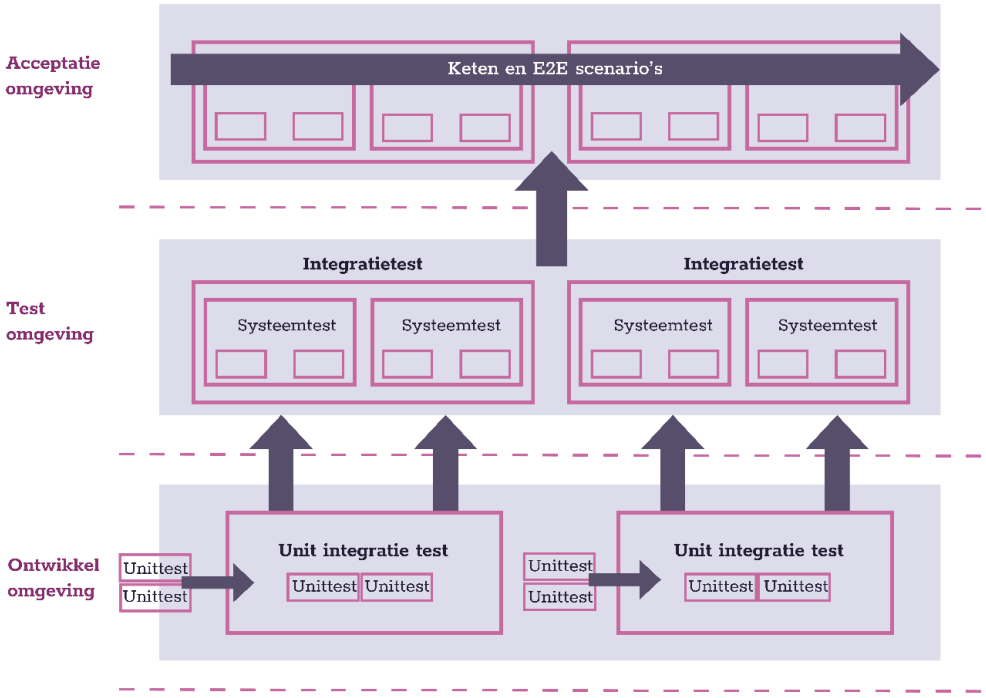
Het volgende testlevel betreft het testen van twee units samen, dat heet integratietesten. Deze tests zouden alleen nog defects moeten genereren die te maken hebben met de integratie van de twee units, aangezien defects in beide units afzonderlijk gevonden hadden moeten worden in de voorgaande unittesten. Door dit proces te volgen wordt het opsporen van fouten aanzienlijk vergemakkelijkt. Opmerking: waar we de term 'unit' gebruiken, kunnen we ook 'component' gebruiken. Aangezien integratietesten zich voornamelijk richten op de interactie tussen twee componenten of systemen, moet de scope alle mogelijke interacties (en situaties daarvan) tussen de twee units omvatten.

Testlevel 3 – Systeem Testen

Systeemtesten is het testniveau dat over het algemeen wordt aanbevolen na de integratietesten. Systeemtesten worden meestal uitgevoerd in de testomgeving en omvatten het testen van verschillende integraties samen. Systeemtesten kunnen soms een keten van processen omvatten, wat **ketentesten** wordt genoemd. De scope van systeemtesten is gericht op de flow en/of consistentie van verschillende integraties en eenheden van het systeem.

Testlevel 4 – Acceptatie Testen

Het volgende level dat je over het algemeen aantreft, wordt **acceptatietesten** genoemd. Acceptatietesten worden meestal uitgevoerd door gebruikers uit de organisatie waarvoor het systeem is ontwikkeld. De scope van acceptatietesten is om te bevestigen of het systeem de gebruikers en organisatorische processen ondersteunt zoals bedoeld. **End-to-End-testen** (E2E-testen) is een vorm van testen die (de belangrijkste scenario's van) de stroom van een systeem van het begin tot het einde test. E2E-testen worden meestal uitgevoerd op het acceptatieniveau van de SDLC, waar veel afhankelijkheden bestaan.



Figuur: testlevels vs. testomgeving

In de figuur 'Testlevels vs. testomgeving', staat elk vakje (unit) voor een test. Als het testen van een box is afgerond, dan voeg je de geteste box samen met een andere (geteste) box. Dit gaat verder volgens de testniveaus (1-4) zoals eerder uitgelegd die op elkaar voortbouwen.

Systeem Integratie Testen

Een veelgebruikt testlevel dat tussen systeem- en integratietesten in zit, wordt **systeemintegratietesten** (SIT) genoemd. Software die integratie vereist met de software van andere teams of organisaties vraagt speciale aandacht. Wanneer softwarecomponenten afzonderlijk worden ontwikkeld, is de kans groter dat er misverstanden ontstaan of dat miscommunicatie optreedt. Deze communicatiestoringen kunnen ertoe leiden dat delen van de software anders worden geïnterpreteerd en dat kan de algehele kwaliteit van de software schaden.

End-to-End-testen

We hebben al gezien hoe klein beginnen en je testscope geleidelijk uitbreiden het opsporen van fouten vergemakkelijkt, maar het principe van vroeg testen is hier ook van toepassing. Een E2E-test is een late en daarom relatief dure vorm van testen, dus deze moet zo soepel en efficiënt mogelijk worden uitgevoerd. E2E-testen moeten alleen fouten vinden die voortkomen uit de integratie van alle procesonderdelen. Daarom is het in je eigen belang om de eerdere testtypes volledig uitgevoerd te hebben voordat je begint met E2E-testen.

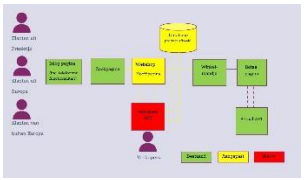
Test Types

Testtypes zijn gericht op specifieke testdoelen en/of kenmerken van een component of systeem **[V]**. Ze geven meestal bepaalde kwaliteitsattributen weer (functionaliteit, beveiliging, prestaties, bruikbaarheid, etc.), maar kunnen zich ook richten op aspecten als regressie of wijzigingen. Sommige van deze testtypes zullen worden uitgewerkt in het volgende hoofdstuk dat zich richt op kwaliteitsattributen.

De verschillende testtypes kunnen op elk van de vier testniveaus worden uitgevoerd, maar het is nuttig om te bedenken dat bepaalde testtypes in bepaalde omgevingen gunstiger zijn. Testtypes worden meestal bepaald na de risicoanalyse of de analyse van de testbasis wanneer duidelijk wordt dat bepaalde aspecten van het systeem speciale aandacht nodig hebben.

Implementeren van Test Levels en Test Types in je Test Strategie

Als we doorgaan met ons voorbeeld uit oefening 3 en onze teststrategie opstellen, dienen we de verschillende onderdelen van het systeem (testdoelen) te analyseren, zodat we kunnen beslissen welke testlevels we op elk testdoel willen toepassen. Als je kijkt naar de vorige voorbeeldschets, zijn je testdoelen de verschillende onderdelen: ‘Klanten’, ‘Inloggen’, ‘Zoeken’, ‘Webshop’, ... etc.



De schets uit practice 3: Alles testen is onmogelijk

Om de verschillende testlevels en testtypes toe te passen op onze testdoelen, is het makkelijk om een tabel te maken, waarin je de informatie die je hebt verkregen in de risicoanalyse verwerken. Dit is de informatie over de dekking voor elk testdoel voor elk testlevel. Het is belangrijk om te overwegen welke testtypes relevant zijn bij het testen van je software op verschillende niveaus.

Op basis van ons voorbeeld zou de teststrategie er zo uit kunnen zien:

Teststrategietabel		**Testlevels**			
Testdoel/test type	**Statische test**	**Unittest**	**Integratie-test**	**Systeem-test**	**Acceptatie-test**
Orders uit Frankrijk	Code review	Hoge dekking	Hoge dekking	Gemiddelde dekking	Meenemen in alle E2E scenario's
Orders uit andere delen van Europa		Gemiddelde dekking	Gemiddelde dekking	Lage dekking	Belangrijkste scenario's testen
Orders uit andere delen van de wereld		Lage dekking	Lage dekking	Lage dekking	Enkele scenario's testen
Inloggen	..	..	..	..	..
Zoekfunctie	..	..	..	..	..
Usability	Prototyping	Walkthrough		Eye tracking	Laten testen m.b.v. gebruiks-groepen
API Documentation	Review				

Dekking

In de teststrategietabel hebben we nu de dekking bepaald voor verschillende processen. Deze dekking is hier verdeeld in 'Hoog', 'Gemiddeld' en 'Laag'. Door middel van het gebruik van verschillende testtechnieken kunnen we deze gewenste dekking gaan bereiken. In de 'Practice 7: Leg je Testen Vast' zullen we hier aandacht aan besteden.

Teststrategieën zijn een zeer uitgebreid onderwerp en je zult merken dat je kennis erover zal groeien tijdens je testcarrière. Als je gefocust blijft en 'klein begint en maak je testen steeds groter en groter', dan ben je al op de goede weg naar het creëren van een goede gestructureerde teststrategie die je kunt gebruiken in je dagelijkse werk.

Definities

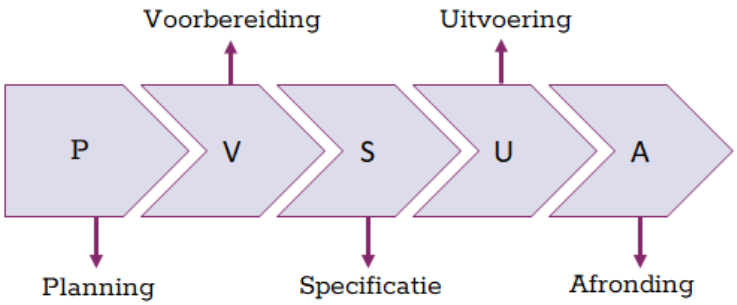
Acceptatietest	Een testsoort die erop gericht is te bepalen of het systeem wordt

	geaccepteerd. **[V]**
Ketentest	Een testtype dat meerdere systemen omvat.
End-to-End-test	Een type test waarbij bedrijfsprocessen van begin tot eind worden getest onder productie-achtige omstandigheden. **[XXI]**
Integratietest	Een testsoort die zich richt op interacties tussen systemen. **[V]**
Systeemtest	Een testsoort die erop is gericht te controleren of een systeem als geheel voldoet aan de gespecificeerde eisen. **[V]**
Testlevel	Een specifieke representatie van een testproces. **[V]**
Testtype	Een verzameling testactiviteiten op basis van specifieke test doelstellingen en gericht op specifieke kenmerken van een component of systeem, zoals een performance- of securitytest of een gebruikers acceptatietest. **[V]**
Unit-/componenttest	Een testsoort die zich richt op afzonderlijke hardware- of softwarecomponenten. **[V]**

Practice 7: Leg je Testen Vast

LD25	Onthoud de basis van een testproces. (K1)
LD26	Begrijp het nut van vastleggen van je testen. (K2)

De bestaande erkende testmethodieken onderscheiden een aantal verschillende fases in het testproces. Deze fases zijn: Planning, Voorbereiding, Specificatie, Uitvoering en Afronding.



Testing Process

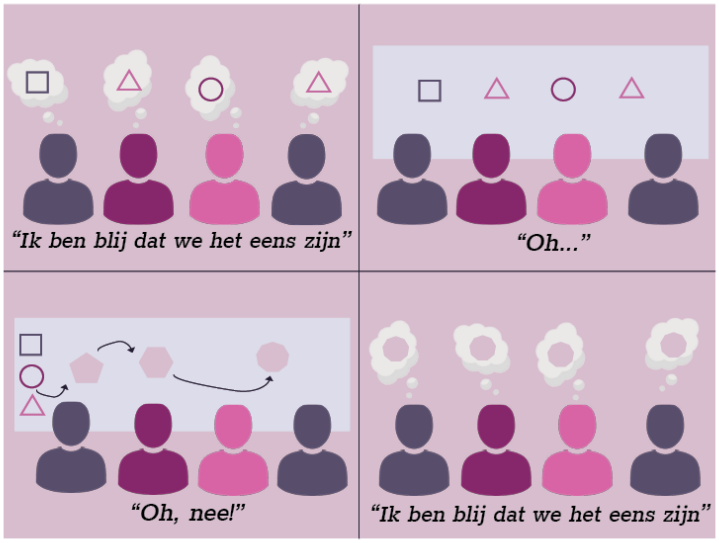
Van alle tijd die je aan testen kwijt bent, is typisch gezien zo'n 60-70% van de tijd voorbereiding **[II]**. De uitvoering (en afronding) van de testen zelf bedraagt vaak maar 10% tot 20% van de tijd. Deze percentages geven al aan dat er dus veel tijd in voorbereiding en specificatie gaat zitten.

Voorbereiding van je testen is dan ook erg belangrijk. Het belang zit erin dat je de juiste zaken test en dat je deze zaken op het juiste moment test. Dit noemen we een test strategie. Je test strategie verdient dus enige aandacht voordat je start met testen uitvoeren.

Soms is het nodig dat je een (uitgebreid) testplan moet maken, met een gedetailleerde teststrategie, waarin tot in detail afspraken en keuzes vastgelegd worden met betrekking tot het gehele test project. Je kunt hier best ver in gaan met een overkoepelend testplan voor alle testsoorten en vervolgens een detail testplan per testsoort, en afspraken over tijd, resources, planningen, budget et cetera. Uitgebreide plannen kunnen waardevol zijn in bepaalde omstandigheden, vooral in omgevingen zoals grote bedrijven of overheidsinstellingen. Uitgebreide planning kan ook de moeite waard zijn als je veel verschillende testpartijen hebt en je wilt afspreken welke partij welke items test om overlap of dubbele tests te voorkomen, of je wilt voorkomen dat meerdere partijen een bepaald gebied testen.

Houd er echter rekening mee dat zulke uitgebreide plannen niet gemaakt moeten worden uit gewoonte of omdat dat de manier van werken is in een bepaalde organisatie: dit soort planning moet alleen gedaan worden als het waarde toevoegt. Het moet nooit de prioriteit zijn om een te uitgebreid testplan te maken alleen maar om het te hebben, omdat het testplan dan waarschijnlijk niet gelezen zal worden.

Onder andere omstandigheden is een minder uitgebreide teststrategie van slechts een paar pagina's voldoende. Deze kortere overzichten moeten je gedachten en discussies met collega's documenteren, met als doel dat ze je in staat stellen om gemakkelijker te communiceren of ernaar te verwijzen. Dit soort documentatie helpt ook om verkeerde interpretaties van de afspraken te voorkomen.



Vandaar het uitgangspunt: 'Leg je testen vast'. Dit bedoelen dit op twee manieren: allereerst je test strategie vastleggen en ten tweede vastleggen wat je gaat testen of getest hebt. Om testen vast te kunnen leggen is het zeer wenselijk dat je vastgelegde requirements hebt. Dit is zeker niet altijd het geval is. Uiteraard is het vastleggen van requirements een uitstekende kwaliteitsmaatregel. Door ze vast te leggen maak je het overdraagbaar, creëer je een gemeenschappelijk begrip van de requirements en kun je ze aanscherpen en gebruiken als testbasis. Probeer dus altijd te zorgen dat opdrachtgevers requirements gaan vastleggen, help ze hierbij. Met deze maatregel kun je al zoveel kwaliteit winnen.

Een goede manier om requirements te formuleren is om er user stories van te maken in het formaat Als <rol> wil ik <functionaliteit> zodat ik <toegevoegde waarde>. Bijvoorbeeld:

*'**ALS** een admin **WIL IK** een .PDF file kunnen toevoegenaan een website*
***ZODAT** ik deze .PDF kan tonen aan de bezoekers van de website'* **[X]**

Door requirements op deze manier te formuleren, zal de praktijk van 'documenteer je tests', samen met de praktijk van 'wees specifiek', je in staat stellen om begrijpelijke requirements te maken. Deze requirements vormen vervolgens de richtlijnen voor het ontwikkelen, testen en onderhouden van software.

De mate van detail waarin requirements worden gedocumenteerd zal variëren in verschillende organisaties en projecten. De gedetailleerdheid van de requirements bepaalt hoe rigoureus en gedetailleerd de tests kunnen zijn. Dit is iets waar rekening mee moet worden gehouden bij het opstellen van een teststrategie en het kiezen van testtechnieken om bepaalde delen van het systeem te testen.

Het documenteren van tests, die zijn afgeleid van de testbasis, biedt een gestructureerde aanpak, die ervoor kan zorgen dat we geen relevante testgevallen vergeten. Precies documenteren wat je hebt getest maakt testen reproduceerbaar en herhaalbaar. Herhaalbaarheid is een onderdeel van testen dat erg belangrijk is. Vaak moet iemand anders (bijvoorbeeld de ontwikkelaar of de testcoördinator) jouw test herhalen om een bepaald defect te reproduceren. Door precies te documenteren wat je hebt gedaan, maak je het makkelijker en sneller voor hen om het systeem te analyseren, wat tijd en geld kan besparen.

Door te documenteren wat je hebt getest, maak je het ook makkelijker om die tests zelf te herhalen. Onthouden wat je hebt gedaan is soms vrijwel onmogelijk gezien de complexiteit van de situatie of het aantal testgevallen en testrondes dat je hebt gedaan. Schermopnames maken terwijl je je tests uitvoert is ook een goede manier om de testuitvoering en resultaten vast te leggen, op voorwaarde dat het een duidelijke en relevante opname is en dat je de verschillende uitgevoerde testgevallen gemakkelijk kunt opzoeken en terugvinden.

Er is nog een andere belangrijke reden om je tests te documenteren. Vaak wil je als voorbereiding op het testen een duidelijk script maken dat gebruikt wordt tijdens de testuitvoering. De testuitvoering kan bijvoorbeeld pas beginnen als de software is opgeleverd

aan de testomgeving. Dit kan dagen of weken na het maken van de testgevallen zijn. Dus, door het testscript eerder te maken, vóór de uitvoering, kunnen de tests onafhankelijk worden uitgevoerd door teamleden (of iemand anders) anders dan de auteur.
Een laatste reden om te documenteren is de volgende: om een precieze en accurate uitkomstvoorspelling te hebben. Als we onze voorspelling vooraf duidelijk opschrijven, wordt de voorspelling niet beïnvloed door de werkelijke uitkomst van de test en is het waarschijnlijker dat afwijkingen van de voorspelling worden opgemerkt. Door niet van te voren de voorspelling op te schrijven, is de kans groter dat je het functioneren van het product tijdens het testen in het script zet. Het gedrag van het product tijdens het testen is echter niet noodzakelijk het juiste gedrag. Als het verwachte gedrag vooraf is vastgesteld, is de kans kleiner dat ongewenst gedrag wordt opgenomen in het testscript en zullen defects eerder worden gedetecteerd.

Denk bijvoorbeeld aan het in elkaar zetten van een IKEA-meubel. Als je zorgvuldig de instructies volgt, krijg je het juiste resultaat (een correct in elkaar gezet meubelstuk). Als je gewoon begint met het in elkaar zetten van onderdelen zonder de instructies te volgen, is de kans groot dat je sommige items ondersteboven of binnenstebuiten bevestigt.

Gebruik van Testtechnieken

Testtechnieken gebruik je om een testontwerp te maken en een testscript op te stellen. Er zijn meerdere testtechnieken, die van toepassing zijn bij het testen van verschillende situaties, testsoorten en softwarecomponenten. Een aantal algemene zaken die je altijd kunt toepassen bij het het toepassen van een testtechniek en het uitwerken van een testscript zijn:

- Pas de testpractices toe.
- Ontwerp je testcases voordat je ze uit gaat voeren, niet tijdens de testuitvoering.
- Gebruik een techniek om een high-level testontwerp te creëren.
- Werk de high-level testcases verder uit met meer detail.
- Formuleer een testcase altijd enkelvoudig.
- Maak een scheiding tussen logische en fysieke testcases.
- Denk om uitgangssituatie (pre-condities).
- Leg een uitvoervoorspelling vast.
- Leg het resultaat vast.

Bovengenoemde punten worden verder uitgelegd in de volgende paragrafen, die focussen op verschillende testtechnieken.

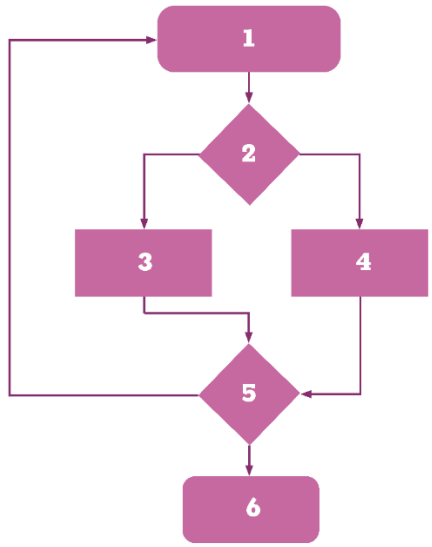
Proces Flow Test

LD27	Leer een testscript maken door het uittekenen van het proces/programma. (K3)

De eerste testtechniek die we gaan bespreken is een techniek die de procesflow-techniek **[XI]** genoemd wordt. Deze techniek kun je goed gebruiken voor het testen van flows in code, van applicaties, van processen of ketens van processen.

Met de procesflow-testtechniek kun je op hoofdlijnen de structuur van een programma, of proces in kaart brengen, inclusief beslismomenten en fout paden. De techniek zorgt ervoor dat de paden binnen een proces/programma inzichtelijk gemaakt worden. Op deze manier kun je controleren of de main-flow met de belangrijkste beslismomenten en de belangrijkste foutpaden in ieder geval werken. Wellicht ten overvloede, maar je kunt ook een processchema tekenen op basis van je code.

Voorbeeld van een procesflow:



Beschrijving:

1. Open app en toon beginscherm.
2. Is de klant een bestaande klant? Indien ja -> 3, indien nee -> 4.
3. Toon inlog scherm.
4. Toon account aanmaak scherm.
5. Inloggen/Account aanmaken & inloggen succesvol? Indien ja -> 6, indien nee -> terug naar 1.

Werkwijze om een Process Flow Diagram op te stellen

Begin met het analyseren van de informatie, het proces of de programmacode. Daarbij kun je een diagram tekenen dat lijkt op het voorbeeld van de procesflow. Tijdens het schetsen kun je problemen tegenkomen bij het verbinden van de lijnen, wat kan betekenen dat je een defect hebt gevonden. Probeer bij het opstellen van het diagram zo uitgebreid mogelijk te zijn. Verzamel informatie over de secties waar informatie ontbreekt, of waar je de lijnen niet aan elkaar kunt knopen, door teamleden en/of de klant te vragen naar het bedoelde gedrag van het ontbrekende onderdeel of pad.

In het processtroomdiagram betekent een rechthoek een actie en een ruit een beslissing. Er wordt een lijn met een pijl getrokken van de actie en beslissing naar de vervolgactie en -beslissing. Door het op deze manier te tekenen, is het eenvoudig om te bepalen welke paden er bestaan in het proces of de code, en of je de paden kunt testen.

Stel je een mobiele app voor waarvan je het inlogproces wilt testen. Schematisch wordt het weergegeven in de vorige procesflow.

Als je een diagram op deze manier tekent, kun je duidelijk maken welke stromen er zijn en welke je kunt testen. Door een 'beschrijving' toe te voegen in dit voorbeeld maak je het proces nog duidelijker en zou het voor iedereen in het team mogelijk moeten zijn om de test ook uit te voeren.

Het Testscript Maken

Eerst schrijven we alle flows uit. We leiden deze af uit de tekening.

Testcase	Schema	Omschrijving
1.	1-2-3-5-6	Succesvolle inlogpoging; bestaande klant
2.	1-2-3-5-1	Onsuccesvolle inlogpoging; bestaande klant
3.	1-2-4-5-6	Succesvolle inlogpoging; nieuwe klant
4.	1-2-4-5-1	Onsuccesvolle inlogpoging; nieuwe klant

Vervolgens werken we de stappen verder uit:

#Testcase 1. Succesvolle inlogpoging; bestaande klant	
Stap 1. Open het scherm met een bestaande klant (pre-conditie)	
Stap 2. Toon inlog scherm (pre-conditie)	
Stap 3. Log in met geldige credentials (pre-conditie)	
Test: dat het vervolg scherm getoond wordt (uitvoer voorspelling)	
(Daadwerkelijk) resultaat	Ok/ Niet ok

Door de testcase in detail uit te schrijven kunnen we nu kijken naar de verschillende onderdelen ervan. Het eerste onderdeel zijn de pre-condities, dat zijn in deze testcase stap 1 tot en met 3. De pre-condities beschrijven de condities waaraan voldaan moet worden om de testcase uit te voeren. Het volgende onderdeel is de test zelf met de uitvoervoorspelling. Het laatste onderdeel is het daadwerkelijke resultaat. Dit volgt uit de uitvoervoorspelling en is Ok of Niet ok. Zo ziet een gedetailleerde testcase er dus uit. Deze testcase noemen we een logische testcase. Om compleet te zijn hebben we de andere gedetailleerde testcases toegevoegd.

#Testcase 2. Onsuccesvolle inlogpoging; bestaande klant	
Stap 1. Open het scherm met een bestaande klant	
Stap 2. Toon inlog scherm	
Stap 3. Log in met ongeldige credentials	
Test: dat je teruggestuurd wordt naar het beginscherm (uitvoer voorspelling)	
Resultaat	Ok/ Niet ok

#Testcase 3. Onsuccesvolle inlogpoging; nieuwe klant	
Stap 1. Open het scherm met een nieuwe klant	
Stap 2. Toon inlog scherm	
Stap 3. Log in met ongeldige credentials	
Test: dat je teruggestuurd wordt naar het beginscherm (uitvoer voorspelling)	
Resultaat	Ok/ Niet ok

#Testcase 4. Succesvolle inlogpoging; nieuwe klant	
Stap 1. Open het scherm met een nieuwe klant	
Stap 2. Toon inlog scherm	
Stap 3. Log in met geldige credentials	
Test: dat het vervolg scherm getoond wordt (uitvoer voorspelling)	
Resultaat	Ok/ Niet ok

Nu de testcases beschreven zijn, kun je ze gaan uitvoeren. Om dit te kunnen doen moet je de juiste data gaan zoeken om de testcases te implementeren en uit te voeren. In ons voorbeeld zijn dit klantgegevens. Je kunt dit doen met bestaande data of je kunt er data voor maken of aanpassen zodat je exact aan de pre-condities van de testcases voldoet. Nadat je dit gedaan hebben noemen we de testcases ‘fysieke testcases’.

Semantische Testtechniek

LD28	Leer een testscript maken met behulp van de semantische test. (K3)

Een andere techniek die je kunt gebruiken is de semantische testtechniek. **[XII]**
Bij deze techniek teken je geen schema, maar maak je gebruik van de woorden ALS, DAN en ANDERS om je testcases te beschrijven. De semantische test kun je goed gebruiken als testtechniek voor eenvoudige en middel-complexe functionaliteit. Je kunt het ook goed inzetten voor het testen van requirements voor een applicatie. Voor zowel requirements die wel onderling verband hebben als voor requirements die dat niet hebben, is de semantische testtechniek geschikt. Bijvoorbeeld verschillende controles op schermen.

We nemen dezelfde beschrijving van de functionaliteit als bij de procesflow techniek:

Beschrijving:

1. Open app en toon beginscherm.
2. Is de klant een bestaande klant? Indien ja -> 3, indien nee -> 4.
3. Toon inlog scherm.

4. Toon account aanmaak scherm.
5. Inloggen succesvol? Indien ja -> 6 indien nee -> terug naar 1.
6. Toon vervolg scherm.

Met eerder genoemd voorbeeld kom je tot het volgende semantische test script:

1.	ALS de app wordt geopend DAN toon het inlogscherm ANDERS geen actie
2.	ALS de app wordt geopend DAN toon het inlogscherm ANDERS geen actie
3.	ALS de app wordt geopend DAN toon het inlogscherm ANDERS geen actie

Met een onderling verband zou het dan worden:

```
ALS de app wordt geopend
DAN toon het inlogscherm
    ALS klant is bestaande klant
    DAN ALS inloggen is succesvol
        DAN toon vervolgscherm
        ANDERS toon beginscherm
    ANDERS toon account aanmaak scherm
ANDERS geen actie
```

Let op de uitlijning hier. De uitgelijnd ALS, DAN en ANDERS horen bij elkaar. Dit heet een geneste statement en wordt ook gebruikt in programmacode. Een ALS heeft verder altijd een DAN en een ANDERS. Op die manier beschrijf je alle mogelijkheden van de functionaliteit. Merk op dat er één uitzondering is, dat is de DAN die gebruikt wordt om de ALS van de volgende stap te koppelen. Feitelijk staat hier EN. Je kunt in dat geval EN gebruiken in plaats van DAN. Dit zie je op de derde regel in het voorbeeld.

De ALS beschrijft de pre-conditie. Een pre-conditie is een voorwaarde waaraan een testcase moet voldoen om uitgevoerd te kunnen worden. De DAN beschrijft de actie die moet gebeuren bij het voldoen aan de conditie. De ANDERS tenslotte beschrijft wat er dient te gebeuren als niet aan de conditie voldaan wordt. Dit betreft dus of een fout pad of een verwijzing naar volgende functionaliteit.

Je hebt nu geleerd hoe je met behulp van de semantische testtechniek een test kunt ontwerpen. Hierboven heb je de testcases op een high-level niveau beschreven. Het verdient nu aanbeveling om net als bij de procesflow een verdere detailuitwerking te maken.

Nr.	High-level testcase	Nr.	Detail testcase
1.	ALS de app wordt geopend DAN toon het inlogscherm ANDERS geen actie	1.	Controleer of het inlogscherm getoond wordt.
		2.	Controleer of er geen scherm getoond wordt bij geen actie.
2.	ALS klant is bestaande Klant DAN toon inlogscherm ANDERS toon account aanmaak scherm	3.	Controleer of het inlogscherm getoond wordt.
		4.	Controleer of het account aanmaak scherm getoond wordt.
3.	ALS inloggen is succesvol DAN toon vervolg scherm ANDERS toon Beginscherm	5.	Controleer of bij een succesvolle inlogpoging het vervolgscherm getoond wordt.
		6.	Controleer of bij onsuccesvol inloggen het beginscherm getoond wordt.

Vervolgens werken we de stappen verder uit:

#Testcase 1. Inlogscherm tonen	
Stap 1. Open de app	
Stap 2. Toon inlog scherm	
Test: dat het inlog scherm getoond wordt (uitvoer voorspelling)	
Resultaat	Ok/ Niet Ok

#Testcase 2. Inlogscherm niet tonen	
Stap 1. Installeer de app, maar open de app niet	
Test: dat er niets geopend wordt (uitvoer voorspelling)	
Resultaat	Ok/ Niet Ok

De tweede testcase lijkt misschien overbodig. Dat is misschien zo in dit voorbeeld, maar vaak heeft het juist wel zin om expliciet te controleren of er geen actie plaatsvindt.

Beslistabellen

LD29	Leer functionaliteit testen met behulp van een beslistabel. (K3)

De beslistabellentest **[VIII]** gebruik je voor het testen van de structuur en logica van een programma, of proces. Je maakt hierbij duidelijk onderscheid tussen verschillende condities waar je wel of niet aan voldoet, en de acties als gevolg van die beslissingen bij een conditie. Op deze manier kun je goed controleren op volledige werking van de functionaliteit. Het is een zware techniek waarmee je grondig kunt testen.

Beslissingstabellen worden opgebouwd door alle condities uit te schrijven. Bij het formuleren van condities in een beslissingstabel is het belangrijk om een paar dingen in gedachten te houden. Condities moeten worden geformuleerd in eenvoudige taal en in enkelvoudige termen. Ze moeten ook zo geformuleerd worden dat de uitkomst van de vraag gemakkelijk te begrijpen is als deze met 'Ja' of 'Nee' beantwoord wordt. Op deze manier is de tabel gemakkelijk te begrijpen en te repliceren.

Een beslistabel stel je op door alle condities uit te schrijven. De condities enkelvoudig en positief te formuleren. Vervolgens vul je alle acties in en dan begin je met het vullen van de beslissingen met J en N. Excel is een goede tool om je hierbij te helpen.

Bijvoorbeeld:
We hebben het volgende requirement:
Als leeftijd > 15 of lengte > 135 heb je toegang tot de attractie. Als je over een VIP Ticket beschikt dan heb je direct toegang to de attractie. Dit geeft de volgende tabel:

Condities								
Leeftijd > 15 jaar								
Lengte > 135 cm								
VIP Ticket								
Acties								
…								

De volgende stap is het invullen van de acties die de uitkomsten van de condities zullen worden. Als je dat gedaan hebt ga je achter de condities de beslissingen vullen met J en N. Omdat in een beslistabel de condities booleaan (J/N) zijn, is het aantal 'mogelijke beslissingen' altijd 2 tot de macht 'aantal condities'. Dus in dit voorbeeld met 3 condities is het aantal mogelijke beslissingen 2 tot de macht 3, dat is $2^3 = 2 \times 2 \times 2 = 8$. Bij 4 condities heb je $2^4 = 2 \times 2 \times 2 \times 2 = 16$ mogelijke beslissingen.

Als het aantal kolomen bekend is, kun je starten met invullen van 'J' en 'N' in de kolommen. Een gemakkelijke manier om dit te doen is om de eerste helft van de eerste conditie met 'J' te vullen en de tweede helft met 'N'. Dus kolommen 1-4 bevatten 'J' en kolommen 5-8 'N'. In totaal heb je nu dus 8 kolommen gevuld met de beslissingen voor de eerste conditie.

Condities	1	2	3	4	5	6	7	8
Leeftijd > 15 jaar	J	J	J	J	N	N	N	N
Lengte > 135 cm								
VIP Ticket								
Acties								
Direct Toegang								
Toegang								
Geen toegang								

Zodra je de eerste conditie (rij) hebt ingevuld, kun je doorgaan naar de volgende conditie. Een goede regel om te volgen is om de beslissingen van de vorige conditie te kopiëren naar de volgende rij en de tweede helft van de opeenvolgende 'J'-gedeelten te wijzigen in 'N'-gedeelten en de eerste helft van de opeenvolgende 'N'-gedeelten in 'J', zoals hier wordt getoond:

Condities	1	2	3	4	5	6	7	8
Leeftijd > 15 jaar	J	J	J	J	N	N	N	N
Lengte > 135	J	J	N	N	J	J	N	N
VIP Ticket								
Acties								
Direct Toegang								
Toegang								
Geen toegang								

Zo kun je verder gaan met de volgende condities totdat de tabel gevuld is, zoals we hier kunnen zien:

Condities	1	2	3	4	5	6	7	8
Leeftijd > 15 jaar	J	J	J	J	N	N	N	N
Lengte > 135	J	J	N	N	J	J	N	N
VIP Ticket	J	N	J	N	J	N	J	N
Acties								
Direct Toegang								
Toegang								
Geen toegang								

Vervolgens kun je de uitkomst van de verschillende condities koppelen aan een actie. In ons voorbeeld moet iemand 15 jaar of ouder zijn of langer dan 135 cm zijn om toegang te krijgen. En een VIP ticket hebben om direct toegang te krijgen tot de attractie. Kolom 1 beschrijft een persoon die 15 jaar of ouder is en langer is dan 135 cm en bovendien over een VIP Ticket beschikt. Daarom moet VIP toegang worden verleend aan de persoon die in kolom 1 wordt beschreven. Schrijf een ‘X’ in het veld dat de juiste actie aangeeft.

Vervolgens kun je de tweede kolom invullen. Aangezien de conditie’VIP ticket’ een ‘N’ oplevert, zal geen directe toegang worden verleend aan een persoon die aan deze criteria voldoet. Alle andere acties in de tabel moeten nu worden ingevuld totdat er een actie is voor elke kolom. Je hebt nu de 8 testgevallen om de functionaliteit grondig te testen, zoals je hier kunt zien:

Condities	1	2	3	4	5	6	7	8

Leeftijd > 15 jaar	J	J	J	J	N	N	N	N
Lengte > 135	J	J	N	N	J	J	N	N
VIP Ticket	J	N	J	N	J	N	J	N
Acties								
Direct Toegang	X		X		X			
Toegang		X		X		X		
Geen toegang							X	X

Grenswaardenanalyse

LD30	Leer je testen verdiepen door het gebruik van grenswaardenanalyse. (K3)

Grenswaardenanalyse **[XIV]**, **[XV]** is het opzoeken van de grenzen van waarden in inputvelden, condities, API's, batchprocessen, plugins. Je zoekt hier bewust de grens op om het onderscheid in functionaliteit te kunnen vaststellen. Met het toepassen van grenswaardenanalyse kun je goed controleren op volledige en correcte werking van de functionaliteit.

Bijvoorbeeld iemand dient 18 jaar of ouder te zijn om bepaalde content te kunnen zien. Dit kun je controleren door te testen met een leeftijd willekeurig gekozen boven de 18, laten we zeggen 23. Maar beter het is beter om dit controleren met een leeftijd van 17 (geen toegang), 18 en 19 (wel toegang). Dit omdat veel fouten ontstaan door verkeerde waarden voor de grenzen denk aan <, >,<=, >=, et cetera. Dus maak 3 testcases per grens: 1 op de grens, 1 gelijk links en 1 gelijk rechts van de grens.

Laten we eens kijken naar een ander voorbeeld van senioren die korting krijgen op het openbaar vervoer om te illustreren hoe grenswaardeanalyse kan worden toegepast. Neem de volgende requirement: *'Mensen van 65 jaar en ouder krijgen 20% korting op hun kaartje voor het openbaar vervoer.'* Wanneer we hier grenswaarde analyse op toepassen, zal dit leiden tot 3 testgevallen per grens, wat de volgende testgevallen oplevert: 64, 65, 66 (ongeldig, geldig, geldig).

Equivalentieklassen

LD31	Leer je testen verdiepen door het gebruik van equivalentieklassen. (K3)

Equivalentieklassen **[XIII], [XIV]** zijn klassen van invoerwaarden die tot eenzelfde soort verwerking leiden. We maken hierbij een onderscheid tussen 'geldige' en 'ongeldige' equivalentieklassen. De invoerwaarden uit geldige equivalentieklassen worden correct verwerkt en de invoerwaarden uit ongeldige equivalentieklassen zouden kunnen resulteren in een foutmelding. Bij dit principe wordt op iedere equivalentieklasse 1 testcase gebaseerd.

Voorbeeld:
Een invoercontrole op leeftijd heeft het volgende requirement: 6 >= leeftijd < 12 (de leeftijd moet 6 of groter, maar kleiner dan 12 zijn).
In dit voorbeeld kun je drie equivalentieklassen onderscheiden, namelijk:

- Leeftijd <6 jaar
- Leeftijd in het bereik 6 tot en met 11 jaar
- Leeftijd >= 12 jaar

We komen dan als we equivalentieklassen toepassen dan tot de volgende testcases:

- Leeftijd = 4 (ongeldig)
- Leeftijd = 8 (geldig)
- Leeftijd = 15 (ongeldig)

Wat opvalt is dat je minder testcases hebt dan bij grenswaardenanalyse. Met andere woorden de dekking van je testcases die je toepast bij grenswaardenanalyse is veel beter. Toch is equivalentieklassen een handig principe om toe te passen als je met iets minder dekking wilt testen of als je bijvoorbeeld alfanumerieke invoerwaardes gaat testen. Denk bijvoorbeeld aan een lijstje met mogelijkheden om je te identificeren: paspoort, rijbewijs, id-kaart, et cetera.

Checklist-gebaseerde testtechniek

LD32	Leer hoe je kunt testen met behulp van een checklist. (K3)

De checklist-gebaseerde test **[XVI]** is een techniek waarbij je een checklist opstelt en die gebruikt om te gaan testen. Deze techniek kun je goed gebruiken bij herhalende activiteiten zoals bijvoorbeeld het opleveren van software naar een test-, acceptatie-, of productieomgeving. Ook bepaalde aspecten van security zijn zaken die je prima met een checklist kan afhandelen. Op de checklist vermeld je de controles die je uitvoert en je geeft de mogelijkheid om deze af te vinken met Ok, dan wel Niet Ok. Je kunt deze checklist opstellen op basis van ervaring, maar je kunt ook de oorzaak van eerdere incidenten toevoegen aan de checklist. Door de checklist toe te voegen aan je proces, maak je het geheel voorspelbaar en consistent. Het kan ook een manier zijn om kennis van een ervaren medewerker vast te leggen, zodat anderen ook deze controles kunnen uitvoeren. De checklist is prima te combineren met andere testtechnieken zoals de CRUD matrix of de semantische testtechniek.

Een voorbeeld van een checklist.

Nr.	Omschrijving testcase	Ok / Niet ok	Opmerking
1.	Controleer of het inlogscherm getoond wordt.	Ok	
2.	Controleer of het veld ‘inloggen’ getoond wordt	Ok	
3.	Controleer of het veld ‘password’ getoond wordt	Ok	

4.	Controleer of invoer in het veld ‘inloggen’ met 35 posities geaccepteerd wordt	Ok	
5.	Controleer of invoer in het veld ‘inloggen’ met 36 posities niet geaccepteerd wordt	Niet Ok	Er kunnen meer dan 36 posities ingevoerd in dit veld. Zie defect 123.
6.	Controleer of invoer in het veld ‘password’ met 35 posities geaccepteerd wordt	Ok	
7.	Controleer of invoer in het veld ‘password’ met 36 posities niet geaccepteerd wordt	Niet Ok	Dit veld staat meer dan 36 posities toe. Zie defect 138.
8.	Controleer of als ik een ‘password’ van minder dan 16 karakters invoer de melding getoond wordt ‘uw password moet minimaal 16 karakters bevatten’	Ok	
9.	Controleer of als ik een ‘password’ zonder speciale karakters opvoer de melding ‘uw password moet minimaal 1 van de volgende speciale karakters bevatten -_()*&^%$#@!?/><‘ verschijnt	Ok	
10.	Controleer of als ik een ‘password’ zonder hoofdletter opvoer de melding ‘uw password moet minimaal 1 hoofdletter bevatten’ verschijnt	Ok	
11.	Controleer of Ik kan inloggen met de rol Beheerder		
12.	…		

Je zou dit kunnen gebruiken voor het controleren van:

- layout
- input waardes (schermen en files)
- output waardes
- bestands- en veldvalidatie
- juist foutmeldingen
- missende velden
- veldlengte
- veldsoort (datatype)
- positie

Met checklists kunnen minder ervaren medewerkers dezelfde dingen controleren als hun meer ervaren collega's. Eenmaal uitgevoerd, moet een checklist echter regelmatig worden

onderhouden, omdat sommige van de repetitieve acties in de loop van de tijd kunnen veranderen. Testen op basis van checklists kunnen goed gecombineerd worden met andere testtechnieken zoals de CRUD-matrix, grenswaardeanalyse, equivalentiepartitionering of de semantische testtechniek. Naast het opstellen van een checklist kun je ook gebruik maken van al bestaande checklisten om applicaties te testen. Bijvoorbeeld op het gebied van security en usabilty zijn er online uitstekende checklisten te vinden die je zo kunt inzetten in je project.

Pairwise Testing Testtechniek

LD33	Leer de testtechniek pairwise testing. (K3)

Pairwise testing is een methode om defects te vinden door telkens twee waarden van een variabele te combineren. Pairwise testing gaat ervanuit dat de meeste defects veroorzaakt worden door één factor of door interactie van twee verschillende factoren. Waar het testen van alle mogelijke combinaties van inputwaarden veel tijd en moeite kost, kan pairwise testing erg effectief zijn door elke combinatie van 2 willekeurige factoren te testen. Het zorgt bovendien voor betere testdata.

Pairwise testing werkt als volgt: Stel je een systeem voor met 3 input variabelen. Deze zijn bijvoorbeeld bestandsformaat, toegangsniveau en autorisatie. Deze gedefinieerde variabelen hebben verschillende waarde die in de volgende tabel beschreven zijn.

Bestandsformaat	**Toegangsniveau**	**Autorisatie**
.pdf	Groen	Medewerker
.gif	Oranje	Beheerder
.docx	Paars	
.jpeg		

Deze velden hebben 4 x 3 x 2 waarden dus in totaal 24 unieke combinaties.

TC	**Bestandsformaat**	**Toegangsniveau**	**Autorisatie**
1.	.pdf	Groen	Medewerker
2.	.pdf	Oranje	Medewerker
3.	.pdf	Paars	Medewerker
4.	.pdf	Groen	Beheerder
5.	.pdf	Oranje	Beheerder
6.	.pdf	Paars	Beheerder
7.	.gif	Groen	Medewerker
8.	.gif	Oranje	Medewerker
9.	.gif	Paars	Medewerker
10.	.gif	Groen	Beheerder

11.	.gif	Oranje	Beheerder
12.	.gif	Paars	Beheerder
13.	.docx	Groen	Medewerker
14.	.docx	Oranje	Medewerker
15.	.docx	Paars	Medewerker
16.	.docx	Groen	Beheerder
17.	.docx	Oranje	Beheerder
18.	.docx	Paars	Beheerder
19.	.jpeg	Groen	Medewerker
20.	.jpeg	Oranje	Medewerker
21.	.jpeg	Paars	Medewerker
22.	.jpeg	Groen	Beheerder
23.	.jpeg	Oranje	Beheerder
24.	.jpeg	Paars	Beheerder

De testtechniek van het pairwise testen houdt in dat je je richt op slechts twee invoerwaarden tegelijk. Als je daarom in de eerste stap alleen naar de eerste twee kolommen kijkt, levert dat de volgende tabel met combinaties op:

Bestandsformaat	**Toegangsniveau**
.pdf	Groen
.pdf	Oranje
.pdf	Paars
.gif	Groen
.gif	Oranje
.gif	Paars
.docx	Groen
.docx	Oranje
.docx	Paars
.jpeg	Groen
.jpeg	Oranje
.jpeg	Paars

Hier heb je elke invoerwaarde uit kolom ‘bestandsformaat’ met elke verschillende invoerwaarde in kolom ‘Toegangsniveau’, wat twaalf mogelijkheden oplevert. Aangezien er vier verschillende ‘bestandsformaat’ invoerwaarden zijn en drie verschillende ‘toegangsniveau’ invoerwaarden, zijn er nu dus 4 invoerwaarden x 3 invoerwaarden = 12 verschillende testgevallen.

Zodra alle mogelijke combinaties voor de invoerwaarden van de eerste twee kolommen duidelijk zijn, moeten we dezelfde techniek toepassen op de tweede en derde kolom om erachter te komen hoeveel combinaties mogelijk zijn. Aangezien er drie verschillende

'Toegangsniveau' invoerwaarden zijn en twee verschillende 'Autorisatie' invoerwaarden, komen we uit op 3 invoerwaarden x 2 invoerwaarden = 6 testgevallen zoals hier te zien is:

Toegangsniveau	Autorisatie
Groen	Medewerker
Groen	Beheerder
Oranje	Medewerker
Oranje	Beheerder
Paars	Medewerker
Paars	Beheerder

Je zou op dit punt kunnen aannemen dat je 18 testgevallen hebt; dit is echter niet het geval bij pairwise testen. De waarde van deze techniek is om een zo relevant mogelijke dekking te krijgen met een kleiner aantal testgevallen.

Als je alle 24 mogelijke combinaties in beschouwing neemt en nu pairwise testen implementeert, moet je ervoor zorgen dat alle hierboven genoemde combinaties van de vorige twee tabellen, die twee invoerwaarden tegelijk vergelijken, vertegenwoordigd zijn in de testgevallen, zodat alle andere testgevallen kunnen worden verwijderd.

Als je in de lijst met alle mogelijke testgevallen de paren van de vergelijking van de invoerwaarden 'bestandsformaat' en 'toegangsniveau' in het groen markeert dan krijg je het volgende:

TC	Bestandsformaat	Toegangsniveau	Autorisatie
1.	.pdf	Groen	Medewerker
2.	.pdf	Groen	Beheerder
3.	.pdf	Paars	Medewerker
4.	.pdf	Paars	Beheerder
5.	.pdf	Oranje	Medewerker
6.	.pdf	Oranje	Beheerder
7.	.gif	Groen	Medewerker
8.	.gif	Groen	Beheerder
9.	.gif	Paars	Medewerker
10.	.gif	Paars	Beheerder
11.	.gif	Oranje	Medewerker
12.	.gif	Oranje	Beheerder
13.	.docx	Groen	Medewerker
14.	.docx	Groen	Beheerder
15.	.docx	Paars	Medewerker
16.	.docx	Paars	Beheerder
17.	.docx	Oranje	Medewerker

18.	.docx	Oranje	Beheerder
19.	.jpeg	Groen	Medewerker
20.	.jpeg	Groen	Beheerder
21.	.jpeg	Paars	Medewerker
22.	.jpeg	Paars	Beheerder
23.	.jpeg	Oranje	Medewerker
24.	.jpeg	Oranje	Beheerder

Als je deze tabel met groene arceringen vergelijkt met de tabel waarin de test-paren van de invoerwaarden 'toegangsniveau' en 'autorisatie' staan en je ze geel markeert, dan zie je dat je al drie van de zes combinaties hebt gemarkeerd.

Bij het pairwise testen moet je ervoor zorgen dat de andere drie combinaties ook vertegenwoordigd zijn. We passen dit eerst toe op het eerste deel van de tabel. Je kunt alle rijen die hieraan voldoen geel markeren en daarmee alle combinaties dekken. Door de combinaties 'Toegangsniveau' en 'Autorisatie' te combineren met bijvoorbeeld de combinaties 'Bestandsformaat' '.pdf, .gif' en 'Toegangsniveau' kunnen we de resterende drie combinaties ook toepassen in de bestaande testgevallen. De resterende rijen (2, 4, 6, 7, 9, 11) kunnen worden verwijderd zoals hier te zien is:

TC	Bestandsformaat	Toegangsniveau	Autorisatie
1.	.pdf	Groen	Medewerker
~~2.~~	~~.pdf~~	~~Groen~~	~~Beheerder~~
3.	.pdf	Paars	Medewerker
~~4.~~	~~.pdf~~	~~Paars~~	~~Beheerder~~
5.	.pdf	Oranje	Medewerker
~~6.~~	~~.pdf~~	~~Oranje~~	~~Beheerder~~
~~7.~~	~~.gif~~	~~Groen~~	~~Medewerker~~
8.	.gif	Groen	Beheerder
~~9.~~	~~.gif~~	~~Paars~~	~~Medewerker~~
10.	.gif	Paars	Beheerder
~~11.~~	~~.gif~~	~~Oranje~~	~~Medewerker~~
12.	.gif	Oranje	Beheerder

Hoewel we nu het principe van pairwise testen al volledig toegepast hebben kunnen we hetzelfde doen met testgevallen 13 tot en met 24. Ook hier combineren we de unieke combinaties van de kolommen 'bestandsformaat' en 'toegangsniveau' selecteert, met de unieke combinaties van de kolommen 'toegangsniveau' en 'autorisatie', dan leidt dit tot de volgende tabel waarin vier testgevallen ('14', '16', '18' '19', '21', '23') kunnen worden verwijderd:

13.	.docx	Groen	Medewerker

~~14.~~	~~.docx~~	~~Groen~~	~~Beheerder~~
15.	.docx	Paars	Medewerker
~~16.~~	~~.docx~~	~~Paars~~	~~Beheerder~~
17.	.docx	Oranje	Medewerker
~~18.~~	~~.docx~~	~~Oranje~~	~~Beheerder~~
~~19.~~	~~.jpeg~~	~~Groen~~	~~Medewerker~~
20.	.jpeg	Groen	Beheerder
~~21.~~	~~.jpeg~~	~~Paars~~	~~Medewerker~~
22.	.jpeg	Paars	Beheerder
~~23.~~	~~.jpeg~~	~~Oranje~~	~~Medewerker~~
24.	.jpeg	Oranje	Beheerder

Uiteindelijk blijven de volgende twaalf testgevallen over:

TC	**Bestandsformaat**	**Toegangsniveau**	**Autorisatie**
1.	.pdf	Groen	Medewerker
3.	.pdf	Paars	Medewerker
5.	.pdf	Oranje	Medewerker
8.	.gif	Groen	Beheerder
10.	.gif	Paars	Beheerder
12.	.gif	Oranje	Beheerder
13.	.docx	Groen	Medewerker
15.	.docx	Paars	Medewerker
17.	.docx	Oranje	Medewerker
20.	.jpeg	Groen	Beheerder
22.	.jpeg	Paars	Beheerder
24.	.jpeg	Oranje	Beheerder

In dit voorbeeld wordt het duidelijk dat je testgevallen hebt die een grote verscheidenheid aan testgegevens dekken, maar je hebt bijvoorbeeld de combinatie van .pdf, oranje en beheerder niet getest (testgeval ‘6’).

Pairwise testing zorgt ervoor dat de dekking van je testgevallen vrij hoog kan zijn; alle afhankelijkheden tussen de verschillende variabelen worden echter niet getest **[XVII].**

Een voordeel van pairwise testing kan zijn dat de dekking van je testcases behoorlijk verhoogd kan worden, net als de het gemiddelde aantal defects per testcase wat je kunt vinden. Daarnaast kan het tijd besparen, omdat je minder testcases hoeft uit te voeren.

Er zijn nadelen aan het gebruik van pairwise testing. Je houdt namelijk geen rekening met afhankelijkheden tussen alle verschillende variabelen die je aan het testen bent. En soms levert

de techniek onrealistische combinaties op. Meer voorbeelden en tools van pairwise testing vind je op deze site **[XVII]**.

Relatie tussen dekking in de teststrategie en testtechnieken

In Practice 6 hebben we het opstellen van een test strategie besproken. Er is een verband tussen de teststrategie en het toepassen van testtechnieken. Waar we bij de teststrategie zagen dat deze verschillende risicoklasses opleverden, kunnen we nu de testtechnieken gaan inzetten om deze risicoklassen af te dekken. Sommige test technieken leveren een zware dekking op en kunnen dus goed gebruikte worden om testdoelen met een hoog risico af te dekken. Andere leveren een gemiddelde of lage dekking op en kunnen dus gebruikt worden voor gemiddelde of lage risico's. Je zou ook binnen een test techniek kunnen variëren met een hogere, gemiddelde of lagere dekking. In onderstaande tabel is beschreven wat de kenmerken eigenschappen zijn van de verschillende test technieken en waar je ze zoal voor in kunt zetten. Hou er verder rekening mee dat het inzetten van een testtechniek afhankelijk is van veel verschillende factoren. Je dient bijvoorbeeld wel over de voor de techniek geschikte test basis te beschikken. Merk op dat de testechniek 'CRUD' besproken zal worden in Hoofdstuk 4.

Testtechniek	**Geschikt voor welke processen en welke dekking levert het op**
Proces Flow Test	Geschikt voor processen waarbij het cruciaal is om de volledige stroom van gebeurtenissen en acties te testen, zoals workflow-toepassingen en bedrijfsprocessen, of complexe code die je in de flow wilt testen. Is goed toepasbaar in E2E, keten of acceptatietesten.
Semantische Test	Geschikt voor processen waarbij de betekenis en interpretatie van gegevens en berichten van belang zijn, zoals in communicatieprotocollen en gegevensuitwisselingssystemen. Vaak goed te gebruiken in systeem- en integratietesten.
Beslistabel	Geschikt voor processen met complexe bedrijfslogica en beslissingsregels, zoals financiële systemen of verzekeringsberekeningen. Kan op meerdere test levels ingezet worden om complexe software te testen. Kan voor een hogere dekking goed gecombineerd worden met Grenswaardeanalyse of equivalentieklasse.
Grenswaardeanalyse	Geschikt voor processen waarbij grenswaarden kritiek zijn, zoals financiële transacties, beveiligingstoepassingen, enzovoort. Het inzetten van grenswaardeanalyse levert over het algemeen een gemiddelde tot hoge dekking op.
Equivalentieklasse	Geschikt voor gestructureerde, goed gedefinieerde processen waarbij inputvariabelen kunnen worden opgedeeld in klassen. Levert een lagere dekking op als grenswaardeanalyse.
Checklist Based Testing	Geschikt voor processen waarbij gestructureerde lijsten met testitems kunnen worden gegenereerd en toegepast, zoals in

Testtechniek	Geschikt voor welke processen en welke dekking levert het op
	gebruikersacceptatietesten en beveiligingsaudits. De dekking is afhankelijk van hoe specifiek de checklist gemaakt is.
Pairwise Testing	Geschikt voor processen met veel combinaties van inputvariabelen, zoals configuratie- en compatibiliteitstesten in softwaretoepassingen. Kan ook toegepast worden op het genereren van test data. Levert een gemiddelde dekking, die vaak wel representatief en efficiënt is.
CRUD (Matrix)	Geschikt voor processen waarbij database-interacties en bewerkingen op gegevens cruciaal zijn, zoals in applicaties voor gegevensbeheer. Goed toepasbaar op authorisatiemodellen en levert hiervoor een hoge dekking op, mits volledig uitgevoerd.

In de volgende tabel, die zie je een aantal voorbeelden hoe deze verschillende testtechnieken zou kunnen koppelen aan onze eerder opgestelde teststrategie tabel. Zodat we ze inzetten om een ‘Hoge’, ‘Gemiddelde’ en ‘Lage’ dekking te bereiken. Bedenk wel dat de inzet en effectiviteit van een techniek sterk afhankelijk is van de specifieke context van het testproject, de testbasis en de te testen software. Bij de keuze zul je deze dan ook in ogenschouw moeten nemen.

Teststrategietabel		Testlevels			
Testdoel/testtype	**Statische test**	**Unittest**	**Integratie-test**	**Systeem-test**	**Acceptatie-test**
Orders uit Frankrijk	Code review	Semantische test + grenswaardeanalyse	Alle mogelijken en uitzonderingen testen	Beslistabellen + grenswaarde analyse	Proces flow test alle E2E scenario’s
Orders uit andere delen van Europa		Pairwise testing	Pairwise testing	Semantische test	Proces flow test met belangrijkste scenario’s testen
Orders uit andere delen van de wereld		Checklist met lage dekking	Checklist met lage dekking	Procesflow met lage dekking	Procesflow test met enkele scenario’s testen
Verkopers API	Code review	CRUD		Beslistabellen + grenswaarde analyse	
Inloggen	..	..	..	..	..
Zoekfunctie	..	..	..	..	..
Usability	Prototyping	Walkthrough		Eye tracking	Laten testen m.b.v. gebruiks-groepen
API Documentation	Review				

In deze practice hebben we een aantal test technieken besproken en uitgelegd hoe je ze kunt toepassen. Tevens hebben we de relatie beschreven tussen de teststrategie en het inzetten van testtechnieken. De practice “Leg je testen vast” leert je dus hoe je dit kunt toepassen in dagelijkse praktijk.

Definities

Boolean	Een resultaat dat maar twee mogelijk uitkomsten kan hebben, zoals Waar of Niet waar, Ja of Nee.
Dekking (coverage)	De mate waarin je testcases de te testen functionaliteit ten opzichte van de totaal mogelijke functionaliteit afdekken.
Detail testcase	Een testcase op een zeer gedetailleerd abstractieniveau. Gedetailleerde testgevallen bestaan bijna altijd uit pre-condities, een uitvoervoorspelling en een resultaat.
High-level testcase	Een testgeval op een hoog abstractieniveau dat meestal bedoeld is om gedetailleerde testgevallen af te leiden uit (of inzicht te geven in) de applicatie. Er is nog een verband tussen testlevels en het abstractieniveau van testcases in de hogere testlevels zul je over het algemeen minder abstracte testcases (dus meer high-level) aantreffen. Waar je in de unit test bijvoorbeeld een heel laag abstractieniveau hebt.
Logische testcase	Een testcase afgeleid van een testbasis bestaande uit pre-condities (de input) en post-condities (de acties of resultaten), op basis van de logica, dus nog zonder concrete data.
Uitvoervoorspelling	Het verwachte resultaat van de test case.
Pre-conditie	De conditie waaraan voldaan moet worden om een testcase uit te kunnen voeren.
Resultaat	Het daadwerkelijke resultaat van de testcase: Ok of Niet ok.
Testproces	De verzameling onderling gerelateerde activiteiten, bestaande uit testplanning, testbewaking en -controle, testanalyse, testontwerp, testimplementatie, testuitvoering en testafronding.
Testcase (testgeval)	Een reeks pre-condities, invoerwaarden, acties (indien van toepassing), verwachte resultaten en postcondities, ontwikkeld op basis van testcondities.
Testtechniek	Een gestructureerde manier om testcases af te leiden uit een test basis.
Testscript	Een reeks instructies voor het uitvoeren van een test.

Testontwerp	De activiteit die testgevallen afleidt en specificeert uit testcondities. **[V]**
Fysieke testcase	Een logisch testcase met testdata daaraan toegevoegd.
Variabele	Een gegeven, kenmerk of factor die kan variëren of veranderen.

Practice 8: Het belang van Goede Communicatie

LD34	Begrijp het belang van communicatie in je activiteiten als tester. (K3)

Je hebt nu een groot aantal vaardigheden geleerd om software op een goede en gestructureerde wijze te kunnen testen. Bij al deze vaardigheden is er één vaardigheid die de voorgaande practices kan versterken. Dat is namelijk de vaardigheid van goed communiceren.

De basisvaardigheden van communiceren bestaan uit spreken, schrijven, luisteren en lezen. Als je één van deze vaardigheden inzet om je boodschap te communiceren is het belangrijk dat je verifieert of je boodschap is overgekomen.

Als we communiceren hebben we altijd een zender van de boodschap en een ontvanger van de boodschap. Goede communicatie houdt daarbij in dat je als zender verifieert dat de boodschap is overgekomen, of in het ander geval dat je als ontvanger verifieert of de gecommuniceerde boodschap ook was wat jij begrepen had. Naast verbale communicatie is er non-verbale communicatie. Ook non-verbale communicatie kun je inzetten om te verifiëren of je boodschap is overgekomen.

Verifiëren is noodzakelijk omdat iedereen persoonlijke filters heeft. Deze filters zorgen ervoor dat de boodschap die verzonden wordt door iedereen anders verwerkt wordt. De persoonlijke filters die iedereen heeft zijn gebaseerd op persoonlijke ervaring, achtergrond, opvoeding en overtuiging. Hierdoor kan een boodschap die op een bepaalde manier bedoeld is voor iedereen op een andere manier overkomen. Het is dus voor jou als tester belangrijk om te verifiëren of jouw boodschap is overgekomen zoals de boodschap bedoeld is. Omgekeerd is het ook

belangrijk om te verifiëren of de boodschap die jij ontvangen hebt ook de boodschap is zoals de zender deze afgegeven heeft.

Deze practice dien je zoveel mogelijk toe te passen op alle andere practices die je geleerd hebt en op al je andere activiteiten als tester. Bij alles wat je communiceert dien je jezelf af te vragen of de boodschap goed is overgekomen bij de andere partij. Dit kan soms best moeilijk zijn om te doen omdat je mensen het gevoel kunt geven dat je ze controleert, of dat je te veel bemoeit met hun werk. Ook kunnen mensen heel zeker overkomen, waardoor je zelf terughoudend bent met doorvragen. Deze eigenschap vereist daarom de nodige takt en mensenkennis.
Ook als je zelf een boodschap krijgt kun je zorgen dat je verifieert of je het goed begrepen hebt. Dit is zo belangrijk omdat softwareontwikkeling een complex geheel is van activiteiten die vaak niet tastbaar zijn.

Laten we een aantal voorbeelden noemen van hoe je dit principe toe zou kunnen passen in combinatie met de testpractices. De practice ‘geen aannames doen’ gaat ook uit van dit principe. Omdat in dit principe je geleerd wordt alles te verifiëren of valideren en er niet op te vertrouwen dat het is zoals het is.

Bij de practice ‘Test zo vroeg mogelijk’ is het belang van reviewen aangestipt. Het reviewen of laten reviewen van producten is een uitstekende manier om de kwaliteit van deze producten in een vroeg stadium te verhogen. Het gaat bij reviewen ook weer om het verifiëren of het product juist, kwalitatief goed en volledig is. Onder producten kun je alles verstaan van functioneel ontwerp, requirement, testscript, resultaat van een risicoanalyse, et cetera. Wees je er dus wel bewust van dat je niet simpelweg één van deze producten ter review verstuurd. Zorg er daarom voor dat de ontvanger begrijpt wat het product inhoudt en wat er van hem verwacht wordt in de reviewtaak die je hem of haar geeft. In dat geval zal de waarde van de review hoger zijn.

Ook de practice ‘leg je testen vast’ gaat uit van goede communicatie. Door zaken vast te leggen en te beschrijven maak je communicatie al kwalitatief beter doordat je het op een later moment terug kunt zoeken en doordat je het ook kunt delen (en dus ook laten reviewen door) anderen.

Een ander aspect van communicatie is verwachtingsmanagement. Het is belangrijk in je werk als tester om juiste en realistische verwachtingen neer te leggen bij de verschillende stakeholders. Dit is van belang omdat er op deze manier zo weinig mogelijk verrassingen ontstaan en om te voorkomen dat verwachting en uiteindelijk resultaat te ver uit elkaar gaan liggen. Het is daarom belangrijk tijdig te communiceren als je bijvoorbeeld voorziet dat een het maken van een testscript of het uit voeren van een test veel langer gaat duren dan in eerste instantie gedacht werd. Hou hier dus rekening mee in je communicatie.

Goede communicatie toepassen in een testproces.

Binnen een test proces zijn er vele momenten waarin communicatie belangrijk is. We hebben in eerdere practices geleerd dat het belangrijk is om dit toe te passen onder andere in de practice 'Wees Specifiek' maar ook in de practice 'Leg je testen vast'. In de practice 'Alles testen is onmogelijk' hebben we geleerd dat we een Risico Gebaseerde Teststrategie kunnen opstellen. In al deze practices komt communicatie terug. Natuurlijk moeten we ook na de uitvoering van de testen communiceren met de opdrachtgever over de behaalde testresulaten. Het zou immers eigenaardig zijn als we na afloop van het proces de software opleverden zonder de opdrachtgever inzage te geven in de behaalde resultaten tijdens het testen van de software. Ook tijdens een lang ontwikkel- en testtraject wil je de stakeholders op de hoogte houden.

We kunnen dit doen door middel van een eenvoudige rapportage. In de rapportage hoeven we eigenlijk maar drie onderwerpen te benoemen: Status van testcases, defects en risico's.
Voor testcases zou je in de rapportage op kunnen nemen de statussen die in onderstaande tabel 'Voorbeeld statussen testcases' zijn beschreven.

Testcases	
Status	**Omschrijving**
Niet gestart	De testcases die nog niet gestart zijn
In uitvoering	De testcases die wel gestart zijn maar nog in de uitvoering zitten
Niet van toepassing	Testcases die wel ontworpen zijn maar niet meer van toepassing
Gefaald	De testcases die niet succesvol zijn en dus geleid hebben tot een defect
Geslaagd	De testcases die succesvol afgerond zijn
Totaal	Het totaal van de testcases

'Voorbeeld statussen testcases'

Voor defects zou je in je rapportage kunnen denken aan de volgende statussen. Neem er notie van dat er soms meer statussen mogelijk zijn als in de tabel beschreven zijn, maar voor een eenvoudig testproces kun je de onderstaande gebruiken.

Defects	
Status	**Omschrijving**
Nieuw	Defects die pas gevonden zijn

Open	Defects die nog open staan
Toegewezen	Defects die zijn toegewezen aan iemand die ze moet gaan onderzoeken. Zodat de impact vastgesteld kan worden.
Aan het oplossen	Defects die gerepareerd worden
Klaar voor Hertest	Defects die gereed zijn om gehertest te worden.
Hertesten	Defects die op het moment gehertest worden
Hertest OK	Defects die opgelost zijn na de hertest
Hertest Niet OK	Defects die nog niet opgelost zijn na de hertest
Geparkeerd	Defects waarvoor gekozen is om ze niet gelijk op te lossen. Bijvoorbeeld omdat ze teveel impact hebben op de huidige sprint, of omdat er meer analyse gedaan moet worden om tot een goede oplossing te komen.

'Voorbeeld statussen Defects'

Voor defects is het aan te bevelen om ook de ernst op te nemen in je rapportage. Hieronder zie je een voorbeeld van verschillende niveaus van ernst die je zou kunnen gebruiken.

Ernst van Defects	
Ernst	**Omschrijving**
Blokkerend	Het defect blokkeert andere functionaliteit, die zodoende niet getest kan worden, waardoor er onzekerheid ontstaat over de werking van de deze testcases
Ernstig	Het defect is ernstig en moet opgelost worden, maar blokkeert geen andere functionaliteit.
Niet ernstig	Het defect is terecht, maar niet ernstig genoeg. Er is een acceptabele workaround aanwezig.
Cosmetisch	Het defect is cosmetisch of een schrijf of spelfout.

'Voorbeeld Ernst Defects'

Rapportage op risico's zou je kunnen doen door te laten zien hoeveel inspanning er gedaan is om de testdoelen die we bepaald hebben in de risicoanalyse voldoende af te dekken. Hiervoor nemen we de tabel die we opgesteld hebben in Practice 3 en passen deze enigszins aan. Eigenlijk rapporteer je dus over je opgesteld teststrategie. En geef je zo dus een terugkoppeling aan de stakeholders hoe en in hoeverre de gevonden risico's tijdens de risicoanalyse gemitigeerd zijn.

Risicorapportage				
Risico(Testdoel)	**Gevolg**	**Ernst**	**Maatregelen**	**Opmerkingen**
Orders uit Frankrijk worden niet verwerkt	Veel omzetverlies door niet kunnen verkopen van deze	Hoog	Uitgebreid testtraject	Alle maatregelen zoals beschreven in het testtraject zijn uitgevoerd. Hierdoor is dit risico volledig gemitigeerd.

	producten. Veel imagoschade bij afnemers			
Orders uit andere delen van Europa	Gemiddels omezetverlies, Imagoschade.	Gemiddeld		Dit risico is aanzienlijk verkleind door de uitgevoerde kwaliteitsmaatregelen.
Orders uit andere delen van de wereld	Kleine omezetverlies, geringe imagoschade	Laag		Dit risico is zo goed als verdwenen door de genomen maatregelen.
Verkopers API	Verkopers kunnen hun producten niet of niet op de juiste manier aanbieden. Kan leiden tot omzetverlies of fouten in aangeboden producten.	Hoog	Testtraject met verkopers, monitoring van beschikbaarheid na in productiename.	Ondanks een uitgebreid testtraject blijft hier een rest risico open. Dit zit met name in het juiste gebruik van de API. Omdat we dit zo goed mogelijk willen faciliteren naar de verkopers toe willen we monitoring en actief support inrichten. Door onze technische API servicedesk te laten terugbellen en ondersteunen als we in de monitoring veel foutmeldingen signaleren.
…	..			

'Voorbeeld risico rapportage'

Realiseer je dat je rapportage ook per testlevel of testtype kunt insteken. In ons voorbeeld gebruiken we het totale testtraject. Maar je zou dus ook alleen over de code review, de unittest, of acceptatietest kunnen rapporteren. Enkele voorbeelden zie je hieronder:

Code Review		
Regels code gerviewd	Gevonden defects	Openstaande defects
16.268	68	5

'Voorbeeld rapportage code review'

Rapportage Test cases Unit test					
Niet gestart	**In uitvoering**	**Niet van toepassing**	**Gefaald**	**Geslaagd**	**Totaal**
15	26	8	5	145	199

'Voorbeeld rapportage Unit test'

Rapportage Defects									
	Nieuw	**Open**	**Toegewezen**	**Aan het oplossen**	**Klaar voor hertest**	**Hertesten**	**Hertest OK**	**Hertest niet OK**	**Geparkeerd**

Blokkerend	0	1	1	0	2	1	5	0	0
Ernstig	0	1	2	0		1	5	1	1
Niet ernstig	0	0	1	2			4	1	3
Cosmetisch	1	0	0	1			2	0	1
Totaal	0	2	4	3	2	2	16	2	5

'Voorbeeld rapportage Defects'

De rapportages zijn bedoeld als voorbeeld. Je kunt er gebruiken van maken, maar je kunt ook zelf een rapportage maken met gecombineerde onderdelen uit deze rapportage. Je kunt deze rapportage natuurlijk ook visueel maken door het gebruik van staafdiagrammen of door het rapporteren over het verloop van het testtraject en de status wijzigingen per dag of week. Door op het juiste moment (dagelijkse, wekelijks, maandelijks) te rapporteren over de voortgang van je test traject en hoe dit de risico's mitigeert kun je voldoende aan de practice 'Het belang van goede communicatie' en je stakeholders inzage geven in je testtraject.

Definities

Verwachtingsmanagement	Het proces van het verduidelijken of bijstellen van bepaalde verwachte resultaten.
Non-verbale communicatie	De lichaamstaal die we gebruiken als we ons uiten.
Verzender	De afzender van een boodschap of bericht, geschreven dan wel mondeling.
Ontvanger	De ontvanger van boodschap of bericht, geschreven dan wel mondeling.
Verbale communicatie	De woorden die we kiezen om ons uit te drukken.

Hoofdstuk 4: Beveiligbaarheid (Security), Gebruiksvriendelijkheid (Usability) en Performance

LD35	Leer de basis van testen op security. (K2)
LD36	Leer de basis van testen op usability. (K1)
LD37	Leer de basis van testen op performance. (K1)

In dit hoofdstuk bespreken we enkele belangrijke voorbeelden van kwaliteitsattributen. Om een beter begrip te krijgen van wat kwaliteitsattributen zijn, richten we ons op drie kwaliteitsattributen die relevant zijn voor bijna elk project: beveiliging, gebruiksvriendelijkheid en prestaties. Deze kwaliteitsattributen zijn van belang om op te nemen bij het maken van een teststrategie. Het zijn aspecten van kwaliteit waar je in bijna elk softwareproject mee te maken hebt en waar je dus over na zou willen denken bij het opstellen van een teststrategie. In ons gekozen voorbeeld van een webshop die producten verkoopt zullen ook security, usability en performance een rol spelen. Denk bijvoorbeeld aan het risico dat je loopt als de security niet goed ingeregeld hebt, niet geautoriseerde verkopers zouden ten onrechte de API kunnen gebruiken. Voor usability en performance van de webshop zal gelden dat als ze niet op orde zijn, we minder bestellingen van gebruikers zullen ontvangen omdat ze afhaken vanwege een matige interface of trage handelingssnelheid. Er bestaan meer kwaliteitsattributen als de hier genoemde. We kiezen er echter voor om, uitgezonderd functionaliteit en de in hoofdstuk 1 genoemde kwaliteitsattributen deze buiten deze syllabus te laten omdat het anders een te omvangrijk geheel zou worden en we bij de basis willen blijven.

Kwaliteitsattribuut: Beveiliging

Een zeer nuttig hulpmiddel bij het testen van de beveiliging van applicaties is *de OWASP Top Tien* **[XVIII]**. Het OWASP (Open Web Application Security Project) is een stichting zonder winstoogmerk die zich richt op het verbeteren van de beveiliging van software. Om de paar jaar publiceert het OWASP een lijst met de tien meest kritieke beveiligingsrisico's, genaamd ‘The OWASP Top Ten’. De OWASP Top Tien wordt algemeen erkend als de eerste stap voor iedereen die geïnteresseerd is in het verbeteren van de beveiliging van zowel de software die ze ontwikkelen, als de methoden waarmee die de software ontwikkeld wordt. De focus ligt niet alleen op de beveiliging van software die wordt gemaakt, maar ook op de bedrijfsprocessen en best practices op de werkplek. Bijvoorbeeld: het is geen goede gewoonte zijn om wachtwoorden op te schrijven in de buurt van de werkplek.

De OWASP Top Tien bevat veel waardevolle suggesties voor het opnemen van verschillende best practices in test- (of ontwikkelings-)processen. Meer specifieke informatie, inclusief voorbeelden en gedetailleerde informatie over het testen op beveiligingslekken, is te vinden op https://www.owasp.org.

Het toekennen van autorisaties is eenvoudig en stelt gebruikers in staat om veel dingen te doen in een ontwikkel- en testtraject. Vaak wordt dit gedaan vanuit het oogpunt van ‘de gebruiker is

koning'. Om gebroken toegangscontrole te voorkomen, moeten gebruikers toestemming krijgen om te doen wat nodig is om hun taken uit te voeren.

Kwaliteitsattribuut Beveiliging: OWASP Top Tien voorbeeld: Broken Access Control

Broken access control (gebroken toegangscontrole) betekent meestal kwetsbaarheden in rollen en machtigingen. Goed ontworpen toegangscontrole moet worden onderhouden en gecreëerd zodat gebruikers niet kunnen handelen buiten hun bedoelde autorisatie. Als toegangsbeheer niet goed is ontworpen, kan dit leiden tot onbevoegde openbaarmaking, bijwerking of verwijdering van gegevens, of het gebruik van functies die buiten de bedoelde autorisaties van gebruikers vallen. Neem bijvoorbeeld een klein bedrijf waar gebruikers op de afdeling financiële administratie alle rechten krijgen voor financiële administratie. Deze rechten kunnen ertoe leiden dat ze hun salaris kunnen verhogen. Daarom is het belangrijk om te herhalen dat het rollen- en permissiesysteem goed ontworpen, afgestemd en getest moet worden. Een goede vuistregel is om alleen de benodigde rechten toe te kennen, niet meer.

De beheerdersrol zou alleen gebruikt moeten worden wanneer dat nodig is en moet niet met een groot aantal mensen gedeeld worden. Er kan ook een specifiek account worden aangemaakt om andere (mogelijk beheerders-)rechten in onder te brengen. Er dient ook een procedure te worden ingesteld die ervoor zorgt dat rechten kunnen worden ingetrokken als dat nodig is, bijvoorbeeld als een gebruiker het bedrijf verlaat.

Kwaliteitsattribuut Beveiliging: CRUD-matrix

Een goede manier om rollen en rechten te testen (en ook te ontwerpen en op elkaar af te stemmen) is het opstellen van een eenvoudige autorisatiematrix. Deze autorisatiematrix wordt ook wel een **CRUD** (Create, Read, Update, Delete) matrix genoemd. Een autorisatiematrix wordt gemaakt door alle rollen en rechten, gekoppeld aan de functionaliteit die ze geacht worden uit te oefenen, in een tabel te plaatsen.

	Klantgegevens	**Order gegevens**	**Productgegevens**
Manager	CRUD	CRUD	CRUD
Werknemer	CRU	CRU	RU
Klant	RU	CR	

Voorbeeld eenvoudige CRUD-matrix

Een goede manier om dit te testen is door het opstellen van een eenvoudige autorisatie matrix. Ook wel CRUD (Create, Read, Update, Delete) matrix genoemd. Een autorisatiematrix stel je op door alle rollen en rechten van een applicatie in een matrix aan elkaar te koppelen. Hiermee kun je ontbrekende stappen vinden, controleren of gegevens wel of niet (ten onrechte) benaderbaar zijn. Test vervolgens met alle rollen zodat je ook alle rechten kunt verifiëren. Controleer ook wat je niet mag en of je dat echt niet kan.

Kwaliteitsattribuut Beveiliging: OWASP Top Tien voorbeeld: Cryptografische Fouten

Cryptografische fouten zijn fouten die het gevolg zijn van versleuteling. Dit kan te maken hebben met het versleutelingsproces zelf of het ontbreken van de benodigde versleuteling. Deze fouten kunnen leiden tot de blootstelling van gevoelige gegevens in REST of transitie, bijvoorbeeld een wachtwoorddatabase die een eenvoudige, gekraakte of eenrichtingsversleuteling gebruikt om wachtwoorden op te slaan.

Een goed beginpunt bij het oplossen van deze problemen is een goede classificatie van gegevens: dit bepaalt de mate van vertrouwelijkheid van de gegevens. Dit is de reden waarom wachtwoorden, medische gegevens, creditcardnummers, persoonlijke informatie en vertrouwelijke bedrijfsinformatie extra bescherming nodig hebben, vooral als die gegevens vallen onder privacywetten zoals de General Data Protection Regulation (GDPR) van de EU of andere normen.

Na classificatie is de volgende stap om de gegevens goed te beveiligen met moderne, ongecompromitteerde encryptiemethoden. Deze acties zijn van toepassing op zowel interne als externe applicatielinks. Zie de OWASP-website voor meer preventiesuggesties en testtips.

'Gebroken toegangscontrole' en 'Cryptografische fouten' zijn slechts twee van de items uit de OWASP Top Tien voor het veilig maken van je software en werkomgeving. Het is bijna onmogelijk om uit te sluiten dat je gehackt wordt, maar het doel moet zijn om dat zoveel mogelijk te voorkomen. De uitdaging is om hacking snel te detecteren en ervoor te zorgen dat je toegang houdt tot en controle houdt over jouw gegevens. Goede monitoring, back-upprocedures, fallback-procedures en herstelprocedures zijn de sleutel tot het behouden van veilige gegevens. Als deze procedures aanwezig zijn, geoefend en regelmatig getest worden, ben je al goed op weg naar betere beveiligingspraktijken.

Op de website van OWASP staan nog meer tips en trucs voor veilig ontwikkelen en testen: https://www.owasp.org.

Kwaliteitsattribuut: Gebruiksvriendelijkheid

Gebruiksvriendelijkheid verwijst naar de mate waarin een product of systeem kan worden gebruikt door specifieke gebruikers om effectief, efficiënt en succesvol specifieke doelen te bereiken in een specifieke gebruikscontext.

Als we kijken naar een website, app of on-premise systeem, is het hoofdscherm vaak de belangrijkste pagina als het gaat om gebruiksvriendelijkheid. Het is niet alleen het meest bezochte onderdeel van de software, het bepaalt ook of de gebruiker de software kan (of zelfs wil) blijven gebruiken om toegang te krijgen tot alle andere onderdelen. Dit is waar gebruikers hun eerste mening over de software vormen, dus herkenning van geschiktheid is uiterst belangrijk. Zodra aan deze behoefte van de gebruiker is voldaan, zal de gebruiker proberen zijn doel(en) te bereiken door het systeem te gaan gebruiken. Om de tevredenheid van de

gebruiker te maximaliseren, moet de invoer duidelijk zijn en moeten de belangrijkste taken worden weergegeven. Het hoofdscherm is een ideale plek om je te onderscheiden van de concurrentie.

Voor een optimale gebruikersvriendelijkheid zijn een duidelijke navigatie en informatiearchitectuur essentieel. Belangrijk hierbij is om je te concentreren op de taak zelf. Je kunt je op de taak oriënteren als je de behoefte van een specifieke gebruiker kent. Als je bijvoorbeeld toegang tot iets krijgt, moet de navigatie duidelijk zijn. Gebruikers willen weten waar ze zijn, wat ze kunnen doen en waar ze naartoe kunnen. Je wilt deze informatie ook op de meest herkenbare manier tonen. Elementen voor een website zijn bijvoorbeeld de 'home'-link, een hamburgermenu, logische categorisatie (broodkruimels), een sitemap, enz.

Het is belangrijk om het mentale model van de applicatieontwerper af te stemmen op het mentale model van de gebruiker. Om hier rekening mee te houden, moet je bedenken wat de belangrijkste focus van de gebruiker is: dit moet de belangrijkste CTA (Call To Action) zijn. Een CTA is de belangrijkste knop om een doel te bereiken. Dit doel is meestal wat de bezoeker nastreeft, en ditzelfde doel komt meestal het bedrijf ten goede omdat het bijdraagt aan het bereiken van hun bedrijfsdoelen. Zorg er daarom voor dat je duidelijk aangeeft wat de belangrijkste taken zijn.

In een formulier moet het duidelijk zijn waar iets ingevuld moet worden en wat er ingevuld moet worden. Er moet direct feedback worden gegeven voor zowel juiste als onjuiste invoer. Bedenk hoe je weergeeft of iets verplicht is of in welk formaat een telefoonnummer of postcode moet worden ingevuld. Als er iets fout gaat, moet er directe duidelijke feedback worden gegeven aan de gebruiker om de fout op te kunnen lossen.

Vanuit gebruiksvriendelijkheid gezien, moet de belangrijkste focus altijd liggen op de doelen van de gebruiker en de algemene doelen van de klant zelf. In het beste geval overlappen deze doelen elkaar en moet de software zo soepel mogelijk werken om aan deze doelen te voldoen.

Kwaliteitsattribuut: Performance

Performancetesten kunnen worden onderverdeeld in 2 verschillende soorten: loadtesten en stresstesten. Bij loadtesten wordt een bepaalde belasting van gebruikers gesimuleerd en wordt gekeken naar de responstijden onder deze belasting. Bij stresstests ga je opzoek naar de grens van wat het systeem aankan in termen van piekbelasting.

Een goede performancetest vereist specifieke kennis en tools: gedetailleerde kennis van de infrastructuur en hoe deze zich op de juiste schaal verhoudt tot de infrastructuur van de productieomgeving. Voor een goede performancetest is het cruciaal om een omgeving te hebben die vergelijkbaar of bijna identiek is aan de productieomgeving. Infrastructurele componenten zoals switches, hardware en firewalls en hun dimensionering moeten in overweging worden genomen om de systeembelasting nauwkeurig te simuleren. Om deze

belasting te bereiken zijn vaak specifieke tools nodig om een groot aantal gelijktijdige gebruikers te simuleren.

Een andere manier om een goede indicatie van de performance te krijgen is door een gebruikersbelastingprofiel te maken van de toekomstige omgeving. Dit kan een beter inzicht geven in de belasting van het systeem en op welke momenten van de dag. Daarnaast is het opzetten van een goede monitoring, inclusief timestamps, een mitigerende maatregel bij performancetesten.

Definities

Call to Action	De CTA is de belangrijkste actie waarvan je wilt dat de bezoeker die uitvoert.
Loadtest	Een type performancetest uitgevoerd om het gedrag van een component of systeem onder verschillende belastingen te evalueren, meestal tussen verwachte omstandigheden van laag, normaal en piekgebruik.
Hoofdscherm	Het belangrijkste scherm van een applicatie van waaruit acties ondernomen kunnen worden. Dit is meestal het eerste scherm na inloggen in een applicatie.
Volmaaktheid gebruikersinteractie	De mate waarin een gebruikersinterface het de gebruiker mogelijk maakt om een plezierige en voldoening gevende interactie te hebben.
Performance	De snelheid waarmee het informatiesysteem transacties afhandelt.
Herkenbaarheid van geschiktheid	De mate waarin gebruikers kunnen herkennen of een product of systeem geschikt is voor hun behoeften.
Beveiligbaarheid (Security)	De zekerheid dat raadpleging of mutatie van de gegevens uitsluitend mogelijk is door de personen die daartoe bevoegd zijn.
Stresstest	Een vorm van performancetest die erop gericht is om een component of systeem te evalueren op of over de grenzen van de daarvoor verwachte of gespecificeerde werkbelasting, of met beperkte beschikbaarheid van middelen zoals geheugen of servercapaciteit.
Gebruiksvriendelijkheid (Usability)	De mate waarin een product of systeem effectief, efficiënt en naar tevredenheid gebruikt kan worden door gebruikers.

Referenties

Referentie	Bron
[I]	**Boek:** L. Anderson, P. W. Airasian, and D. R. Krathwohl, A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, 2001, Allyn & Bacon, ISBN 03-21084-05-5
[II]	**Website:** ISO 9000:2015 Standard information: https://www.iso.org/standard/45481.html
[III]	**Website:** American Society for Quality glossary: https://asq.org/quality-resources/quality-glossary/
[IV]	**Website:** ISO 25010 Standard information: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[V]	**Website:** ISTQB® glossary application: https://glossary.istqb.org/
[VI]	**Website:** Wikipedia page for Vilfredo Pareto: https://en.wikipedia.org/wiki/Vilfredo_Pareto
[VII]	**Artikel:** Doran, G. T. (1981). "There's a S.M.A.R.T. Way to Write Management's Goals and Objectives", Management Review, Vol. 70, Issue 11, pp. 35-36.
[VIII]	**Artikel:** Jan Vanthienen, "The History of Modeling Decisions using Tables (Part 1)" Business Rules Journal, Vol. 13, No. 2, (Feb. 2012): https://www.brcommunity.com/articles.php?id=b637
[IX]	**Boek:** Boehm, B. (1981) Software Engineering Economics, Prentice-Hall Inc., ISBN 01-38221-22-7
[X]	**Website:** Agile Alliance glossary: https://www.agilealliance.org/glossary/user-story-template/
[XI]	**Website:** Wikipedia page for Flowchart: https://en.wikipedia.org/wiki/Flowchart
[XII]	**Boek:** Pol. M., Teunissen. R., Veenendaal van. E., Testen volgens TMap 2e druk, p326, ISBN 90-72194-58-6
[XIII]	**Boek:** Beizer, B. (1990) Software Testing Techniques. 2nd Edition, Van Nostrand Reinhold, New York. ISBN 18-50328-80-3
[XIV]	**Boek:** Burnstein, Ilene (2003), Practical Software Testing, Springer-Verlag, ISBN 0-387-95131-8
[XV]	**Boek:** G. Myers, The Art of Software Testing; John Wiley, New York, 1979. ISBN 0-471-04328-1
[XVI]	**Website:** ISTQB Foundation syllabus, 2019 version, p61: https://www.istqb.org/certifications/certified-tester-foundation-level
[XVII]	**Website:** Software Testing Help regarding for Pairwise Testing: https://www.softwaretestinghelp.com/what-is-pairwise-testing/
[XVIII]	**Website:** OWASP Top10: https://owasp.org/Top10/
[XIX]	**Artikel:** The New Religion of Risk Management, Bernstein, 1996: https://hbr.org/1996/03/the-new-religion-of-risk-management
[XX]	**Website:** ISO29119-4:https://www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:29119:-4:ed-2:v1:en
[XXI]	**Website:** TMap Glossary Online: https://www.tmap.net/page/tmap-glossary-online