AGILE TESTING CONDENSED

Nederlandse vertaling

A Brief Introduction by

Janet Gregory & Lisa Crispin

Vertaald onder begeleiding van

Yves Hanoulle

Agile Testing Condensed Nederlands

YvesHanoulle, Janet Gregory en Lisa Crispin

Dit boek is te koop bij http://leanpub.com/agiletestingcondensed-nl

Deze versie is gepubliceerd op 2022-12-23

Dit is een Leanpub boek. Leanpub stelt auteurs en uitgevers in staat om volgens het Lean uitgeefproces te werken. Lean Publishing is het uitgeven van een boek dat nog onderhanden is met lichtgewicht gereedschap en vele iteraties om feedback te krijgen van de lezers. Op deze manier kun je aanpassingen maken tot je het juiste boek hebt, en als je zover bent helpt het om te zorgen dat je een positie krijgt in de markt.

Tweet over dit boek!

Gelieve YvesHanoulle, Janet Gregory en Lisa Crispin te helpen door reclame te maken over het boek Twitter!

De voorgestelde hashtag voor dit boek is #AgileTestingCondensed.

Lees wat andere mensen over het boek zeggen door op deze link te klikken en op Twitter naar deze hashtag te zoeken:

#AgileTestingCondensed

Inhoudsopgave

Dankbetuigingen

Onze boeken zijn altijd een inspanning van de gemeenschap. Veel dank aan onze collega's die hun visie hebben gegeven over hoe de rol van tester zich evolueert in hoofdstuk 11. We gaan hier voor een beknopt boek, dus we kunnen niet alle mensen afzonderlijk bedanken van wie we steeds nieuwe ideeën en concepten leren, deze leren toe te passen in ons eigen werk en leren om deze verder uit te dragen. Maar weet dat we je dankbaar zijn.

Ook dank aan de mensen die ons manuscript hebben gelezen en nuttige feedback gaven, waaronder: Mike Talks, Nikola Sporczyk, Lena Pejgan Wiberg, Pascal Stiefel, Barbara Zaleska, Bertold Kolics, Carol Vaage en onze redacteur Erica Hunter. Daarnaast zijn we Constance Hermit heel dankbaar dat ze ons haar wondermooie tekening voor hoofdstuk 12 liet gebruiken, en Jenn Sinclair waarvan we vaak haar lijntekeningen mochten gebruiken.

Heel veel dank aan José Diaz en Johanna Rothman voor hun mooie voorwoorden.

Tenslotte waarderen we oprecht onze echtgenoten, Jack Gregory en Bob Downing, die er altijd zijn met een goed glas wijn als we het nodig hebben.

Acknowledgments

Our books always involve a community effort. Many thanks to our colleagues who contributed their visions of how the tester role is evolving in Chapter 11. We are going for a short book here, so we canâ€™t individually thank all the people from whom we keep learning new ideas and concepts to apply in our own work and transfer to more people, but please know that we are grateful.

More gratitude to the people who reviewed our draft manuscript and gave us such helpful feedback, including Mike Talks, Nikola Sporczyk, Lena Pejgan Wiberg, Pascal Stiefel, Barbara Zaleska, Bertold Kolics, Carol Vaage, and our copy editor Erica Hunter. Thanks so much to Constance Hermit for letting us use her wonderful drawings in Chapter 12, and Jenn Sinclair for her line drawings that we use so much.

A special thanks to JosÃ© Diaz and Johanna Rothman for their lovely forewords.

And last but not least, we truly do appreciate our husbands, Jack Gregory and Bob Downing, who are always there with a glass of wine when we need one.

Voorwoord door José Díaz

CEO, Trendig.com

Het leven leidt je naar je doel. Wie had gedact in 2009, toen ik de eerste *Agile Testing Days* (ATD) startte, dat ik het voorwoord zou schrijven van een boek van de koninginnen van het agile testen, ook al behoren Janet en Lisa al van toen tot het ATD gezelschap. Ik ben heel blij en vereerd dat ze het mij gevraagd hebben.

Agile Testing Condensed is in lijn met de twee vorige boeken van Janet en Lisa. Deze keer verlichten ze de hoeken en verkennen ze de randen van testen en kwaliteit in agile projecten - kort, beknopt, raak en recht uit het hart van twee vrouwen die hun carrière en leven hebben gewijd aan het delen van hun kennis en het professionaliseren van agile testen. Het bevat inzichtelijke voorbeelden en anekdotes en is leuk om te lezen. Beide auteurs delen hun praktijkervaringen en nodigen de lezers uit om hun reis langs talrijke projecten te reconstrueren. Ze nemen je bij de hand voor een wandeling door deze uitdagende en prachtige wereld van agile testen en leggen het je uit. Het is een schatkist, voor jou en je team.

Ik raad niet alleen dit inspirerende boek aan, maar ook hun levensechte training “Agile Testing for the Whole Team.”

Het boek is de kroon op het werk van de laatste 20 jaar, waarin ze zich intensief hebben bezig gehouden met de agile wereld en haar gemeenschap. Hun ontelbare projecten, trainingen, workshops, keynotes, praatjes, webinars, lean coffees, coaching, open spaces, mentor sessies en gesprekken komen samen in dit boek en maken de inhoud zo prakisch en waardevol voor iedereen die een reis naar de wereld van agile overweegt (inclusief managers).

Het is gemaakt voor jou en mij. Met veel liefde! Laten we er van genieten!

Foreword by José Díaz

CEO, Trendig.com

Life leads you to your goal. Who would have thought back in 2009 when I started the first Agile Testing Days (ATD) that I would write the foreword to a book by the Agile Testing Queens, even though Janet and Lisa have been part of the ATD ensemble ever since. I'm very happy and honored that they asked me to do so.

Agile Testing Condensed is in line with the two previous books of Janet and Lisa. This time they shed light into the corners and explore the edges of testing and quality in agile projects – short, concise, spot-on and right from the hearts of two women who dedicated their careers and life to share their knowledge and advance the activity of testing into an agile profession. It contains insightful examples and anecdotes and is fun to read. Both authors share their practical experiences and invite readers to retrace their journey through numerous projects. They take you by the hand for a walk through this challenging and beautiful world of agile testing and explain it to you. It is a treasure chest, for you and your team.

I recommend not only this inspiring book but also their true-to-work life training "Agile Testing for the Whole Team."

The book crowns their work of the last 20 years, in which they have dealt intensively with the agile world and its community. Their countless projects, trainings, workshops, keynotes, talks, webinars, lean coffees, coaching, open spaces, mentoring sessions, and conversations are condensed into this book and make its content so practical and worthwhile. This book is a must read for anyone considering a journey into the world of agile (including managers).

It is made for you and me. With a lot of love! Let's enjoy it!

Voorwoord door Johanna Rothman

Auteur van *Create Your Successful Agile Project.*

Misschien heb je de andere boeken van Janet en Lisa over agile testen al gelezen. Die boeken gaan over veel meer dan alleen agile testen. Ik raad je ten zeerste aan om ze te lezen.

Dit boek, *Agile Testing Condensed*, richt zich op het creëren van een omgeving waarin het team - en vooral de testers - kunnen slagen. Het boek biedt een overvloed aan bronnen in boeken, artikelen en blogberichten om dat idee te versterken en jou, de lezer, manieren bieden om over het onderwerp na te denken.

Elk hoofdstuk bevat juweeltjes - dat is het beknopte deel - die kunnen helpen richting te geven aan hoe je denkt over testen en kwaliteit.

Bijvoorbeeld, te veel teams denken dat testen gaat over hoe een tester een specifieke functionaliteit test. In plaats daarvan is er een sectie die iedereen helpt na te denken over het product en hoe de testinspanning over verschillende niveaus van het product kan worden gepland.

Een ander voorbeeld gaat over hoe ze ons herinneren aan samenwerking. Van pairing om te verkennen, tot het werken in triades om stories te definiëren: testers werken samen.

Ik hield van de paragraaf “Testers zijn kwaliteitslijm voor een team”. Zo waar, en te weinig testers en teams maken gebruik van die lijm.

Als je je afvraagt hoe je een agile tester moet zijn, of als je niet zeker weet of je testers nodig hebt in je agile team, lees dan dit boek - een kort en snel te lezen boek dat voor lange tijd zijn vruchten zal afwerpen.

Foreword by Johanna Rothman

Author of *Create Your Successful Agile Project*

You may have read Janet and Lisa's other books about agile testing. Those books are much more than just agile testing. I strongly recommend you read them.

This book, *Agile Testing Condensed,* focuses on how to create an environment in which the team – and especially the testers – can succeed. The book offers plenty of sources in books, articles, and blog posts that help reinforce the idea and offer you, the reader, ways to think about the issue.

Each chapter contains gems – that's the condensed part – that can help guide your thinking about testing and quality.

For example, too many teams think testing is about how a tester tests a specific feature. Instead, there's a section that helps everyone think about the product and how to plan the testing effort across the various levels of the product.

Another example is how they remind us about collaboration. From pairing to explore, to working in triads to define stories, testers collaborate.

I loved the section, "Testers are quality glue for a team." So true, and too few testers and teams take advantage of that glue.

If you're wondering about how to be an agile tester, or if you're not sure if you need testers on your agile team, read this book – a short and quick read that will pay dividends for a long time.

Waarom dit boek?

Ons doel was om een klein, beknopt en gemakkelijk te lezen boek te maken dat iedereen zou kunnen vastpakken om de basis te begrijpen van hoe je slaagt met testen en hoe je een kwaliteitscultuur opbouwt in een agile context. Onze eerste twee (veel grotere) boeken gaan daar dieper op in en bevatten waargebeurde verhalen van beoefenaars. We noemen die boeken:

- *Agile Testing*, oftewel *Agile Testing: A Practical Guide for Testers and Agile Teams*, 2009
- *More Agile Testing*, oftewel *More Agile Testing: Learning Journeys for the Whole Team*, 2014

Dit boek is geen inleidend boek over testen. Er zijn veel geweldige bronnen beschikbaar om de basisprincipes van testen, testautomatisering, DevOps en andere onderwerpen te leren. In onze literatuurlijst vindt je enkele goede links. Ook is dit geen basisinleiding tot agile ontwikkeling. Het is bedoeld voor lezers die in teams zitten die agile werken of die willen weten hoe testen en kwaliteit passen in agile ontwikkeling en op zoek zijn naar begeleiding, zoals managers of leidinggevenden.

Hoe dit boek te lezen

Je kan in elke sectie beginnen of afzonderlijke hoofdstukken lezen op basis van wat je wilt leren. Elk hoofdstuk is onafhankelijk, hoewel we naar andere hoofdstukken kunnen verwijzen.

Gebruik het als een zakboekje voor agile testen, om bij de hand te houden terwijl je werkt. Als je vastloopt en inspiratie nodig hebt, of als je team twijfelt bij een testuitdaging, blader dan door dit boek voor ideeën. Als je meer diepgang wilt, bekijk dan onze andere boeken.

Door het hele boek heen vind je hints die je op ideeën brengen over hoe je een specifiek probleem aanpakt.

Why this book?

Our goal with this book was to create a small, concise, easy-to-read book that anyone could pick up to get a basic understanding on how to succeed with testing and build a quality culture in an agile context. Our first two (much larger) books go into more depth and feature real-life stories from practitioners. We refer to those books as:

- *Agile Testing,* which is *Agile Testing: A Practical Guide for Testers and Agile Teams,* 2009
- *More Agile Testing,* which is *More Agile Testing: Learning Journeys for the Whole Team,* 2014

This book is not an introductory testing book. There are many great resources available to learn the basics of testing, test automation, DevOps, and other topics. You can find some good links in our bibliography. Similarly, this is not a basic introduction to agile development. It is for readers who are on teams adopting agile or those who want to know how testing and quality fits into agile development and are looking for guidance, such as managers or executives.

How to read this book

You are welcome to start in any section, or read individual chapters based on what you want to learn. Each chapter is self-sufficient, although we may refer to other chapters.

Use it as a pocket guide to agile testing to keep handy as you work. When you get stuck and need inspiration, or when your team is wondering about a testing challenge, look through this book for ideas. When you want to get more in-depth, check out our other books.

Throughout the book, youâ€™ll find hints that will give you ideas about how to approach a specific problem.

DEEL 1: Fundamenten

In dit eerste deel geven wij onze definitie van “agile testen”. Het hart van agile testen betrekt het hele team bij het testen en inbouwen van kwaliteit in ons product. Hierbij is er een grote mentaliteitsverandering waar het team leert om defecten te voorkomen in plaats van te vertrouwen op testers als een vangnet om ze op te vangen.

Nieuwe agile teams leren om grote functionaliteiten in kleine, incrementele stukjes te hakken en regelmatig kleine veranderingen op te leveren. Tegelijkertijd moeten ze het grote geheel voor ogen houden om klanttevredenheid te behouden.

SECTION 1: Foundations

In this first section, we share our definition of "agile testing." The heart of agile testing involves the whole team in testing and building quality into our product. There's a big mindset shift as the team learns to prevent defects rather than relying on testers as a safety net to catch them.

New agile teams learn to slice big features into small, incremental chunks and deliver small changes frequently. At the same time, they must keep the big picture in mind to maintain customer happiness.

Hoofdstuk 1: Wat bedoelen we met agile testen?

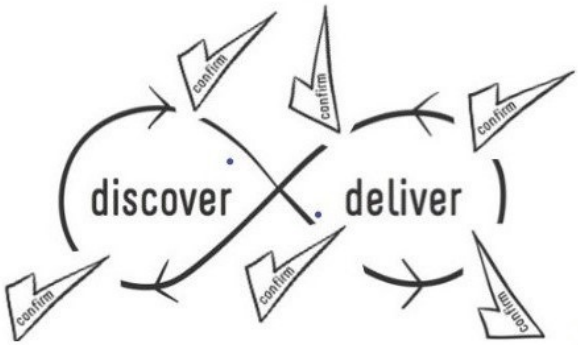
In de loop der jaren is ons vaak gevraagd hoe we 'agile testen' moeten definiëren. We hebben onze 'definitie' van agile testen in het laatste deel van dit hoofdstuk opgenomen, maar er zijn veel factoren die bij deel uitmaken van die definitie. Enkele praktijken die teams helpen bij hun reis naar succes zijn:

- Het inbouwen van kwaliteit: teams richten zich op het voorkomen van misverstanden over het gedrag van functionaliteiten en het voorkomen van defecten in de code
- Ontwikkeling sturen met concrete voorbeelden: gebruik van praktijken als *acceptance test-driven development* (ATDD), *behavior-driven development* (BDD) of *specification by example* (SBE)
- Testactiviteiten voorzien zoals het voeren van gesprekken om gedeeld begrip op te bouwen; vragen stellen om ideeën en aannames te testen; testen automatiseren; het uitvoeren van onderzoekende testen; testen op kwaliteitskenmerken zoals prestatie, betrouwbaarheid en beveiliging; en leren van gebruik in productie
- Het gebruiken van retrospectieven met het hele team en kleine experimenten om het testen en de kwaliteit voortdurend te verbeteren en te ontdekken wat in hun context werkt

We zullen de bovenstaande praktijken doorheen dit boek verder uitwerken. We beschouwen ze niet als '*best practices*' omdat we weten dat ze voortdurend evolueren.

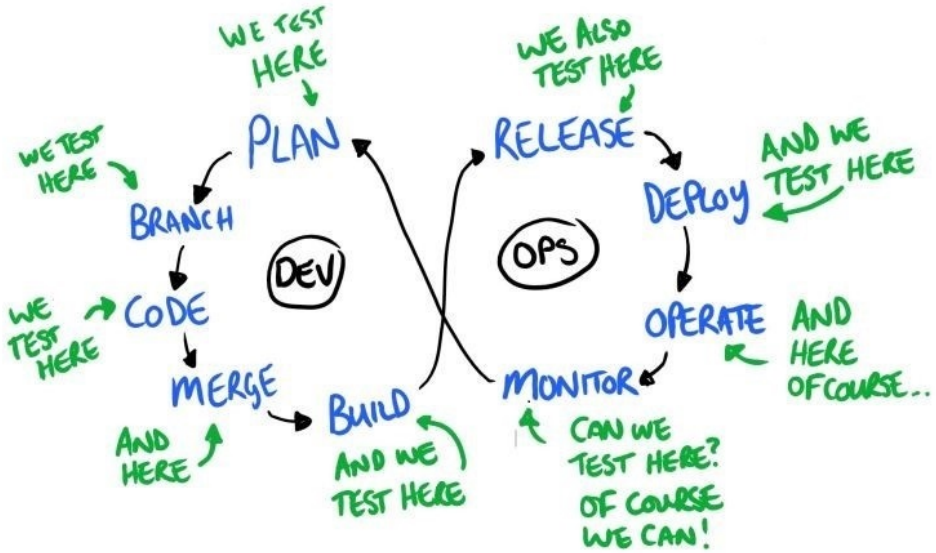
Continue testmodellen

Testen is een integraal onderdeel van softwareontwikkeling, samen met codering, operations, inzicht krijgen in de behoeften van de klant, en meer. We houden van de modellen die continue of holistische testmethodes vertegenwoordigen. Een van onze favorieten is de methode die Ellen Gottesdiener en Mary Gorman gebruiken in hun boek *Discover to Deliver*, 2012. In figuur 1.1 wordt weergegeven dat het ontwikkelingsproces een oneindige lus is, voortdurend bevestigend – en dat is hoe we eigenlijk software ontwikkelen. We leren over een functionaliteit die onze klanten willen, we bouwen het en leveren het op, om dan te leren hoe de klanten het daadwerkelijk gebruiken. We gebruiken die feedback om te beslissen wat we vervolgens gaan bouwen (of verwijderen).



Figuur 1.1: "Discover to deliver" lus © 2015 by EBG Consulting

Dan Ashby nam een vergelijkbare benadering met onderstaand diagram (Figuur 1.2) in zijn blogpost als hij het heeft over testen in DevOps[1]. Dit model laat zien dat testen een integraal onderdeel is van DevOps. In hoofdstuk 8 gaan we hier dieper op in.



Figuur 1.2: Continu Testen in DevOps lus

We houden van termen als continu testen of holistisch testen en erkennen dat elk team ze moet aanpassen aan hun eigen unieke context.

[1] https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/

Tien principes voor agile testen

In *Agile Testing* introduceerden we het idee van 10 Principes voor Agile Testers. We realiseren ons nu dat deze principes niet alleen voor testers zijn, maar voor iedereen in het team. We schreven deze principes in een tijd dat de meeste testers nog deel uitmaakten van een testteam in een gefaseerd project met *gates*, en het testen gebeurde aan het einde - nadat alle codering "af" was.

Deze principes zijn vandaag nog steeds van toepassing op iedereen in een agile team dat een product van de hoogste mogelijke kwaliteit wil leveren.

- Zorg voor continue feedback.
- Lever waarde aan de klant.
- Maak *face-to-face* communicatie mogelijk.
- Wees moedig.
- Hou het simpel.
- Verbeter jezelf voortdurend.
- Reageer op verandering.
- Organiseer jezelf.
- Focus op mensen.
- Geniet!

Testmanifest

We sluiten dit hoofdstuk af met dit "testmanifest[2]" gemaakt door Karen Greaves en Samantha Laing. Hun manifest weerspiegelt de mentaliteitsverandering die nodig is voor een succesvolle agile testaanpak. We testen gedurende het hele ontwikkelingsproces, we richten ons op het voorkomen van bugs, we testen veel meer dan functionaliteit en het hele team neemt de verantwoordelijkheid voor de kwaliteit. Je vindt deze principes terug in al onze boeken. Telkens wanneer je team vastzit met een kwaliteits- of testprobleem, moet je nadenken over deze principes om een manier te vinden om vooruit te komen. (Figuur 1.3).

[2] http://www.growingagile.co.nz/2015/04/the-testing-manifest/

Figuur 1.3: Het testmanifest

Definitie van Agile Testen

De eenvoudigste definitie die we hebben bedacht voor wat we bedoelen met agile testen is de volgende:

> *Collaboratieve testpraktijken die continu plaatsvinden, van bij de aanvang tot oplevering en daarna, ter ondersteuning van frequente oplevering van waarde voor onze klanten. Testactiviteiten zijn gericht op het inbouwen van kwaliteit in het product, waarbij snelle feedbacklussen worden gebruikt om ons begrip te valideren. De praktijken versterken en ondersteunen het idee van verantwoordelijkheid voor de kwaliteit van het hele team.*

Het duurt even voordat het verwerkt is en je kunt onze blogpost[3] bekijken voor meer details.

[3] https://agiletester.ca/ever-evolving-never-set-stone-definition-agile-testing/

Chapter 1: What Do We Mean by Agile Testing?

Over the years we've been asked many times how to define "agile testing." We have included our "definition" of agile testing in the last part of this chapter, but there are many factors that go into that definition. Some of the practices that help support teams in their journey toward success are:

- Building quality in: teams focus on preventing misunderstandings about feature behavior as well as preventing defects in the code
- Guiding development with concrete examples: using practices like acceptance test-driven development (ATDD), behavior-driven development (BDD), or specification by example (SBE)
- Including testing activities such as having conversations to build shared understanding; asking questions to test ideas and assumptions; automating tests; performing exploratory testing; testing for quality attributes like performance, reliability, and security; and learning from production usage
- Using whole-team retrospectives and small experiments to continually improve testing and quality and find what works in their context

We'll elaborate on the above practices throughout this book. We do not consider them to be "best practices" because we know they are ever evolving.

Continuous testing models

Testing is an integral part of software development, along with coding, operations, understanding customer needs, and more. We like the models that represent a continuous or holistic testing approach. One of our favorites is the approach that Ellen Gottesdiener and Mary Gorman use in their book *Discover to Deliver*, 2012. Shown in Figure 1.1., the development process represents an infinite loop, confirming continuously – which is how we really develop software. We learn about a feature that our customers want, we build and deliver it, and then we learn how the customers actually use it. We use that feedback to decide what to build (or remove) next.

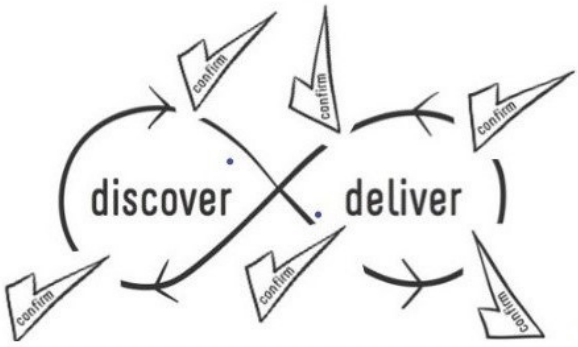


Figure 1.1: Discover to Deliver loop © 2015 by EBG Consulting

Dan Ashby took a similar approach using the diagram below (Figure 1.2) in his blog post as he talks about testing in DevOps[4]. This model shows that testing is an integral part of DevOps. We go into more detail on this subject in Chapter 8.

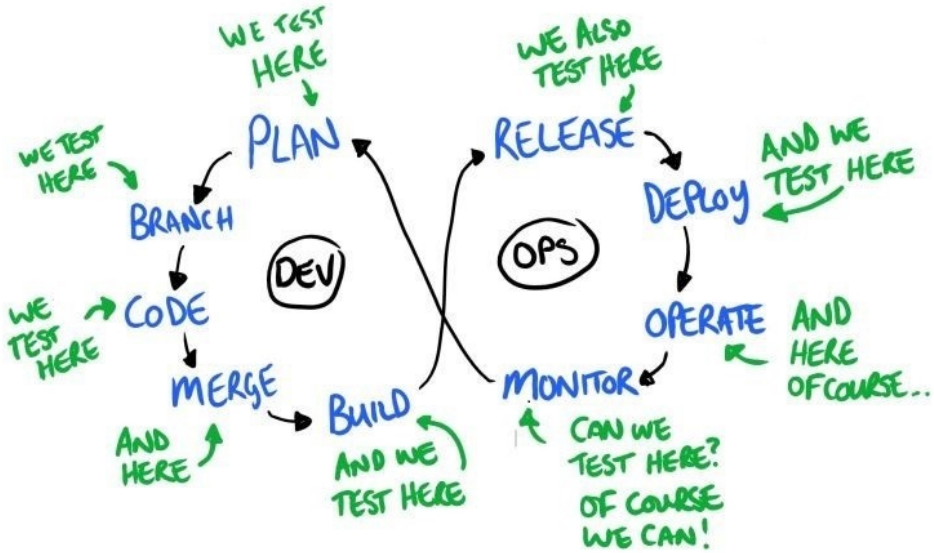


Figure 1.2: Continuous testing in the DevOps loop

We like terms like continuous testing or holistic testing and recognize that each team needs to adapt them for their own unique context.

[4] https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/

Ten principles for agile testing

In *Agile Testing,* we introduced the idea of 10 Principles for Agile Testers. We realize now that these principles are not necessarily just for testers but for anyone in the team. We wrote these principles at a time when most testers were still part of a testing team with a phased and gated project, and the testing was at the end – after all the coding was "finished."

These principles still apply today for anyone on an agile team wanting to deliver the highest-quality product they can.

- Provide continuous feedback.
- Deliver value to the customer.
- Enable face-to-face communication.
- Have courage.
- Keep it simple.
- Practice continuous improvement.
- Respond to change.
- Self-organize.
- Focus on people.
- Enjoy!

Testing manifesto

We'll finish this chapter with this "testing manifesto[5]" created by Karen Greaves and Samantha Laing. Their manifesto reflects the mindset shift needed for a successful agile testing approach. We test throughout the development process, we focus on preventing bugs, we test much more than functionality, and the whole team takes responsibility for quality. You will find these principles reflected throughout all our books. Anytime your team is stuck on a quality or testing problem, reflect on these principles to find a way to move ahead. (Figure 1.3).

[5] http://www.growingagile.co.nz/2015/04/the-testing-manifesto/

Figure 1.3: The testing manifesto

Agile testing definition

The simplest definition that we've come up with for what we mean by agile testing is the following:

> *Collaborative testing practices that occur continuously, from inception to delivery and beyond, supporting frequent delivery of value for our customers. Testing activities focus on building quality into the product, using fast feedback loops to validate our understanding. The practices strengthen and support the idea of whole-team responsibility for quality.*

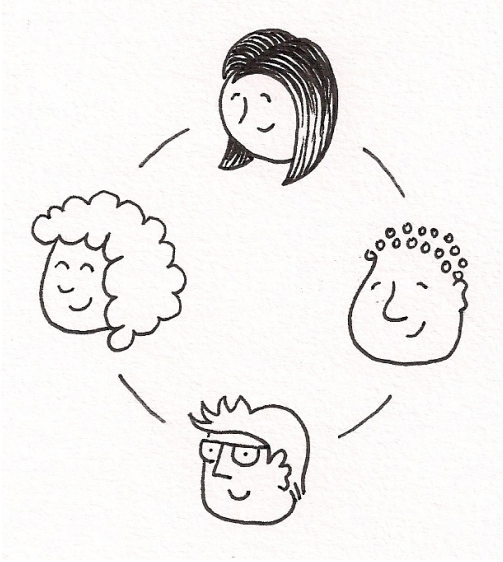
It takes a while to digest, and you can look at our blog post[6] for more details.

[6] https://agiletester.ca/ever-evolving-never-set-stone-definition-agile-testing/

Hoofdstuk 2: Teambrede aanpak en agile testmentaliteit

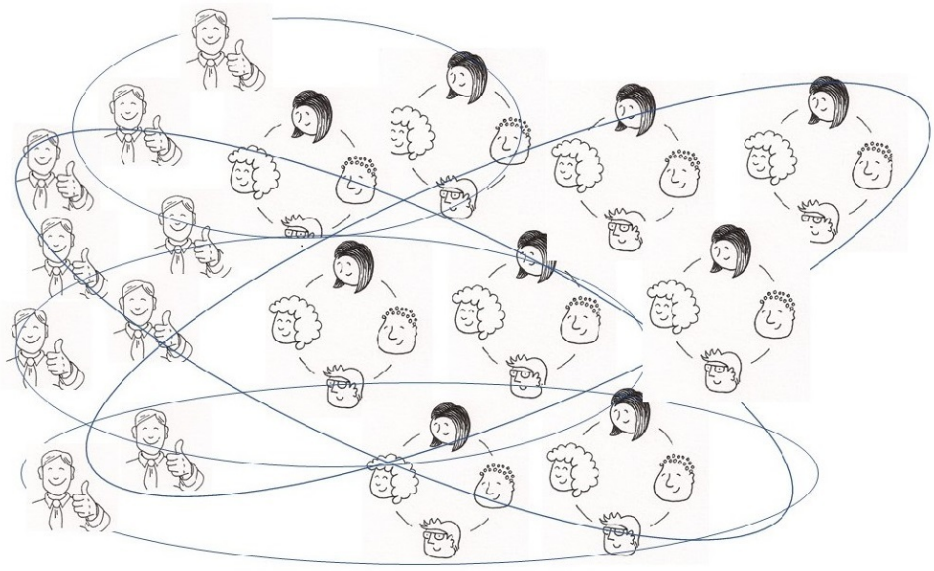
Veel softwareteams gebruiken nog steeds een lineaire benadering, een gefaseerd project met *gates*, voor het leveren van software. Mensen in een bepaalde rol worden ingedeeld in specifieke geïsoleerde teams en het werk wordt overgedragen van het ene team naar het andere. Het test- of QA-team wordt gezien als verantwoordelijk voor het waarborgen van de kwaliteit - meestal helemaal aan het einde van het proces en vlak voor levering aan productie - wanneer het te laat is om nog veel te doen om de kwaliteit te verbeteren.

Bij agile softwareontwikkeling doorbreken we de silo's en maken we van ontwikkeling een continu, iteratief proces. Het hele team werkt samen om kwaliteit in te bouwen gedurende het hele proces. Met het 'hele team' bedoelen we meestal het opleverteam - de mensen die verantwoordelijk zijn voor het begrijpen van wat te bouwen, het bouwen en het opleveren van het eindproduct aan de klant.



Figuur 2.1: Eén enkel team

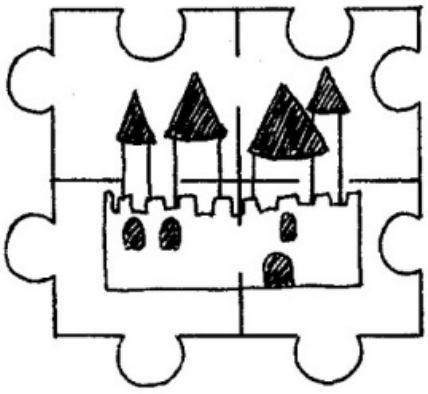
In grotere organisaties, zelfs in organisaties die agile principes en praktijken hebben aangenomen, kan er misschien meer dan één team aan een product werken, zoals een onafhankelijk databaseteam, gebruikers-ervaringsteam of ander productteam. In deze gevallen wordt de definitie van het hele team uitgebreid tot wie je nodig hebt om het product te leveren. De DevOps-beweging heeft het opnemen van *operations* in de oplevering zichtbaarder gemaakt. Janet noemt de mensen buiten het opleverteam een uitgebreide familie.



Figuur 2.2: Meerdere teams

Focus op kwaliteit

De teambrede aanpak houdt in dat alle teamleden verantwoordelijk zijn voor de kwaliteit van hun product. Een deel van deze verantwoordelijkheid is ervoor zorgen dat testtaken worden voltooid samen met de rest van de ontwikkeltaken. Wanneer het doel is om de hoogst mogelijke kwaliteit te leveren, in plaats van sneller te leveren, bouwt het team een solide fundament van technieken. Om dat kwaliteitsniveau te bereiken, managen teams hun werklast zodat ze tijd hebben om kerntechnieken te leren, zoals *test-driven development* (TDD) en *onderzoekende* testen. Ze nemen ook de tijd om het bedrijfsdomein te leren kennen en relaties op te bouwen met bedrijfsexperts om functionaliteiten met de meeste bedrijfswaarde te identificeren en deze vervolgens zo eenvoudig mogelijk te implementeren. Na verloop van tijd, door te focussen op kwaliteit, zijn teams in staat sneller te werken.



Figuur 2.3: Jouw bedrijfswaarde opgeleverd zoals verwacht

Er zijn verschillende gebieden die een verandering vereisen in de manier waarop teamleden ontwikkeling benaderen. Wanneer het hele team verantwoordelijk is voor zowel de productkwaliteit als de proceskwaliteit, moet elk teamlid proactief zijn in het oplossen van problemen. Iedereen in het team kan bijvoorbeeld helpen uitzoeken wat het meest waardevol is voor klanten. Ze werken in kleine stappen om net genoeg van die waarde te leveren en te leren hoe de klant die functionaliteit gebruikt. Door deze snelle feedbacklus te creëren, kan het team het testen concentreren op de functionaliteiten die het meest waardevol zijn voor de klant.

Elk team moet een volwaardige *Definition of Done* (DoD) bespreken en overeenkomen. Die zou ondermeer moeten bevatten hoe het team van plan is om te gaan met fouten die in de code worden gevonden. DoD moet ook testen bevatten en de vraag die moet worden gesteld is: “Welke soorten testen bedoel je?” In hoofdstuk 9 behandelen we de agile testkwadranten en beantwoorden we die vraag. De DoD moet door elk teamlid op dezelfde manier worden begrepen.

Hoe teams omgaan met fouten

Een grote mentaliteitsverandering voor testers is het aannemen van een teamaanpak om fouten te corrigeren. In *More Agile Testing* hebben we het over focussen op het voorkomen van defecten in plaats van ze te vinden nadat het coderen is voltooid. Wanneer het hele team zich concentreert op het inbouwen van kwaliteit in het product, door gebruik te maken van praktijken zoals *acceptance test-driven development* en na te denken over beperkingen rond kwaliteitskenmerken, kan dit een grote bijdrage leveren aan het verminderen van het aantal gevonden fouten in de code.

Veel teams passen een tolerantie van nul fouten toe (*zero defects*). Dit betekent dat nul bekende fouten overblijven na een iteratie of na voltooiing van een story. Om dit te laten werken, moeten teams snel feedback hebben van testactiviteiten, zodat gevonden fouten onmiddellijk kunnen worden verholpen. Eenmaal gevonden, schrijven de programmeurs een of meer uitvoerbare test(s), corrigeren de code zodat de test(s) slagen en voeren indien nodig exploratory testen uit. Het team hoeft er dan niet meer aan te denken, wetende dat ze het probleem hebben verholpen.

Heel vaak helpt deze filosofische verandering in aanpak om een vijandige omgeving te veranderen in een samenwerkende omgeving.

Meerdere perspectieven

Teamleden hebben verschillende gezichtspunten, vaardigheden en perspectieven. We merken dat door alle perspectieven te gebruiken, we een beter begrip hebben van de risico's die verbonden zijn aan het opleveren van een functionaliteit. Zo helpt ontwerpen voor testbaarheid om voorbeelden van gewenst en ongewenst softwaregedrag om te zetten in uitvoerbare testen. Teamleden worden generalistische specialisten - dat wil zeggen, ze kunnen experts zijn op een of twee gebieden, maar ze kunnen op verschillende manieren bijdragen aan de gemeenschappelijke doelen van het team.

Een paar voorbeelden:

- Testers kunnen experts zijn in het testen van het product, maar kunnen bijdragen aan het begrijpen van de functionaliteiten en stories door vragen te stellen om verborgen veronderstellingen te ontdekken.
- Ontwikkelaars kunnen *onderzoekende* testen uitvoeren op hun eigen code voordat ze hun code inchecken.
- *Product owners* voeren acceptatietesten uit op elke story.

In hoofdstuk 11 zullen we wat meer spreken over de rol van een tester en hoe deze kan veranderen voor agile teams.

Chapter 2: Whole-Team Approach and Agile Testing Mindset

Many software teams still use a phased-and-gated, linear approach to delivering software. People in a given role are siloed on specific teams, and they hand work off from one team to the next. The test or QA team is seen as responsible for ensuring quality, usually at the very end of the process and right before delivery to production, when it's too late to do much to improve quality.

In agile development, we break down the siloes and turn development into a continual, iterative process. The whole delivery team works together to build quality in throughout the process. By "whole team," we usually mean the delivery team – the people who are responsible for understanding what to build, building it, and delivering the final product to the customer.

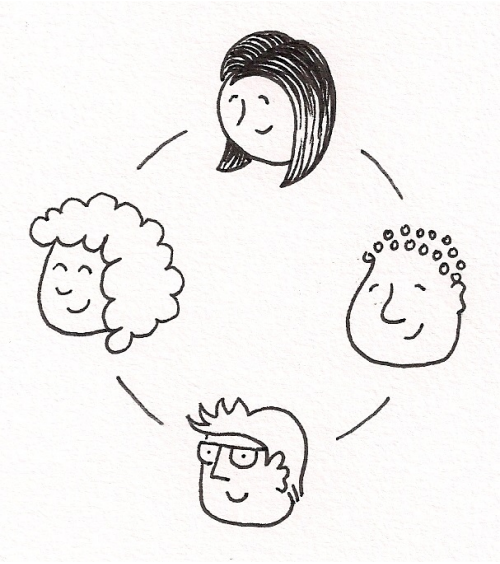


Figure 2.1: A single team

In larger organizations, even those that have adopted agile principles and practices, there may be more than one team working on a product, such as an independent database team, user experience team, or other product team. In these cases, the whole-team definition extends to mean whoever you need to deliver the product. The DevOps movement has made the inclusion of operations in the delivery more visible. Janet refers to the people outside the delivery team as an extended family.

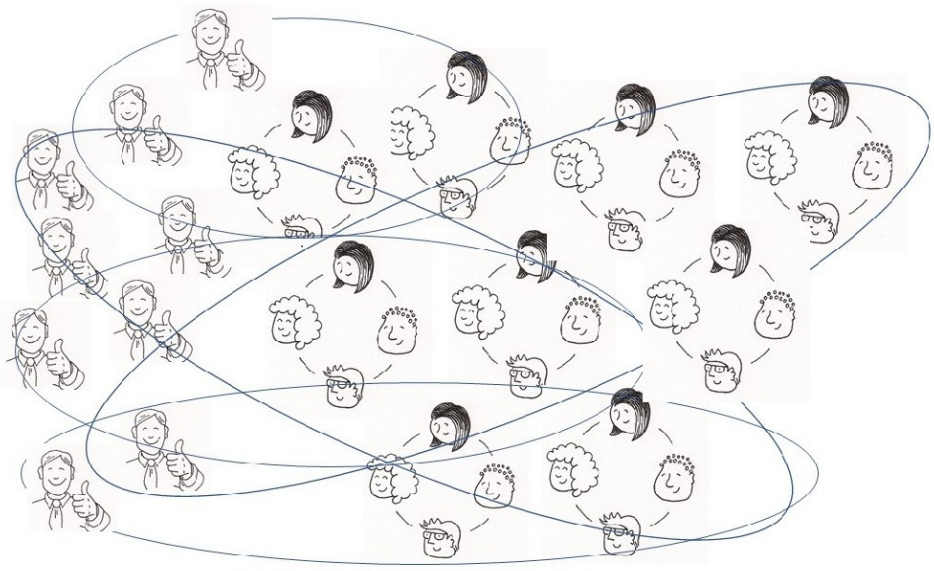


Figure 2.2: Multiple teams

Focus on quality

The whole-team approach means that all team members are responsible for the quality of their product. Part of this responsibility is ensuring that testing tasks are completed alongside the rest of the development tasks. When the goal is to deliver the highest quality possible, rather than deliver faster, the team builds a solid foundation of practices. To achieve that quality level, teams manage their workload so that they have time to learn core practices such as test-driven development (TDD) and exploratory testing. They also take time to learn the business domain and build relationships with business experts to identify features with the most business value and then implement them as simply as possible. Over time, by focusing on quality, teams do begin to be able to work faster.

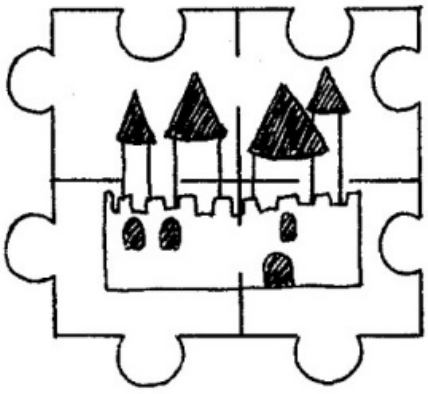


Figure 2.3: Your business value delivered as expected

There are several areas that require a change in how team members approach development. When the whole team is responsible for quality of the product as well as quality of the process, each team member needs to be proactive in solving problems. For example, everyone on the team can help figure out what is most valuable to customers. They work to deliver just enough of that value in small increments to learn how the customer uses that capability. By creating these quick feedback loops, the team can focus their testing on the features that are most valuable to the customer.

Each team needs to discuss and agree on a "valuable Definition of Done" (DoD). That should include how the team plans to deal with defects found in the code. DoD must include testing, and the question that needs to be asked is, "What types of testing do you mean?" In Chapter 9, we cover the agile testing quadrants and answer that question. DoD needs to be understood in the same way by every team member.

How teams deal with defects

One big mindset shift for testers is adopting a team approach to fixing defects. In *More Agile Testing,* we talk about focusing on preventing defects rather than finding them after coding is complete. When the whole team concentrates on building quality into the product by using practices like acceptance test-driven development and thinking about constraints around quality attributes, it can go a long way to reducing the number of defects found in the code.

Many teams practice zero defect tolerance. This means zero known defects escape an iteration or story completion. To make this work, teams must have fast feedback from testing activities so that any defects found can be fixed immediately. Once found, the programmers write one or more executable test(s), correct the code so the test(s) passes, and perform exploratory testing if needed. The team can then forget about it, knowing that they have corrected the issue.

Quite often, this philosophical change in approach helps to change an adversarial environment to a cooperative environment.

Multiple perspectives

Team members have different viewpoints, skill sets, and perspectives. We find that by using all perspectives, we have a better understanding of risks involved when delivering a feature. For example, designing for testability helps turn examples of desired and undesired software behavior into executable tests. Team members become generalized specialists – that is, they may be experts in one or two areas but are able to contribute to the team's common goals in a variety of ways.

Some examples:

- Testers may be experts at testing the product but can contribute to understanding the features and stories by asking questions to uncover hidden assumptions.
- Programmers can perform exploratory testing on their own code before checking in their code.
- Product owners execute acceptance tests on every story.

In Chapter 11, we'll talk a bit more about a tester's role and how it may change for agile teams.

Hoofdstuk 3: Testplanning in agile contexten

Eén van onze top zeven succesfactoren van *Agile Testing* is “Vergeet Niet Het Grote Geheel”. Teams raken verstrikt in het bouwen, testen, en opleveren van kleine delen - wat we aanmoedigen - en vergeten hoe dit past in het systeem of hoe dit kan bijdragen aan het oplossen van het bedrijfsprobleem.

Om testactiviteiten effectiever in te plannen, moet het team rekening houden met de context. Denk aan de volgende 3 aspecten om jouw context te begrijpen: het team, het product, en de detailniveaus van jouw systeem.

Team

Niet alle teams zijn hetzelfde. Werk je in een klein team samen op kantoor, dan is de situatie ideaal voor een vlotte communicatie. Je kan goed elkaars waarden leren kennen en ze met elkaar delen. Het is een leuke plek om een fantastisch product op te leveren, en het maakt plannen een stuk eenvoudiger. Teams begrijpen goed de volgende functionaliteit en kunnen ze gemakkelijk uitdiepen tot op story- en taakniveau planning.

Maar veel mensen werken in grote wereldwijde organisaties. Dit geeft andere uitdagingen. Grote organisaties hebben vele projecten en teams. Wanneer zij starten met agile werken, vervangen ze vaak de silo’s gebaseerd op rollen (zoals ontwikkelaars en testers) door scrum of feature team silo’s.

Wanneer vele grote teams op dezelfde codebasis werken, kan integratie een enorme uitdaging worden. Teams hebben nood aan specialisten zoals prestatie-, beveiligings-, en betrouwbaarheidstesters. Maar vaak zijn er onvoldoende van deze specialisten om alle crossfunctionele teams te ondersteunen. Het is zelfs moeilijk om alle problemen en uitdagingen zichtbaar te maken. Plannen op release-niveau is erg uitdagend in zo’n omgeving, maar is essentieel om nieuwe mogelijkheden naar klanten te brengen.

Ongeacht de context, moet het ontwikkelteam zijn verantwoordelijkheid nemen voor het plannen en afwerken van alle testactiviteiten, zelfs als daarvoor specialisten nodig zijn. Als ze afhankelijkheden hebben, dan moeten ze samenwerken met andere teams om die afhankelijkheden te beheren of te voorkomen, en dit bij voorkeur voordat het coderen start. Dat gezegd zijnde, er zullen altijd bijsturingen nodig zijn om geschikt te zijn voor elke unieke situatie.

Product

Het kwaliteitsniveau dat jouw stakeholders willen, hangt af van je product evenals het type en de hoeveelheid testen die mogelijk vereist zijn. Een contentmanagementsysteem dat alleen door interne gebruikers wordt gebruikt, heeft bijvoorbeeld andere prioriteiten dan software voor medische hulpmiddelen. Beide situeren zich in een andere omgeving en dat brengt verschillende risico’s met zich mee.

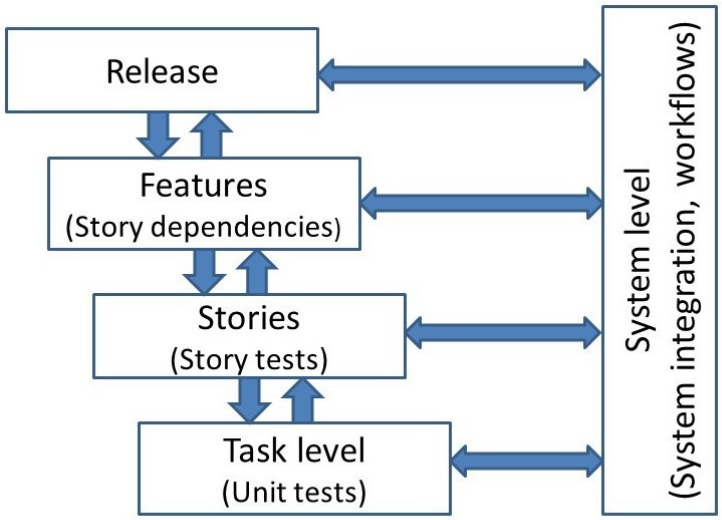
Hou zeker ook rekening met de grootte van je product, hoeveel mensen het gebruiken en of het is geïntegreerd met externe toepassingen. Denk na over hoe het product wordt opgeleverd en het risico dat gepaard gaat met het opleveringsmechanisme. Als de organisatie bijvoorbeeld zelf zijn webapplicatie host, heeft ze veel meer controle over wanneer en hoe vaak het product een update krijgt. Of als het product op veel apparaten zoals mobiele telefoons moet werken, hoe kunnen updates dan plaatsvinden zonder het reguliere gebruik te onderbreken?

Eén van de belangrijkste doelen van testen is het identificeren en verminderen van risico's - voor de gebruiker en de business. Uiteraard speelt het een grote rol in hoe je je testen plant. Dit is een van de redenen waarom ontwikkelteams het bedrijfsdomein leren kennen en nauw samenwerken met bedrijfsexperts. Domeinexpertise helpt bij het plannen van wat exact te testen. Heeft je team een heel goed idee van hoe het product wordt gebruikt? Hebben alle teamleden domeinkennis? Sowieso helpt samenwerken met product- en bedrijfsexperts het ontwikkelteam om optimale manieren te vinden om functionaliteiten te bouwen die je klanten waarderen.

Er zijn veel dingen om te overwegen met betrekking tot je productdomein. Het is niet alleen de software die je test; het is ook het product waar je eindgebruikers van afhankelijk zijn.

Planning op verschillende detailniveaus

Testen op meerdere niveaus (figuur 3.1) vereist extra planning. Releasecycli beginnen meestal met het bepalen van wat er in de eerste 'learning release' kan worden opgeleverd. Misschien wordt slechts een deel van een functionaliteit opgeleverd. Functionaliteiten worden opgesplitst in stories en geprioriteerd, zodat het team weet welke als eerste moet worden opgeleverd. Het is belangrijk dat het team het grote geheel begrijpt voordat ze stories in een iteratie opnemen. Wanneer ontwikkelaars aan een story werken, zijn ze meer gericht op het voltooien van individuele taken. Testers trappen soms in de val van alleen te denken aan de story die ze testen, dus is het belangrijk hen regelmatig te herinneren aan het grote geheel.



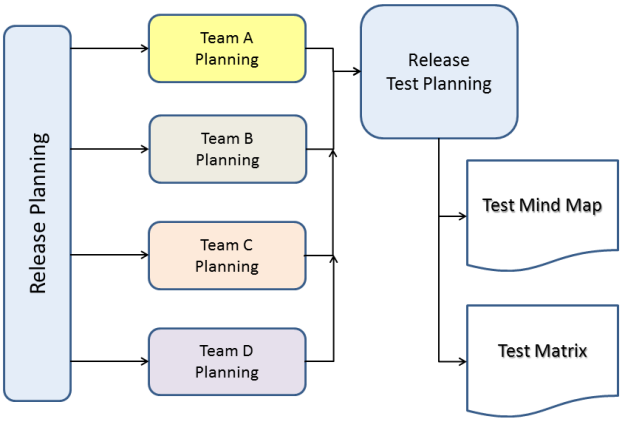
Figuur 3.1: Detailniveaus voor planning

Release/functionaliteitsniveau

Teams moeten begrijpen hoe elke geleverde story het grote geheel kan beïnvloeden, vooral in grotere of wereldwijde organisaties. Elke release kan bestaan uit vele functionaliteiten, die op hun beurt kunnen bestaan uit vele stories en taken die van invloed zijn op het systeem als geheel.

In grote organisaties, waar meerdere teams aan hetzelfde product werken, zien we vaak dat teams de neiging hebben om 'silo's' te worden. Ze vergeten met andere teams te praten om mogelijke afhankelijkheden op te lossen.

Figuur 3.2 toont het belang van een testaanpak waarbij alle teams werken aan één productrelease. Om een globaal overzicht te geven van de testdekking, overweeg om mensen van verschillende teams samen te brengen om een test mindmap of een functionaliteit test matrix (details in *More Agile Testing*) te maken die betrekking heeft op het product.



Figuur 3.2: Planning voor meerdere teams

Vergeet niet dat een standaardaanpak niet voor iedereen geschikt is. Hou dus zeker rekening met de grootte van de teams en het aantal teams, waar de teams zich bevinden, hoe het werk tussen teams wordt gecoördineerd en of alle vaardigheden die nodig zijn voor het testen beschikbaar zijn voor elk team.

Idealiter worden activiteiten gecoördineerd met andere teams naarmate de ontwikkeling van functionaliteiten vordert. Het is echter belangrijk om op te merken dat een meer voltooid product kan nodig zijn voor zaken als het toevoegen van definitieve screenshots aan gebruikers- of trainingsdocumentatie. Of, omdat het meestal geen snel proces is om een patch voor een mobiele applicatie te implementeren, kunnen aanvullende tests nodig zijn waarbij het hele team de nieuwste versie nog een keer verkent.

> ***Hint:*** * Maak niet de fout om testers, en eventueel operationeel personeel, de voorbereiding van de uitrol naar productie te laten afronden terwijl ontwikkelaars nieuwe stories beginnen. Net als ontwikkelingen, moet de voorbereiding van de oplevering een inspanning van het hele team zijn.*

Storyniveau

Op dit niveau maakt het niet uit of teams hun iteraties timeboxen of werken in een flow gebaseerde methode zoals Kanban. Begin met acceptatietests op hoog niveau (zie Hoofdstuk 4: Ontwikkeling begeleiden met voorbeelden voor details). Verzamel voorbeelden om ervoor te zorgen dat iedereen de story goed begrijpt en zet die voorbeelden om in tests. Als de testen geschreven worden voor het coderen, kunnen ze helpen bij de ontwikkeling en het voorkomen van defecten.

> ***Hint:*** *Overweeg welke verkennende testcharters nodig kunnen zijn (zie Hoofdstuk 6: Onderzoekend Testen). Denk na over de beperkingen van het product en wat dat betekent voor het testen van kwaliteitskenmerken (Zie Hoofdstuk 7: Testkwaliteitskenmerken).*

Terwijl teams testen plannen en de implementatie voor elke story bespreken, komen details over het testgebeuren naar boven. Maak nieuwe voorbeelden en testen aan om weer te geven wat er geleerd is over elke story.

Taakniveau

Ontwikkelaars gebruiken Test-Driven Development (TDD) om testen op een laag niveau (unit) te schrijven voorafgaand aan elk klein stukje code. Sommige ontwikkelaars noemen dit Design-Driven Development omdat het hen helpt om hun code te ontwerpen in functie van testbaarheid. Deze tests zijn relatief eenvoudig te schrijven, lopen snel en geven vlug feedback aan het team. Ze vormen een groot deel van de basis van het model van de testautomatiseringspiramide dat we behandelen in hoofdstuk 10: Het Visualiseren van een Testautomatiseringsstrategie

Planning voor regressietesten

Regressietesten hebben als doel ervoor zorgen dat het systeem doet wat het gisteren deed. Hedendaagse praktijken om regelmatig kleine veranderingen in productie op te leveren, laten geen tijd voor volledige handmatige regressiecontroles. Daarom werkt men aan testautomatisering terwijl het product wordt ontwikkeld. Testautomatisering zou deel moeten uitmaken van elke story, vooral op de servicelaag (zie Hoofdstuk 10: Visualiseren van een testautomatiseringsstrategie). Let op als een story een functionaliteit wijzigt, zorg er dan voor dat er taken opgenomen worden om de bestaande testen te wijzigen.

Geautomatiseerde regressietesten zorgen ervoor dat feedback snel komt, wat ons in staat stelt om vertrouwen te hebben in ons product. Veel (zo niet alle) testen worden uitgevoerd als onderdeel van Continuous Integration (CI). Sommige teams plannen dat de testen die langer duren minder vaak worden uitgevoerd. Bijvoorbeeld door ze meerdere keren per dag uit te voeren in plaats van bij elke build. Door releasefunctionaliteitsschakelaars te gebruiken om wijzigingen voor productiegebruikers te “verbergen”, kunnen teams sommige testactiviteiten asynchroon uitvoeren terwijl ze voortdurend in productie geïmplementeerd worden. De functionaliteit wordt “aangeschakeld” in productie wanneer alle testen voltooid zijn (zie hoofdstuk 8: Testen in DevOps).

Meer informatie hierover vind je in Hoofdstuk 23: Testen en DevOps in *Meer Agile Testen*.

Chapter 3: Test Planning in Agile Contexts

One of our top seven success factors from *Agile Testing* is "Don't Forget the Big Picture." Teams often get caught up in building, testing, and delivering the small increments – which we encourage – and forget about how that small slice fits into the system or how it works toward solving the business problem.

To plan testing activities effectively, a team needs to consider its context. To understand your context, think about three aspects of it: the team, the product, and the levels of detail of your system.

The team

Not all teams are created equal. If you're on a small, co-located team, you have an ideal situation for easy communication. You have a good chance of learning each other's values and sharing them. It's a sweet spot for delivering a great product, and planning is much easier. Teams can easily understand the next feature and dig down into story and task level planning.

However, many people work in large, globally distributed organizations. That brings on different challenges. Larger organizations have multiple projects and many teams. When they adopt agile, they often replace the silos based on role, such as developers and QA, with Scrum or feature team silos.

When many large teams work in the same code base, integration can become a huge challenge. Teams may need specialists such as performance, security, and reliability testers, but there may not be enough of them to go around to all the cross-functional teams. It's even hard to make all these issues and challenges visible. Release-level planning is particularly challenging in this environment but is critical to delivering new capabilities to customers.

No matter what the context, the delivery team must take responsibility for planning and completing all testing activities, even if it means bringing in specialists. If they have dependencies, they need to work with other teams to manage or eliminate those dependencies, preferably before coding starts. That said, adjustments need to be made to suit each unique context.

The product

The level of quality that your stakeholders want depends on your product as well as the type and amount of testing that might be required. For example, a content management system used only by internal users has different priorities than medical device software. Each has a different environment in which they dwell and involves different risks.

Consider the size of your product, how many people use it, or whether it is integrated with external applications. Think about how the product is delivered and the risk associated with the delivery mechanism. For example, if the organization is hosting its own web application, it has much more control about when and how often the product is updated. Or if the product needs to work on many devices such as phones, how do updates happen without interrupting regular usage?

One of the main purposes of testing is to identify and mitigate risks – to the user and to the business. Obviously, this plays a big part in how you plan your testing. This is one reason why delivery teams need to learn the business domain and work closely with business experts. Domain expertise helps when it comes to planning what to test. Does your team have a really good idea of how the product is used? Do all team members have domain knowledge? Collaborating with product and business experts helps the delivery team find optimal ways to build capabilities that your customers value.

There are many things to consider around your product domain. It is not only the software you are testing; it is the product that your end users depend on.

Planning across levels of detail

Testing across multiple levels (Figure 3.1) requires extra planning. Release cycles usually start by determining what might be delivered in the first "learning release." Perhaps only part of a feature will be delivered. Features are broken into stories and prioritized so the team knows which to deliver first. It's important that the team understands the big picture before they bring stories into an iteration. When developers work on a story, they're more focused on making sure individual tasks are completed. Testers sometimes fall into the trap of thinking only of the story they are testing, so reminders of the big picture are important.

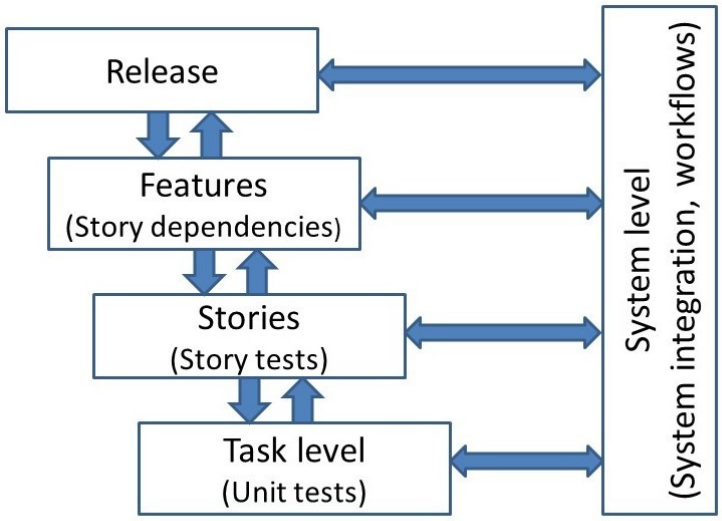


Figure 3.1: Levels of detail for planning

Release/feature level

Teams need to understand how each delivered story may affect the big picture, especially in larger or global organizations. Every release could be made up of many features, which in turn may be made up of many stories and tasks that have an impact on the system as a whole.

In large organizations with multiple teams all working on the same product, one of the problems we often see is that teams tend to become "siloed." They forget to talk to other teams to solve possible dependencies.

Figure 3.2 shows the importance of a test approach that includes all teams working toward a single product release. To give a big picture of test coverage, consider bringing people from different teams together to create a testing mind map or a feature test matrix (details in *More Agile Testing*) that encompasses the product.

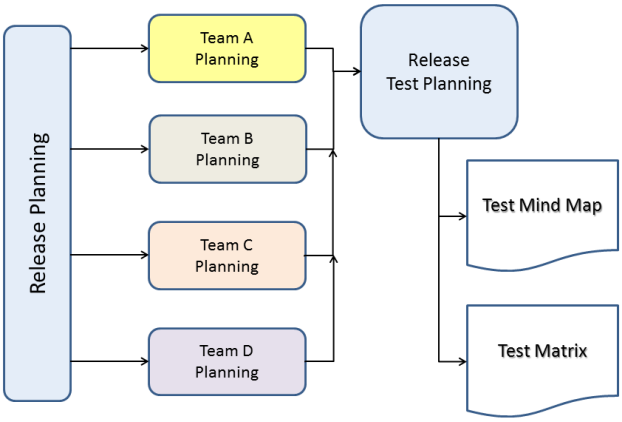


Figure 3.2: Planning for multiple teams

Remember, one size does not fit all, so make allowances for the size and number of your teams, where they are located, how work will be coordinated among teams, and whether all the skills needed for testing are available to each team.

Ideally, activities are coordinated with other teams as feature development progresses. However, it is important to note that a more finalized product may be needed for things like adding final screenshots to user or training documentation. Or, because it's not usually a quick process to deploy a patch for a mobile application, additional testing may be needed where the whole team explores the newest version one more time.

> ***Hint:*** *Do not make the mistake of having testers and perhaps operations staff wrapping up the pre-deploy endgame while developers start new stories. Just like development, preparing for delivery should be a whole-team effort.*

Story level

At this level, it doesn't matter whether teams are time-boxing their iterations or working in a flow-based method such as Kanban. Start with high-level acceptance tests (see Chapter 4: Guiding Development with Examples for details). Get examples to increase shared understanding of the story and turn those examples into tests. If the tests are written before coding happens, they help guide the development and prevent defects.

> ***Hint:*** *Consider what exploratory test charters might be needed (see Chapter 6). Think about the product's constraints and what that means for testing quality attributes (Chapter 7).*

As teams plan testing and discuss implementation for each story, details about testing emerge. Create new examples and tests to reflect what has been learned about the story.

Task level

Programmers use Test-Driven Development (TDD) to write low-level (unit) tests prior to each small piece of code. Some programmers call it Design-Driven Development since it helps them to design their code for testability. These tests are relatively easy to write, and they run quickly and give fast feedback to the team. They form much of the base of the test automation pyramid model we discuss in Chapter 10: Visualizing a Test Automation Strategy.

Planning for regression testing

Regression testing is about making sure the system does what it did yesterday. Contemporary practices for delivering small changes to production frequently leave no time for full manual regression checking, so the test automation is created as the product is developed. Test automation should be part of each story, especially at the service layer (see Chapter 10). If a story changes functionality, be sure to include tasks to change the existing tests.

Automated regression tests allow us to have confidence in our product with fast feedback. Many (if not all) of the tests run as part of continuous integration (CI). Some teams schedule slower tests to run less often. For example, running them several times a day instead of every build. Using release feature toggles to "hide" changes from production users allows teams to do some testing activities asynchronously as they continually deploy to production. The feature is "turned on" in production when all testing is completed (see Chapter 8: Testing in DevOps).

You'll find more information about this in Chapter 23: Testing and DevOps in *More Agile Testing*.

DEEL 2: Testaanpakken

In dit deel duiken we in de basistechnieken van agile testen. Het gebruik van concrete voorbeelden om de ontwikkeling te sturen, is een van de effectiefste manieren om vertrouwen op te bouwen bij elke nieuwe wijziging. We delen manieren om teamleden in verschillende rollen te helpen leren samen te werken om kwaliteit in het product in te bouwen. Onderzoekend testen is een andere vertrouwen-opbouwende kernpraktijk waar het hele team zich zou moeten mee bezighouden.

Agile teams trappen vaak in de valkuil om enkel functionaliteit te testen; hoe elke functionaliteit of mogelijkheid zich moet gedragen. Er zijn nog veel andere belangrijke kwaliteitskenmerken die we moeten testen, die we ook behandelen in dit deel.

DevOps en continuous delivery zijn vandaag de dag hot topics. We kijken hoe testen and testers daarin passen en helpen hun team succesvol te zijn bij deze aanpakken.

SECTION 2: Testing Approaches

In this section, we dive into core practices for agile testing. Using concrete examples to guide development is one of the most effective ways to build confidence in each new change. We share ways to help team members in different roles learn to collaborate to build quality into the product. Exploratory testing is another core confidence-building practice in which the whole team should engage.

Agile teams often fall into the trap of only testing for functionality â^' how each feature or capability should behave. There are many other important quality attributes that we need to test, which we also cover in this section.

DevOps and continuous delivery are hot topics today. We look at how testing and testers fit in and help their team succeed with those approaches.

Hoofdstuk 4: De ontwikkeling in goede banen leiden met voorbeelden

Het idee om voorbeelden te gebruiken om de ontwikkeling van functionaliteiten en stories in goede banen te leiden, wordt al jaren door veel teams toegepast. We beschouwen het als een beproefde, waardevolle aanpak. Vooraanstaande beoefenaars blijven nieuwe manieren vinden om teams te helpen slagen met deze technieken, bijvoorbeeld met Matt Wynne's *Example Mapping* (zie meer in Hoofdstuk 5).

Concrete voorbeelden van gewenst en ongewenst systeemgedrag helpen teams om gedeeld begrip op te bouwen van elke functionaliteit of story. Hierdoor kunnen ze de juiste dingen ontwikkelen, met minder afgewezen stories en een kortere cyclustijd van start tot productie. De bijdrage van testers bestaat erin naar deze concrete voorbeelden te vragen en ze te gebruiken om uitvoerbare testen te maken die de ontwikkeling in goede banen leiden. Zij kunnen de ervaringsdeskundige zijn bij het leiden van deze gesprekken.

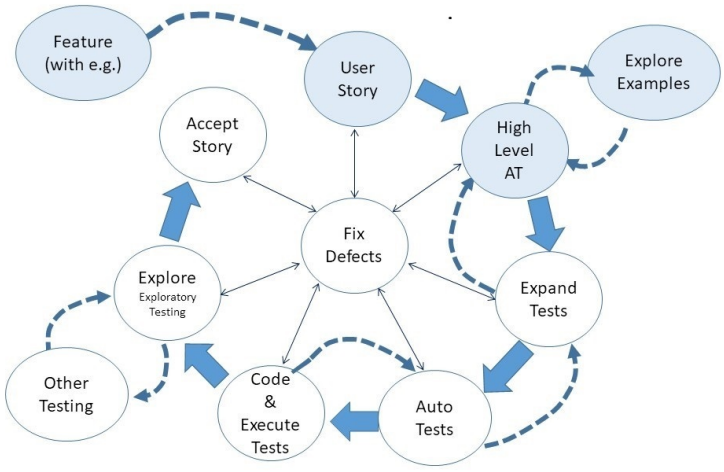
Voorbeeldgebaseerde methoden

Er zijn een paar varianten voor het bouwen van functionaliteiten en stories op basis van voorbeelden. **Gedragsgestuurde ontwikkeling** of *Behavior-driven development* (BDD) is degene waarvan Janet hoort dat de meeste teams beweren dat ze deze gebruiken. BDD, voor het eerst geïntroduceerd door Daniel Terhorst-North, legt voorbeeldscenario's vast in een natuurlijke, domeinspecifieke taal. De "Given/When/Then" syntax beschrijft precondities, een triggeractie en de resulterende postconditie. Tijdens het schrijven van productiecode is het waarschijnlijk dat ontwikkelaars enkele of alle scenario's automatiseren om te weten of ze hebben opgeleverd wat de klant wil.

Het schrijven van deze scenario's klinkt misschien eenvoudig, maar het vergt oefening om testen zo te vereenvoudigen dat echt maar één ding wordt getest. Zie Figuur 4.1 voor een voorbeeld.

Acceptatietest gestuurde testen of *Acceptance test-driven development* (ATDD) is vergelijkbaar. Het is een generiekere manier om ontwikkeling met voorbeelden in goede banen te leiden zonder een strikte taal of regels. De meeste mensen gebruiken ATDD voor functionele testen, hoewel vereisten van andere kwaliteitskenmerken, zoals beveiligbaarheid of toegankelijkheid er ook kunnen worden in opgenomen. Eén ATDD aanpak die we hebben gebruikt, is om bij het plannen van een story tenminste één voorbeeld van het gewenst gedrag of succespad (*happy path*) vast te leggen en tenminste één voorbeeld voor ieder type ongewenst gedrag. Testers en andere teamleden achterhalen meer gedetailleerde voorbeelden terwijl het programmeerwerk aan de story doorgaat. Ten minste enkele van de voorbeelden worden omgezet in uitvoerbare testen die het team helpen bepalen wanneer ze klaar zijn.

Figuur 4.1 toont de voortgang van activiteiten, startend met het opdelen van een feature in user stories. De blauwe activiteiten worden gedaan als onderdeel van *story readiness workshops, backlog refinement* of *"three amigos"* discussies.



Figuur 4.1: Acceptatietest gestuurde testen (ATDD)

Teams die **specificatie door voorbeeld** of *Specification by Example* (SBE) toepassen, beginnen met het identificeren van doelen rond de story, bijvoorbeeld met een aanpak als *Impact Mapping* (zie meer in hoofdstuk 5). Tijdens een specificatie workshop achterhalen teams vervolgens belangrijke voorbeelden, die uitgroeien tot specificaties. Tijdens de verdere ontwikkeling worden voorbeelden verfijnd en omgezet in uitvoerbare specificaties om zo het product regelmatig te valideren. Deze uitvoerbare voorbeelden worden, net als bij BDD en ATDD, levende documentatie van de applicatie. Toen Gojko Adzic als eerste de term *Specification by Example* gebruikte, heeft hij bewust niet het woord 'testen' gebruikt om deze activiteiten te beschrijven.

Waarom voorbeelden helpen

Door elke nieuwe functionaliteit vanuit verschillende perspectieven te bekijken, is de kans groter dat een team de waarde vaststelt die klanten uit elke nieuwe functionaliteit halen. Deze diversiteit helpt alle teamleden om onbewuste vooroordelen te overwinnen en "out of the box" te denken. Wanneer elke stakeholder wordt gevraagd naar concrete voorbeelden van systeemgedrag, bekijkt het team al die voorbeelden en is het gemakkelijk om afwijkingen te ontdekken. Het is ook gemakkelijker om de minimale specifieke waarde van wat de klant nodig heeft, te achterhalen. Dat maakt het voor het team mogelijk om "net genoeg" van het juiste te leveren.

We zullen je laten zien wat we bedoelen aan de hand van een voorbeeld.

Story: Als Canadese klant wil ik de kassierster cash betalen en verwacht ik het correcte wisselgeld terug, zodat ik het juiste bedrag betaal voor mijn boodschappen. De kassa toont het juiste wisselgeld om terug te geven.

Scenario: Het succespad waar de klant meer geeft dan het bedrag voor de boodschappen en het juiste wisselgeld terugkijgt.

Given Ik ben een Canadese klant en heb boodschappen gekocht ter waarde van $4.97,

When Ik geef de kassierster $5.00,

Then Ik verwacht $0.05 wisselgeld terug te krijgen.

Opmerking: Door dit voorbeeld te gebruiken, ontdekte het team een bedrijfsregel waar nog niet aan was gedacht. In Canada zijn er geen pennies (0.01), bedragen worden afgerond naar boven of beneden naar de dichtsbijzijnde nickel (0.05).

Eén van Janet's favoriete manieren om voorbeelden weer te geven is in tabelformaat. Daarmee wordt snel zichtbaar wat ontbreekt en wat mensen denken als ze discussiëren. Elke regel kan een test worden. Figuur 4.2 toont dit formaat voor het scenario hierboven.

Grocery total	Cash given	Change returned	Test
4.97	5.00	0.05	Happy path
4.97	4.00	0.00	Not enough cash
4.97	10.00	5.05	Happy path

Figuur 4.2: Voorbeeld van een tabelformaat van het gebruik van concrete voorbeelden

Er zijn zoveel goede manieren om gesprekken te structureren waarbij teams voorbeelden achterhalen. Jeff Patton's *Story Mapping*[7] helpt het hele team de klantreis (*customer journey*) van hun product te doorlopen. We gebruiken graag *Example Mapping* om specifieke bedrijfsregels met illustrerende voorbeelden vast te leggen. Gestructureerde gesprekken met behulp van Ellen Gottesdiener en Mary Gorman's 7 product dimensies[8] zorgen ervoor dat teams voorbeelden krijgen van veel verschillende aspecten van waarde. Elke techniek die face-to-face samenwerking in kleine, cross-functionele groepen promoot, is het proberen waard. Zie hoofdstuk 7 in *More Agile Testing* voor meer details en verhalen.

Dit is je basis

Leden van oplever- en businessteams samenbrengen om voorbeelden te verzamelen is de sleutel tot het frequent opleveren van waarde aan klanten in een duurzaam tempo. Het stelt teams in staat om het zó belangrijke gedeeld begrip op te bouwen nog vóór ze beginnen te programmeren. Het helpt iedereen om met beide benen op de grond te staan. De concrete voorbeelden worden uitvoerbare tests, die ervoor zorgen dat we het juiste bouwen en dat dit in de toekomst correct blijft werken totdat klanten het willen wijzigen.

> ***Hint:*** *Wanneer je merkt dat je in een vage discussie bent beland over de vereisten van een functionaliteit, of in een twist over hoe een functionaliteit zich moet gedragen, STOP. Vraag om een voorbeeld. Of beter nog, zet de groep aan het tekenen van voorbeelden op een whiteboard (echt of virtueel). Je zal veel tijd besparen en dichter komen bij het bouwen van het juiste ding.

[7] https://www.jpattonassociates.com/user-story-mapping/
[8] https://discovertodeliver.com/image/data/Resources/visuals/DtoD-7-Product-Dimensions.pdf

Chapter 4: Guiding Development with Examples

The idea of using examples to guide development of features and stories has been used by many teams for years. We see it as a tried-and-true, valuable approach. Leading practitioners continue to find new ways to help teams succeed with these techniques: for example, Matt Wynne's example mapping (see more in Chapter 5).

Concrete examples of desired and undesired system behavior help teams build a shared understanding of each feature and story. This enables them to build the right thing with fewer story rejections and shorter cycle time from start to production deploy. Testers contribute by asking for these concrete examples and using them to create executable tests that guide development. They can be the voice of experience at leading these conversations.

Example-based methods

There are a few variations for building features and stories based on examples. Behavior-driven development (BDD) is the one that Janet hears most teams claim they use. BDD, first introduced by Daniel Terhorst-North, captures example scenarios in a natural, domain-specific language. The "Given/When/ Then" syntax describes preconditions, some trigger action, and resulting post-condition. As developers write the production code, they are likely to automate some or all the scenarios to help know when they've delivered what the customer wants.

Writing these scenarios may sound easy, but it takes practice to simplify tests so that there really is only one thing being tested. See Figure 4.1 for an example.

Acceptance test-driven development (ATDD) is similar. It is a more generic form of guiding development with examples without strict language or rules. Most people use ATDD for functional tests, although requirements for other quality attributes, such as security or accessibility, can be included. One ATDD approach we've used is to capture at least one high-level example of desired or "happy path" behavior and at least one example for each type of misbehavior as the team plans the story. Testers and other team members elicit more detailed examples as coding proceeds on the story. At least some of the examples are turned into executable tests that help the team decide when they're done.

Figure 4.1 shows a progression starting with slicing the feature into stories. The blue bubbles are done as part of story readiness workshops, backlog refinement, or "three amigos" discussions.

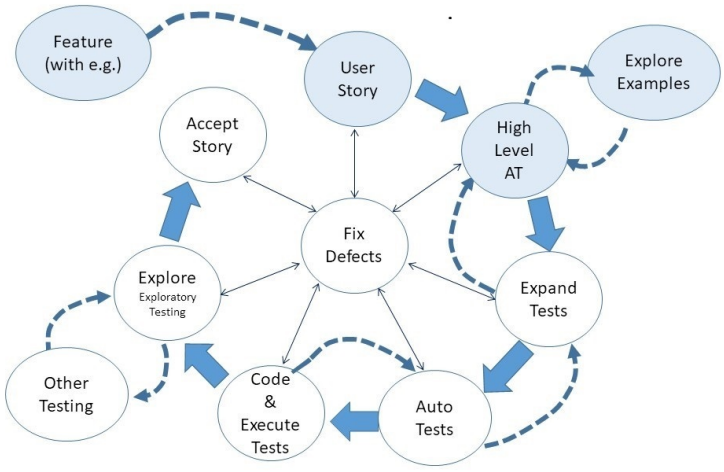


Figure 4.1: Acceptance test-driven development (ATDD)

Teams practicing Specification by Example (SBE) start by identifying goals around the story using an approach like impact mapping (See more in Chapter 5). The team then elicits key examples, which become the specifications, during a specification workshop. As development proceeds, examples are refined and turned into executable specifications to validate the product frequently. These executable examples, as with BDD and ATDD, become living documentation of the application. When Gojko Adzic coined the term Specification by Example, he deliberately did not use the word "test" to describe any of the activities.

Why examples help

Looking at each new capability from a variety of perspectives helps a team be more likely to pinpoint the value customers get from each new capability. That diversity helps each team member overcome unconscious biases and "think outside the box." When each stakeholder is asked for concrete examples of system behavior, teams look at those examples, and it's easy to see discrepancies. It is also easier to dig down to the minimum specific value of what customers need and enables teams to deliver "just enough" of the right thing.

We'll show you what we mean by using an example scenario.

Story: As a Canadian shopper, I want to give the cashier cash and I expect the correct change so that I only pay the right amount for my groceries. The cash register reports the correct amount of change to give.

Scenario: The happy path where the shopper gives more than the amount of the groceries and receives the correct change.

> **Given** I am a Canadian shopper and have purchased groceries worth $4.97,
>
> **When** I give the cashier $5.00,
>
> **Then** I expect to get $0.05 change.

Note: By using this example, a business rule that the team may not have considered is that in Canada, there are no pennies in use, so the number is rounded up or down, and change is given to the nearest nickel (0.05).

One of Janet's favorite ways to show examples is in a tabular format. It quickly shows what you are missing and what people are thinking as they have the discussion. Each line can become a test. Figure 4.2 shows this format for the scenario we used above.

Grocery total	Cash given	Change returned	Test
4.97	5.00	0.05	Happy path
4.97	4.00	0.00	Not enough cash
4.97	10.00	5.05	Happy path

Figure 4.2: Example of a tabular format of using concrete examples

There are so many good ways to structure conversations where teams can elicit examples. Jeff Patton's story mapping[9] helps the entire team walk their user's journey through their product. We like to capture specific business rules as well as examples that illustrate them with example mapping. Structured conversations using Ellen Gottesdiener and Mary Gorman's 7 product dimensions[10] ensure teams get examples of many different aspects of value. Any technique that promotes face-to-face collaboration in small cross-functional groups is worth trying. See Chapter 7 in *More Agile Testing* for more details and stories.

This is your foundation

Getting delivery and business team members together to gather examples is key to delivering value to customers at a frequent and sustainable pace. It enables teams to build the all-important shared understanding before they start coding. It helps everyone stay grounded in reality. The concrete examples become executable tests that ensure we build the right thing and that thing keeps working correctly into the future until customers want to change it.

> ***Hint**: When you find yourself in a hand-wavy discussion about requirements for a feature or an argument over how a certain capability should behave, STOP. Ask for an example. Even better, get the group to start drawing examples on the whiteboard (real or virtual). You'll save lots of time and get closer to building the right thing.*

[9] https://www.jpattonassociates.com/user-story-mapping/
[10] https://discovertodeliver.com/image/data/Resources/visuals/DtoD-7-Product-Dimensions.pdf

Chapter 5: Samenwerking mogelijk maken

Samenwerking binnen een team en tussen teams is één van de pijlers die agile teams succesvol maken. We merken echter dat veel teams geen idee hebben hoe te beginnen met het opbouwen van die relaties. In dit hoofdstuk zullen we het hebben over een paar zeer eenvoudige oefeningen die jou en je team kunnen helpen grip te krijgen.

Samenwerken met klanten

Laten we beginnen met de samenwerking met de klant – meestal vertegenwoordigd door een product owner. Als teams niet begrijpen welk probleem de klant probeert op te lossen, kunnen ze het verkeerde probleem oplossen. Het is essentieel dat teams samenwerken met hun klant om zijn ware behoeften te begrijpen.

Ten eerste, raden we ten zeerste aan dat iedereen in het team het domein begrijpt. Dit kan worden bereikt door nauw samen te werken met eindgebruikers, te vragen naar voorbeelden of scenario's, of zelfs tekeningen te maken op whiteboards om verschillen te begrijpen en betekenissen te verduidelijken.

Vragen als deze zullen de klant doen nadenken over het gebruik en het bijbehorende risico.

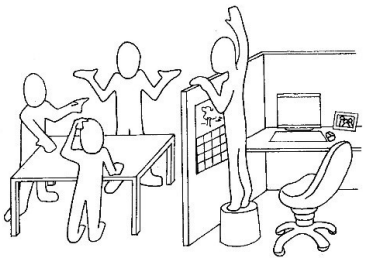
> "Hoe zal u dit gebruiken?"
>
> "Wat is het ergste dat er kan gebeuren?"

Testers kunnen de communicatie tussen ontwikkelaar en klant vergemakkelijken, maar het is belangrijk om niet in de weg te staan. We noemen de praktijk van het samenbrengen van een tester, een programmeur en een bedrijfsexpert (product owner, product manager of business analist) om te praten over een user story, de "Kracht van drie." George Dinwiddie noemt het de **Drie Amigo's**[11]. Het is een krachtige manier om een gedeeld begrip op te bouwen over stories, functionaliteiten en hoe ze in het product passen.

> ***Hint:*** * Breng de tester, programmeur en bedrijfsexpert en misschien één of twee andere rollen, samen wanneer er een vraag opkomt. Een ontwikkelaarspaar werkt bijvoorbeeld aan een nieuwe story en één van de bedrijfsgerichte testen mislukken. Ze gaan met een tester praten en zeggen dat ze denken dat de test het verkeerde gedrag verwacht. Dat is het moment om een product owner vast te nemen en een drieweggesprek te voeren. Deze korte conversatie bespaart later zoveel tijd bij het herstellen van een defect dat in de code is geraakt.*

Afhankelijk van het product en het type functionaliteiten dat wordt ontwikkeld, kunnen meer perspectieven nodig zijn, zoals die van een UX-ontwerper, een data-expert, of een operationeel expert.

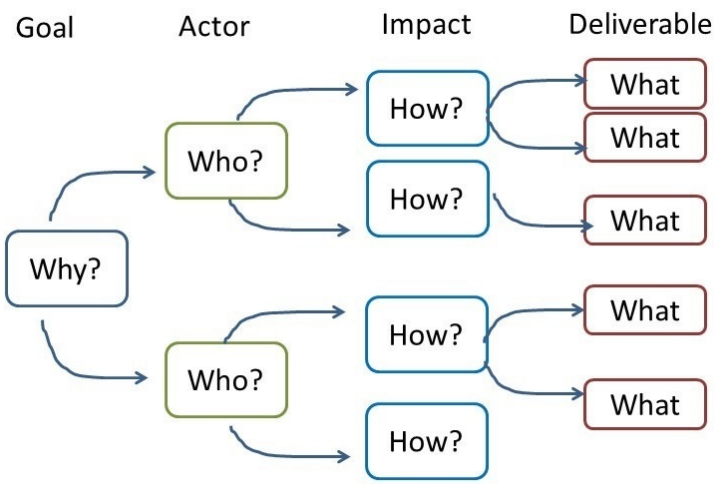
[11] https://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories



Figuur 5.1: Nodig de juiste mensen uit

Impact mapping

Frameworks zoals **impact mapping**[12] zijn nuttig bij het beslissen welke functies we moeten bouwen en bepalen misschien zelfs wat de prioriteit moet zijn. Begin met het doel van een functionaliteit (het "waarom"). Stel vervolgens vast wie kan helpen dat doel te bereiken en wie ons in de weg zou kunnen staan. Vraag voor elke "wie," hoe ze ons kunnen helpen of hinderen bij het bereiken van het doel (dat zijn de impacten). Denk ten slotte eens na welke resultaten kunnen voortvloeien uit elke impact (het "wat"). Deze oefening helpt het team het grote plaatje en de redenen voor wat ze ontwikkelen te begrijpen.



Figuur 5.2: Impact mapping

Het beantwoordt de vraag "Hoe weten we of deze functionaliteit het doel bereikt nadat we het hebben gereleased?"

[12] https://www.impactmapping.org/

Stel vragen

Het is gebruikelijk dat een feature planning meeting begint met een discussie over de implementatie van de functionaliteit. Soms komt de product owner met eigen ideeën: “Neem dezelfde code die wij gebruiken voor kortingscodes en maak er negatieve bedragen van zodat we toeslagen kunnen toevoegen.” (Ja, Lisa had exact die ervaring.) Het is belangrijk om dat niet te laten gebeuren - begin met het waarom.

Wanneer je samenkomt met een business stakeholder, zoals een product owner, om te praten over aankomende functionaliteiten, is de eerste vraag die gesteld moet worden: “Waarom doen we deze functionaliteit?” Andere goede vragen: “Welk probleem lost dit op voor het bedrijf, de klant of de eindgebruiker?”

QA staat voor “Question Asker” – een idee dat we kregen van Pete Walen. Testers stellen routinematig vragen die niemand anders denkt te stellen, dus als je geen tester in het team hebt, probeer dan een vraagsteller-rol aan te wijzen.

Ervaren teams bouwen kwaliteitscriteria vaak in de manier waarop ze werken. Als ze bijvoorbeeld beveiligingsproblemen zoals cross-site scripting (XSS) en SQL-injectie willen vermijden, zit dit waarschijnlijk in de architectuur van het systeem ingebouwd.

Onze ervaring is echter dat business stakeholders vaak ten onrechte van uitgaan dat het technische team al weet welke kwaliteitsattributen belangrijk zijn - attributen zoals hoeveel gelijktijdige gebruikers het product gaan gebruiken, welke apparaten moeten ondersteund worden, of hoe snel de waargenomen antwoordtijd van een toepassing moet zijn. Zie hoofdstuk 7 voor meer informatie over kwaliteitsattributen.

Het stellen van zowel specifieke als open vragen helpt om verborgen aannames bloot te leggen.

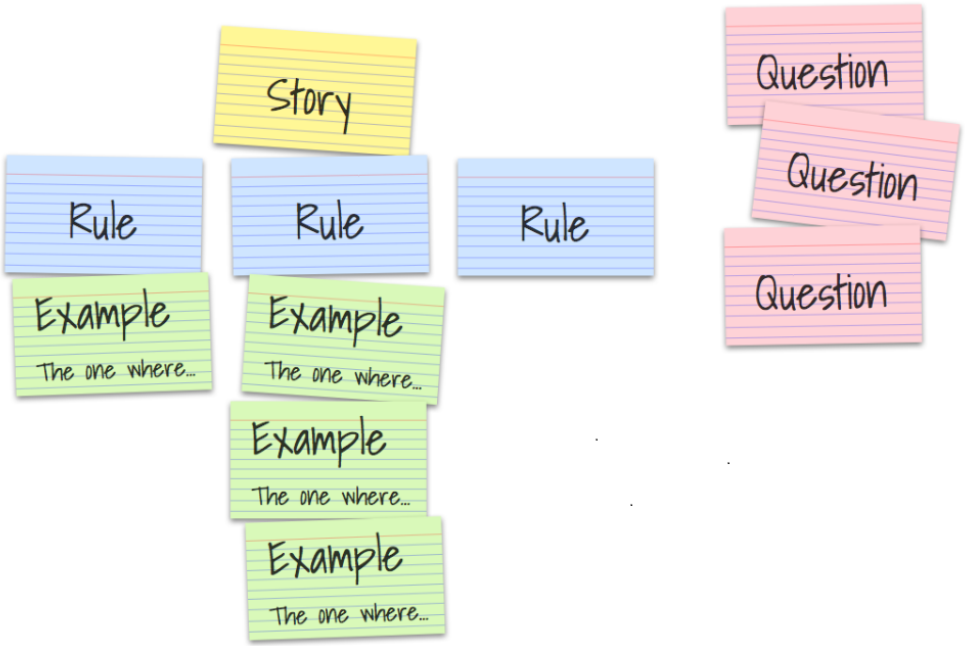
- “Is het mogelijk dat we deze functionaliteit implementeren en het probleem niet oplossen?”
- “Wat zullen gebruikers doen voordat ze deze functionaliteit gebruiken?”
- “Wat gaan ze daarna doen?”

Voorbeeld mapping

Matt Wynne heeft ons kennis laten maken met het idee van voorbeeldmapping[13] (*example mapping*), en wij vonden het een geweldige manier om een nieuwe functionaliteit te verkennen en de waarde die het zou opleveren. Werk samen met de product owner of andere stakeholders over de bedrijfsregels in een discussie van het type “Kracht van drie”. Bedrijfsregels zijn geweldig om te beginnen met het onderzoeken van een functionaliteit, omdat ze ons kunnen helpen een functionaliteit op te delen in stories en eveneens een gedeeld begrip te verkrijgen van hoe de functionaliteit zich zou moeten gedragen.

De voorbeeldmappingtechniek van Matt Wynne is een zeer effectieve basis voor dit type gesprek omdat concrete voorbeelden worden gebruikt om ons begrip van de regels te helpen verduidelijken. Hou tijdens het gesprek het hoofddoel voor ogen en concentreer je op de waarde die de functionaliteit oplevert voor klanten en eindgebruikers. Teams vinden vaak dat er meer bedrijfsregels worden blootgelegd door het gebruik van echte voorbeelden.

[13] https://cucumber.io/blog/example-mapping-introduction/



Figuur 5.3: Example mapping

Het gebruik van voorbeeldmapping om bedrijfsregels, voorbeelden, en vragen die beantwoord moeten worden, in kaart te brengen, is een effectieve manier om ervoor te zorgen dat het team op één lijn zit wanneer ze de iteratie plannen.

Bouw vertrouwen op met zichtbaarheid

Testen stelt teams in staat risico's te identificeren, zodat klanten de beste beslissingen kunnen maken, wat op zijn beurt vertrouwen opbouwt. Wanneer het team om input vraagt krijgen klanten het vertrouwen dat ze werkende software krijgen.

We kunnen mindmaps, stroomdiagrammen, contextdiagrammen, toestandsdiagrammen, of andere hulpmiddelen gebruiken om te kijken naar de afhankelijkheden en de rimpeleffecten van elke nieuwe functionaliteit. Als onze functionaliteiten niet gemakkelijk door iedereen worden begrepen en gebruikt, kunnen ze niet de beoogde waarde bieden. Het grote geheel in de gaten houden is één van de sterke punten die testers brengen in agile teams.

Tekenen op een whiteboard tijdens het bespreken van een story is een bewezen manier om communicatie te optimaliseren. Opstaan en bewegen helpt mensen doen denken en leren.

> ***Hint**: Pak een stift en wat indexkaarten, plakbriefjes, een whiteboard. Krijg mensen zo ver om recht te staan en actief mee te doen door te tekenen of indexkaarten of plakbriefjes te verplaatsen.*

Figuur 5.4: Gebruik zichtbaarheid om vertrouwen op te bouwen

Als er deelnemers op afstand zijn, gebruik dan online samenwerkingstools om te helpen. Onze teams hebben ontdekt dat het gebruik van mindmaps, ofwel op een fysiek whiteboard of via een real-time samenwerkingstool zoals Mindmup, uiterst effectief kan zijn in het helpen om de onbekenden te identificeren en creatieve oplossingen te vinden.

Dit soort visuele hulpmiddelen stellen teams in staat om beter en gerichter vragen te stellen. Telkens wanneer het team een probleem tegenkomt, vinden ze een manier om het zichtbaar te maken, zodat ze kunnen beginnen na te denken over experimenten om het probleem kleiner te maken.

Chapter 5: Enabling Collaboration

Collaboration within a team and between teams is one of the cornerstones that make agile teams successful. However, we find that many teams have no idea how to get started with building those relationships. In this chapter, we will talk about a few very simple practices that can help you and your team get traction.

Collaborate with customers

Let's start with collaborating with the customer – usually represented by a product owner. If teams don't understand what problem the customer is trying to solve, they may solve the wrong one. It is essential that teams work with their customer to understand their true needs.

First, we strongly suggest that everyone on the team understands the domain. This can be accomplished by working closely with end users, asking for examples or scenarios, or even drawing pictures on whiteboards to understand differences and clarify meanings.

Questions such as these will make the customer consider the usage and the associated risk.

> "How will you use this?"
>
> "What's the worst that can happen?"

Testers can facilitate developer-customer communication, but it's important not to get in the way. We call the practice of getting a tester, a programmer, and a business expert (product owner, product manager, or business analyst) together to talk about a user story, the "Power of Three." George Dinwiddie refers to it as the **Three Amigos**[14]. It is a powerful way to build shared understanding about stories, features, and how they fit into the product.

> ***Hint**: Gather the tester, programmer, and business expert, and perhaps one or two other roles, together anytime a question comes up. For example, a developer pair is working on a new story, and one of the business-facing tests fails. They go talk to a tester and say they think the test is expecting the wrong behavior. That's the time to grab a product owner and have a three-way discussion. This quick conversation saves so much time later trying to fix a defect that made it into the code.*

Depending on the product and the type of features being developed, more perspectives may be needed, such as from a UX designer, a data expert, or an operations expert.

[14] https://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories

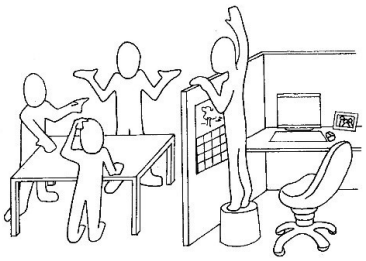


Figure 5.1: Invite the right people

Impact mapping

Frameworks such as **impact mapping**[15] are helpful in deciding what features we should build and maybe even determine what the priority should be. Start with the goal of a feature (the "why"). Then identify who might help us achieve that goal and who might get in our way. For each "who," ask how they might help or hinder us in achieving the goal (those are the impacts). Lastly, think about what deliverables might result from each impact (the "what"). This exercise helps the team understand the big picture and the reasons behind what they are developing.

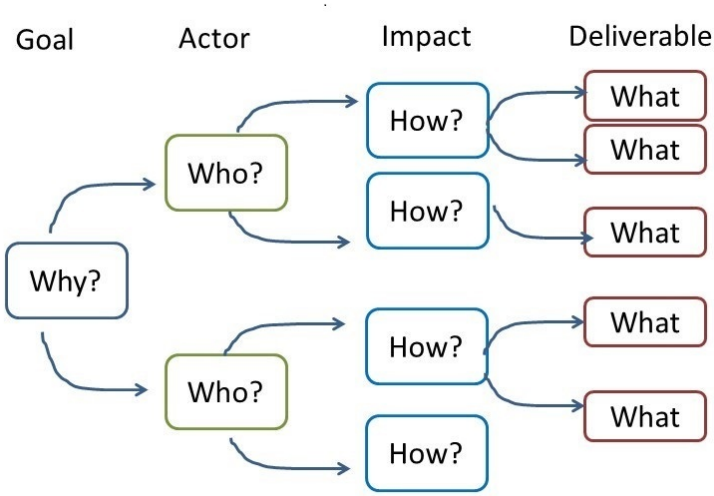


Figure 5.2: Impact mapping

It answers the question "How will we know if this feature achieves the goal after we release it?"

[15] https://www.impactmapping.org/

Ask questions

It's common that a feature planning meeting starts with a discussion about how to implement the feature. Sometimes the product owner has come with her own ideas: "Take the same code as we use for discount codes and make them negative amounts so we can add surcharges." (Yes, Lisa had that exact experience.) It is important not to let that happen – start with the why.

When you get together with a business stakeholder such as a product owner to talk about upcoming features, the first question to ask is, "Why are we doing this feature?" Other good questions: "What problem will this solve for the business, the customer, or the end user?"

QA stands for "Question Asker" – an idea we got from Pete Walen. Testers routinely ask questions that nobody else thinks of asking, so if you don't have a tester on the team, try designating a question-asker role.

Experienced teams often build quality criteria into the way they do work. For example, if they want to prevent security issues such as cross-site scripting (XSS) and SQL injection, it is probably built into the architecture of the system.

However, in our experience, business stakeholders often assume incorrectly that the technical team already knows what quality attributes are important – attributes like how many concurrent users will be using the product, what devices need to be supported, or how fast the perceived response time of an application needs to be. See Chapter 7 for more on quality attributes.

Asking specific as well as open-ended questions help expose hidden assumptions.

- "Is it possible we could implement this feature and not solve the problem?"
- "What will users do before using this feature?"
- "What will they do afterwards?"

Example mapping

Matt Wynne introduced us to the idea of example mapping[16], and we found it to be a great way to explore a new feature and the value it should deliver. Work with the product owner or other stakeholders about the business rules in a "Power of Three" type discussion. Business rules are great places to start exploring a feature, since they can help us slice a feature into stories as well as get a shared understanding of how the feature should behave.

Matt Wynne's example-mapping technique is a highly effective basis for this type of conversation because concrete examples are used to help clarify our understanding of the rules. As the conversation continues, keep the main goal in mind, and focus on the value the feature delivers to customers and end users. Teams often find that more business rules are exposed as a result of using real examples.

[16] https://cucumber.io/blog/example-mapping-introduction/

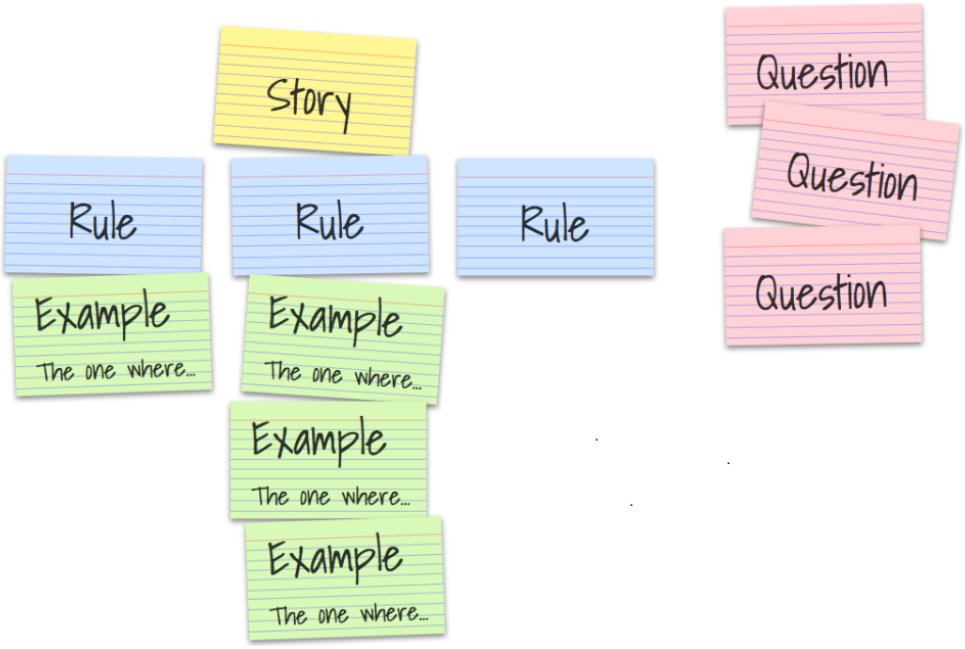


Figure 5.3: Example mapping

Using example mapping to elicit business rules, examples, and questions that need answering is an effective way to make sure the larger team starts on the same page when they plan the iteration.

Build trust using visibility

Testing allows teams to identify risks so customers can make the best decisions, which in turn builds trust. When the team asks for their input, customers gain confidence that they will get working software.

We can use mind maps, flow diagrams, context diagrams, state diagrams, or other tools to look at dependencies and ripple effects of each new feature. If our features aren't easily understood and used by everyone, they can't provide the intended value. Keeping an eye on the big picture is one strength that testers bring to agile teams.

Drawing on a whiteboard while discussing a story is a proven way to optimize communication. Getting up and moving helps people think and learn.

> ***Hint**: Grab a marker and some cards, stickies, a whiteboard. Get people to stand up and actively participate by drawing or moving index cards or sticky notes around.*

Figure 5.4: Use visibility to create trust

If there are remote participants, use online collaborative tools to help. Our teams have found that using mind maps, either on a physical whiteboard or via a real-time collaborative tool such as Mindmup, can be extremely effective in helping to identify the unknowns and find creative solutions.

These types of visual aids enable teams to ask better and more focused questions. Anytime the team encounters a problem, they find a way to make it visible, so that they can start thinking of experiments to make the problem smaller.

Hoofdstuk 6: *Onderzoekend* Testen

Meer agile teams vinden waarde in *onderzoekende* testen, maar het is nog steeds een nieuw of onbekend idee voor veel teams. We beginnen met het uitleggen van het doel van *onderzoekende* testen en wat daarbij komt kijken.

In *Explore It!* definieert Elisabeth Hendrickson *onderzoekende* testen als "…**gelijktijdig** ontwerpen en uitvoeren van tests om meer te weten te komen over het **systeem**, waarbij je je inzichten uit het laatste **experiment** gebruikt om het volgende te informeren. "

Bij *onderzoekende* testen werkt een persoon met het systeem en observeert hij het echte gedrag, waarbij kleine experimenten worden ontworpen. Op basis van wat ze leren, passen ze het experiment aan en blijven ze meer leren over het systeem. In het proces kunnen ze verrassende ontdekkingen doen, zoals gevolgen van interacties waar niemand aan had gedacht. *Onderzoekende* testen leggen misverstanden bloot over wat de software zou moeten doen.

Testers, programmeurs of andere teamleden die *onderzoekende* testen uitvoeren, moeten openstaan om te observeren, te leren, hun kritisch denkvermogen te gebruiken en verwachtingen in vraag te stellen. Het doel van *onderzoekende* testen is om het risico te verminderen en vertrouwen te krijgen in het product. Er zijn geen scripts of lijsten met verwachte outputs. In plaats daarvan wordt een doel geïdentificeerd, met middelen (of variaties) en een missie. Teamleden maken aantekeningen terwijl ze onderzoeken en leren en debriefen later met andere teamleden en business stakeholders. Als resultaat van een *onderzoekende* testsessie kan de tester eventuele gevonden bugs aan teamgenoten laten zien of nieuwe functionaliteiten of stories voorstellen die mogelijk nodig zijn.

Onderzoekende testen is een gedisciplineerde aanpak van testen. Het moet niet worden verward met ad-hoctesten, die worden gedaan zonder enige planning of documentatie, of *monkey* testen, waarbij willekeurige invoer en willekeurige acties worden ingevoerd om te zien wat er kapot gaat. Denk aan het verschil tussen willekeurig ronddwalen (misschien verloren) en bedachtzaam onderzoeken (om inzicht te krijgen en met een doel). We zullen een paar technieken introduceren die je kunnen helpen om doelgericht te onderzoeken.

Persona's, banen, and rollen

Een nieuw product testen met een frisse blik is een geschenk! Er zijn niet zoveel vooroordelen over hoe het product zich zou moeten gedragen, en het bevestigingsvooroordeel van mensen is niet zo sterk. Een nieuw paar ogen is beter in staat om het product objectief te observeren, al heeft natuurlijk iedereen onbewuste cognitieve vooroordelen. Lisa heeft gemerkt dat wanneer ze met nieuwe testers in haar team samenzit, ze onmiddellijk de bugs opmerken die er altijd al waren, maar niemand anders kon ze "zien"!

Een persona[17], of een rol, opnemen stelt een teamlid in staat met een frisse blik een product te testen dat zij van binnen en van buiten kennen; onbewuste cognitieve vooroordelen zoals: onoplettende blindheid[18] en bevestigingsvooroordeel[19] kunnen worden overwonnen.

[17] https://www.stickyminds.com/article/how-pragmatic-personas-help-you-understand-your-end-user
[18] http://www.theinvisiblegorilla.com/gorilla_experiment.html
[19] https://en.wikipedia.org/wiki/Confirmation_bias

Een persona is een fictieve gebruiker die het team maakt met kenmerken zoals leeftijd, opleiding, ervaring, persoonlijkheid, beroep, enzovoort. Sommige teams hebben een vastgelegde set persona's die hun doelgroep vertegenwoordigen en die ze gebruiken bij het ontwerpen van nieuwe functionaliteiten. Figuur 6.1 toont een hacker persona dat je zou kunnen gebruiken om te testen.

Figuur 6.1: Hacker persona

Het combineren van persona's met banen of rollen is nog beter voor *onderzoekende* testen. Hier is een voorbeeld.

> *Jill, een directieassistente, is 30 jaar oud, heeft altijd haast met te veel zaken, ze zoekt naar sluipwegen bij het gebruiken van het product. Test de hotelreserveringsapplicatie van uw team zoals Jill, die op het laatste moment een hotel boekt voor haar baas.*

Wanneer een tester de persona van Jill aanneemt, zal ze de mogelijkheden van de functionaliteit waarschijnlijk op een andere manier gebruiken dan ze normaal zou doen. Ze kan bijvoorbeeld ontdekken dat het meerdere keren klikken op de verzendknop uit ongeduld dubbele reserveringen veroorzaakt.

Werkstromen en tours

Een veelgebruikte manier om te onderzoeken is door verwachte werkstromen of gebruikerstrajecten in de toepassing te gaan. Begin met één traject en onderzoek vervolgens variaties daarop. In die hotelboekingsapplicatie is een voor de hand liggend traject het zoeken naar een specifieke locatie en datumbereik voor een bepaald aantal gasten, het kiezen van een kamer, en het invoeren van adresgegevens om te bevestigen. Een variatie op die aanpak zou zijn om te proberen een adres uit een ander land in te voeren. Accepteert het formulier meerdere formaten voor postcode? Doet het de validatie op de postcode versus het adres?

Een andere populaire aanpak is het gebruik van tours. Vergelijk het met een tour op een reisbestemming. Als toerist, wanneer je naar Parijs gaat, wil je misschien verschillende bezienswaardigheden zien: de Eiffeltoren, het Louvre, de Arc de Triomphe of misschien zelfs de Notre Dame. Als je dat eenmaal hebt gedaan, herhaal je de tour, maar ga je in een andere volgorde naar de bezienswaardigheden. Dingen kunnen er anders uitzien! Hetzelfde gebeurt in software. Bij de [landmark tour (https://blogs.msdn.microsoft.com/james_whittaker/2009/04/06/tour-of-the-month-the-landmark-tour/) worden verschillende functionaliteiten in verschillende volgordes onderzocht wat onverwacht gedrag kan veroorzaken.

Hint: *Zoek op internet naar "onderzoekende test tours" en je zal veel verschillende ideeën vinden - we raden je aan jouw eigen ideeën te maken.*

Figuur 6.2: Landmark tour

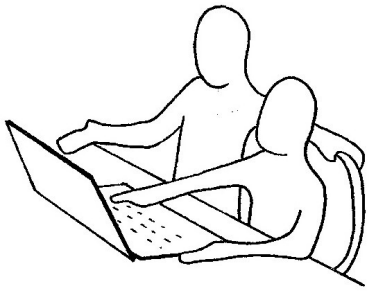
Risico's en waarde voor de klant

Teams zien vaak de bedrijfsrisico's of wat van waarde is voor een klant over het hoofd. *Onderzoekende* testsessies kunnen worden ontworpen om zich op deze aspecten te concentreren om verborgen veronderstellingen bloot te leggen. De vraag stellen: "Wat is het ergste dat er kan gebeuren?" kan een risico blootleggen dat moet worden onderzocht. Als diefstal van klantgegevens bijvoorbeeld een enorm risico is, dan wil het team extra tijd besteden aan het onderzoeken van de beveiligingsaspecten van het product.

De keerzijde van risico is waarde, dus vraag: "Wat is het beste dat kan gebeuren?" en onderzoek die bedrijfswaarde.

Onderzoek in *pairs* or groepen

Onderzoeken kan op elk moment gedaan worden. Programmeurs kunnen onderzoeken tijdens het coderen om hun feedbacklus voor visuele problemen te verkorten, of testers kunnen de browsercompatibiliteit onderzoeken naar browsercompatibiliteit. We raden echter aan om met iemand te *pairen* om het meeste uit de ervaring te halen (Figuur 6.2).



Figuur 6.3: *Pairing*

Een manier om op functionaliteitsniveau te onderzoeken, is met groepen. Eén van Lisa's teams bracht vaak mensen bij elkaar voor ad-hoc- of *onderzoekende* testsessies voor extra vertrouwen in belangrijke of risicovolle functionaliteiten. Samenwerken om gelijktijdigheidsproblemen te testen is hier een goed voorbeeld van. Meer ogen op het probleem betekent een grotere kans dat er iets wordt gevonden.

Net zoals *mob*-programmeren, kan *mob*-testen[20] worden gebruikt voor *onderzoekende* testen. Dit betekent dat er één aanvoerder is (een rol die om de paar minuten wisselt) met meerdere mensen die helpen door vragen te stellen of suggesties te maken. Meerdere perspectieven kunnen effecten op andere delen van het systeem blootleggen.

Charters

In haar boek *Explore It!* beschrijft Elisabeth Hendrickson hoe charters gebruikt kunnen worden voor effectieve *onderzoekende* testen. Charters helpen je de informatie die je nodig hebt om meer te weten te komen over uw aanvraag te organiseren, in voldoende geconcentreerde timeboxsessies. Deze werken goed in combinatie met persona's, banen en rollen.

Het sjabloon van Elisabeth ziet er als volgt uit:

> **Onderzoek** <doel>
>
> **Met** <middelen>
>
> **Om** <informatie van waarde voor iemand&gt **te ontdekken**;

Gezien de middelen (of soorten variaties, zoals Janet er graag over denkt) die zullen worden gebruikt, is dit een goede manier om te beginnen met het schrijven van een charter. Beveiligbaarheidstests moeten bijvoorbeeld verschillende *format-exploits* kunnen testen. Er kan een charter worden geschreven om verschillende pagina's in de gebruikersinterface met deze exploits te onderzoeken. Het volgende voorbeeld toont één mogelijkheid.

> **Onderzoek** de gebruikersaanmeldingspagina voor 30 minuten
>
> **Met** *cross-site scripting exploits*
>
> **Om** zwakke plekken **te ontdekken**

We houden ervan om onze sessies te time-boxen om te helpen bij het scherpstellen van het charter. Een eenvoudige manier om dit te doen, is door de tijdslimiet aan het charter zelf toe te voegen, zoals in ons vorige voorbeeld.

Lisa laat mensen graag kennismaken met onderzoekende testen door ze in kleine groepjes speelgoed en spelletjes voor jonge kinderen te laten testen. Ze creëren eerst een persona, zoals: "Judy is een zeer sterke, actieve vierjarige." Daarna testen ze een spel dat is ontworpen voor kinderen van 3-6 jaar met een charter:

> **Onderzoek** het spel als Judy
>
> **Met** al haar energie, onverwachte bewegingen, en kracht
>
> **Om te ontdekken** of het spel wel veilig is voor haar leeftijdsgroep

Als ze een klein stukje kunnen afbreken dat verstikkingsgevaar oplevert, is het spel mogelijk niet veilig. Dat is een andere manier om te testen dan wanneer je het spel gewoon als je volwassen zelf zou gebruiken.

Er zijn andere manieren om charters te schrijven. Sommige teams gebruiken mindmaps, terwijl andere ezelsbruggetjes gebruiken of gewoon een zin schrijven over wat ze willen onderzoeken.

[20] https://www.stickyminds.com/article/amplified-learning-mob-testing

Uitvoeren, leren, sturen

Mensen lopen vaak vast bij het schrijven of uitvoeren van hun eerste charter.

> ***Hint**: We raden je aan om de eerste keer niet te hard na te denken als je zich in deze positie bevindt (vastgelopen). Schrijf het op en probeer het dan. Gebruik je observatievaardigheden, je kritisch denken en je intuïtie.*

Terwijl charters worden uitgevoerd, leert de ontdekkingsreiziger en kan hij nieuwe charters schrijven. We stimuleren ook het maken van aantekeningen. Een voordeel van *pairen* is dat de persoon die niet leidt de notities kan schrijven. We zijn ook van mening dat debriefing met andere teamleden na het onderzoek de sleutel is tot het leren en delen van informatie.

Aanvullende technieken

Andere technieken helpen ons 'buiten de kaders te denken'. Mike Talks heeft bijvoorbeeld zijn "Oblique Testing"-kaarten[21] die de tester kunnen helpen op een pad dat anders niet zou zijn overwogen. De TestSphere[22]-kaarten van Beren van Daele zetten testers ook op verschillende manieren aan het denken en praten over hun testen. Als mensen hebben wij onbewuste vooroordelen en die kunnen ons ervan weerhouden om belangrijke problemen te zien. Het gebruik van dergelijke kaarten kan die vooroordelen compenseren en teams helpen creatiever te zijn.

Maak gebruik van tools voor effectief te onderzoeken

Onderzoekende testen is mensgericht, maar geautomatiseerde scripts of tools kunnen worden gebruikt om testgegevens te genereren of om de toon te zetten. Andere tools kunnen ook helpen bij het onderzoeken. Emulators kunnen bijvoorbeeld worden gebruikt voor embedded of mobiele apparaten, hoewel echte apparaten ook moeten worden getest en onderzocht. Bronnen zoals logbestanden kunnen worden gebruikt om "stille" storingen of mogelijk gegevensverlies op te sporen. Eén van Janet's teams begon bijvoorbeeld waarschuwingen te markeren om ze beter zichtbaar te maken, wat een groot probleem aan het licht bracht in de manier waarop een methode onjuist werd gebruikt. Tools zoals recorders kunnen bijhouden welke pagina's zijn bezocht of welke gegevens zijn gebruikt, zodat het kan worden afgespeeld als er iets onverwachts wordt gevonden.

[21] https://leanpub.com/obliquetesting
[22] https://www.ministryoftesting.com/dojo/series/testsphere

Chapter 6: Explore Continuously

More agile teams are finding value in exploratory testing, but it's still a new or unknown idea to many teams. We'll start by explaining the purpose of exploratory testing and what's involved.

In *Explore It!*, Elisabeth Hendrickson defines exploratory testing as "…**simultaneously** designing and executing tests to learn about the **system**, using your insights from the last **experiment** to inform the next."

In exploratory testing, a person interacts with the system and observes actual behavior, designing small experiments. Based on what they learn, they adapt the experiment and continue to learn more about the system. In the process, they may make surprising discoveries, including implications of interactions that no one had considered. Exploratory testing exposes misunderstandings about what the software is supposed to do.

Testers, programmers, or other team members who perform exploratory testing need to be open to observe, learn, use critical thinking skills, and challenge expectations. The goal of exploratory testing is to reduce risk and gain confidence in the product. There are no scripts or lists of expected outputs. Instead, a goal is identified, with resources (or variations), and a mission. Team members take notes as they explore and learn and debrief with other team members and business stakeholders later. As a result of an exploratory testing session, the tester may show any bugs found to teammates or propose new features or stories that may be needed.

Exploratory testing is a disciplined approach to testing. It is not to be confused with ad hoc testing, which is done without any planning or documentation, or monkey testing, which entails entering random inputs and random actions to see what breaks. Think about the difference between wandering randomly (perhaps lost) and exploring thoughtfully (to gain insight and with a purpose). We'll introduce a few techniques that might help you explore with purpose.

Personas, jobs, and roles

Testing a new product with a fresh set of eyes is a gift! There are not as many preconceptions about how the product should behave, and people's confirmation bias is not as strong. A new pair of eyes are better able to observe the product objectively, although of course everyone has unconscious cognitive biases. Lisa has noticed that when she pairs with new testers on her team, they immediately notice bugs that have been there all along, but nobody else could "see" them!

Assuming a persona[23], or a role, enables a team member to test a product they know inside and out with a fresh perspective; unconscious cognitive biases such as inattentional blindness[24] and confirmation bias[25] can be overcome.

A persona is a fictitious user the team creates with characteristics such as age, educational background, experience, personality quirks, profession, and so on. Some teams have a defined set of personas representing

[23] https://www.stickyminds.com/article/how-pragmatic-personas-help-you-understand-your-end-user
[24] http://www.theinvisiblegorilla.com/gorilla_experiment.html
[25] https://en.wikipedia.org/wiki/Confirmation_bias

their target customer base that they use as they design new features. Figure 6.1 shows a hacker type of persona that you could use for testing.

Figure 6.1: Hacker persona

Combining personas with jobs or roles is even better for exploratory testing. Here's an example.

> *Jill, an executive assistant, is 30 years old, always in a hurry with too much to do, and looks for shortcuts as she uses the product. Test your team's hotel reservation application as Jill, who is booking a hotel at the last minute for her boss.*

When a tester assumes Jill's persona, she's likely to use the feature's capabilities in a different way than she normally would. For example, she might discover that clicking on the submit button multiple times out of impatience causes duplicate reservations.

Workflows and tours

A common way to explore is to go through expected workflows or user journeys in the application. Start with one journey and then explore variations on it. In that hotel-booking application, an obvious journey is searching for a specific location and date range for a certain number of guests, choosing a room, and entering address information to confirm. A variation on that approach would be to try to enter an address from a different country. Does the form accept multiple formats for postal code? Does it do validation on the postal code versus the street address?

Another popular approach is to use tours. Compare it to taking a tour in a travel destination. As a tourist, if you go to Paris, you might like to see several landmarks: the Eiffel Tower, the Louvre, Arc de Triomphe, or maybe even Notre Dame cathedral. Once you've done that, repeat the tour but go to the sights in a different order. Things may look different! The same thing happens in software. Trying the landmark tour[26] uses different features and capabilities in different orders and may cause unexpected behavior.

> ***Hint***: *Search for "exploratory test tours" on the internet and you'll find many different ideas – we suggest designing your own.*

[26] https://blogs.msdn.microsoft.com/james_whittaker/2009/04/06/tour-of-the-month-the-landmark-tour/

Figure 6.2: Landmark tour

Risks and value to the customer

Teams often overlook business risks or what is of value to a customer. Exploratory test sessions can be designed to focus on these aspects to uncover hidden assumptions. Asking the question, "What is the worst thing that can happen?" may expose a risk that needs exploring. For example, if theft of customer data is a huge risk, then the team wants to spend extra time exploring the security aspects of the product.

The flip side of risk is value, so ask, "What is the best thing that can happen?" and explore around that business value.

Explore in pairs or groups

Exploring can happen at any time. Programmers may explore during coding to shorten their feedback loop for visual issues, or testers may explore for browser compatibility. However, we recommend pairing with someone to get the most out of the experience (Figure 6.2).

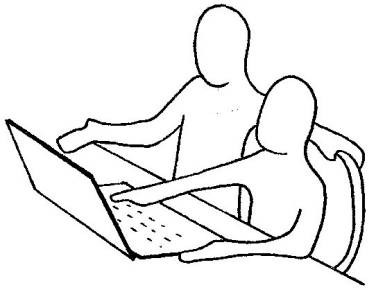


Figure 6.3: Pairing

One way to explore at the feature level is with groups. One of Lisa's teams often got people together for ad hoc or exploratory test sessions for extra confidence on major or risky features. Collaborating to test concurrency issues is a great example of this. More eyes on the problem means better chances of something being found.

Much like mob programming, mob testing[27] can be used for exploratory testing. This means there is one driver (a role that rotates every few minutes) with multiple people helping by asking questions or making suggestions. Multiple perspectives may uncover impacts to other parts of the system.

[27] https://www.stickyminds.com/article/amplified-learning-mob-testing

Charters

In her book *Explore It!* Elisabeth Hendrickson details how to use charters for effective exploratory testing. Charters help you organize the information you need to learn about your application into appropriately focused time-boxed sessions. These work well in combination with personas, jobs, and roles.

Elisabeth's template looks like this:

> **Explore** <target>
>
> **With** <resources>
>
> **To discover** <information of value to someone>

Considering the resources (or types of variations, as Janet likes to think of them) that are going to be used is a good way to start writing a charter. For example, security testing may need to test various format exploits. A charter can be written to explore several pages in the UI with these exploits. The following example shows one possibility.

> **Explore** the user signup page for 30 minutes
>
> **With** cross-site scripting exploits
>
> **To discover** any vulnerabilities

We like to time-box our sessions to help with focusing the charter. One simple way to do this is to add the time limit to the charter itself, as in our previous example.

Lisa likes to introduce people to exploratory testing by having them get into small groups to test toys and games for young children. They first create a persona, such as: "Judy is a very strong, active four-year-old." Then they test a game designed for ages 3–6 with a charter:

> **Explore** the game as Judy
>
> **Using** all her energy, unexpected movements, and strength
>
> **To discover** whether the game is safe for her age group

If they're able to break off a small piece that's a choking hazard, the game may not be safe. That's a different way to test than if you simply used the game as your adult self.

There are other ways to write charters. Some teams use mind maps, while others use mnemonics or simply write a sentence about what they want to explore.

Executing, learning, steering

People often get stuck trying to write or execute their first charter.

> ***Hint**: We suggest that if you find yourself in this position (stuck), don't think too hard the first time. Write it, then try it. Use your observation skills, your critical thinking, and your intuition.*

As charters are executed, the explorer learns and can write new charters. We also encourage notetaking. One advantage to pairing is that the person not driving can write the notes. We also believe that debriefing with other team members after the exploration is key to learning and sharing information.

Additional techniques

Other techniques help us "think outside the box." For example, Mike Talks has his "Oblique Testing" cards[28] that can help the tester down a path that might not have been considered otherwise. Beren van Daele's TestSphere[29] cards also get testers thinking and talking about their testing in different ways. As human beings, our unconscious biases can keep us from seeing important problems. Using cards like this can offset those biases and help teams be more creative.

Leverage tools for effective exploring

Exploratory testing is human-centric, but automated scripts or tools can be used to generate test data or to set the scene. Other tools can assist exploring as well. For example, emulators can be used for embedded or mobile devices, although real devices do need to be tested and explored as well. Resources like log files can be used to spot "silent" failures or potential data loss. For example, one of Janet's teams started highlighting warnings to make them more visible, which exposed a major issue in how one method was used incorrectly. Tools like recorders can keep track of what pages have been visited or what data was used, so it can be replayed if something unexpected was found.

[28] https://leanpub.com/obliquetesting
[29] https://www.ministryoftesting.com/dojo/series/testsphere

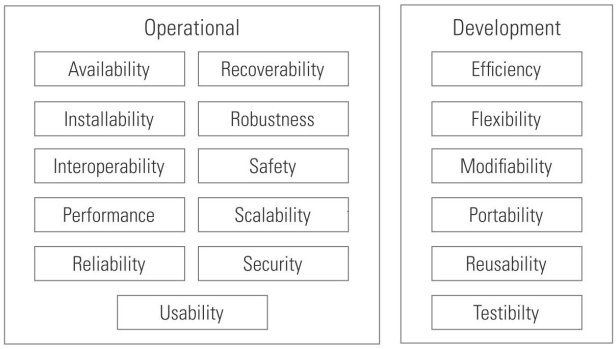
Hoofdstuk 7: Testkwaliteitskenmerken

Kwaliteitskenmerken - of zoals sommige mensen ze graag noemen, niet-functionele vereisten - worden vaak over het hoofd gezien bij het bespreken van een nieuwe functionaliteit of story. Een kwaliteitskenmerk definieert de eigenschappen waaronder een functionaliteit moet werken. In plaats van ze te beschouwen als iets dat er bijkomt, beschouwen we ze liever als een beperking waarmee het team rekening moet houden bij elke functionaliteit of story.

Kwaliteitskenmerken definiëren

Twee belangrijke soorten kwaliteitskenmerken waarmee rekening moet worden gehouden, zijn ontwikkelings- en operationele kenmerken. Ontwikkelingskenmerken omvatten onderhoudbaarheid van code, hergebruik van code en testbaarheid - het 'hoe' we onze code ontwikkelen. Dat is interne of technologiegerichte kwaliteit en is eigendom van het software opleverteam.

Als mensen het hebben over kwaliteitskenmerken, bedoelen ze meestal de operationele kenmerken. Ellen Gottesdiener en Mary Gorman classificeren enkele van de kwaliteitskenmerken in figuur 7.1 als operationele of ontwikkelingskenmerken.



Figuur 7.1: Metamodel kwaliteitskenmerken

Veel van deze kwaliteitskenmerken zijn technologiegericht (zie Hoofdstuk 9: De Agile Testkwadranten voor uitleg). Als de business stakeholders hen niet goed begrijpen, kan het opleverteam hen helpen bij het instellen van de juiste kwaliteitsniveaus voor elk kenmerk.

Risico's beperken door vroeg samen te werken

Elk product of organisatie heeft unieke behoeften en risico's die moeten worden beoordeeld. Door na te gaan welke kwaliteitskenmerken belangrijk zijn voor uw klanten, kan het team praten over de risico's voor het product. Janet werkte bijvoorbeeld in een team waar betrouwbaarheid het belangrijkste kwaliteitskenmerk

was (hoewel niet het enige). Het team vroeg zich af: “Wat hebben we nodig om de betrouwbaarheid aan te kunnen tonen?” Bij het beantwoorden van de vraag realiseerde de organisatie zich dat ze moesten investeren in een complete betrouwbaarheidstestomgeving waar de build aan het einde van elke iteratie kon worden ingezet om een complete set geautomatiseerde tests uit te voeren, samen met enkele *onderzoekende* testen. Het team werkte eraan om de geautomatiseerde tests en simulaties voor elke story te laten werken.

Sommige teams denken na over hun kwaliteitskenmerken nadat ze de functionaliteit hebben geleverd en ze creëren stories voor de “niet-functionele” vereisten. Dit is over het algemeen geen goed idee, omdat het kan betekenen dat zowel de architectuur als het codeontwerp opnieuw moeten gedaan worden. Deze kenmerken kunnen zelfs een hogere prioriteit hebben dan functionele of gedragsvereisten. Waardevolle functionaliteit in sommige gebieden kan het gebrek aan beveiliging of resultaten in andere bedrijfsdomeinen niet wegnemen. Teams die te lang wachten met het testen van deze kenmerken (vaak tijdens het eindspel net voor de release) lopen volledig vast. Dit zijn meestal ontwerpproblemen en kunnen niet zo laat in de releasecyclus nog worden opgelost.

Wees als opleverteam proactief. Wacht niet tot de product owner het gesprek begint. Ze denkt waarschijnlijk aan functionaliteiten en neemt kwaliteitskenmerken als vanzelfsprekend aan. Bedenk als team welke aspecten van kwaliteit het meest waardevol zijn voor klanten en het bedrijf. Een goede manier om te beginnen is door een contextdiagram van de voorgestelde nieuwe functionaliteit te tekenen om te zien hoe deze omgaat met andere functionaliteiten of systemen. Door de afhankelijkheden en potentieel kwetsbare gebieden in een vroeg stadium te kennen, is er voldoende tijd om de juiste ontwerpbeslissingen te nemen en ervoor te zorgen dat het team over de nodige technische kennis beschikt. Het team kan van plan zijn om een *spike* te doen (een experiment of onderzoeksstory) om mogelijke ontwerpen en architectuur te onderzoeken.

Release- of functionaliteitsplanning geeft een uitstekende gelegenheid om de business stakeholders de volgende vragen te stellen:

- Wat is het ergste dat kan gebeuren nadat we deze functionalteit releasen? Is dit dan een hoog risico?
- Is het een probleem als het systeem of de systeemfunctionaliteit voor een tijdje plat ligt? En als dit zo is, wat is de maximumtijd of percentage van de tijd dat het plat kan liggen?
- Voor een webgebaseerde app, welke browsers zouden klanten gebruiken?
- Kunnen we ervan uitgaan dat klanten mobiele apparaten gaan gebruiken? Zijn dit zowel telefoons als tablets, en is dit zowel Apple als Android? Hoe zit het met ……?
- Hoe weten we of de functionaliteit succesvol is als we deze eenmaal hebben uitgebracht?

Planning voor pre-release testen

Sommige kwaliteitskenmerken vereisen mogelijk meer tests vlak voor de release wanneer alle componenten van de functionaliteitsset zijn aangesloten. Het team kan bijvoorbeeld belasting- of prestatietests doen in een staging-omgeving om een definitieve baseline te krijgen. Als het team *blue/green deploys*[30] gebruikt in een cloudinfrastructuur, kunnen ze deze tests uitvoeren in de inactieve productieomgeving.

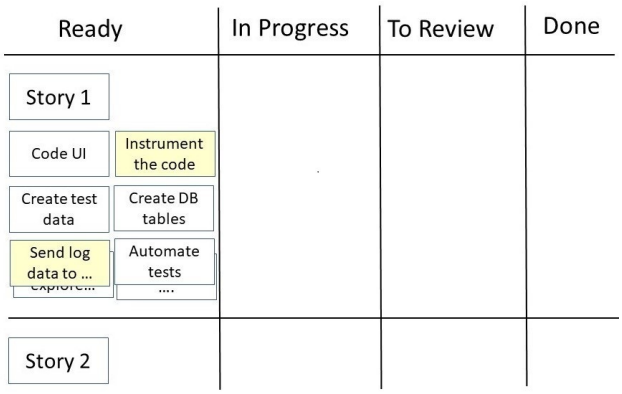
Teams kunnen releasefunctionaliteitsschakelaars en andere technieken gebruiken om nieuwe functionaliteiten voor klanten te verbergen totdat ze verschillende kwaliteitskenmerken in productie testen. Zodra ze zeker zijn van het kwaliteitsniveau voor alle aspecten van de functionaliteit, kunnen ze de functionaliteit inschakelen. (Zie meer over testen in productie in hoofdstuk 8: Testen binnen devops teams.)

[30] https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html

Plannen om later te leren

"Hoe weten we of de functionaliteit succesvol is als we deze eenmaal hebben gereleased?" – deze laatste vraag stellen is een geweldige manier voor het team om te praten over het doel van de nieuwe functionaliteit en hoe ze kunnen meten of deze aan de gewenste doelen voldoet. Provisioning data voor analysetools moeten worden gepland in de stories voor elke functionaliteit. Bedenk op storyniveau hoe het team het systeem kan monitoren om het gebruik en de kwaliteit van de kenmerk te meten. Het team heeft mogelijk nieuwe tools nodig voor de juiste logging en monitoring.

Op taakniveau moeten de programmeurs nadenken over hoe de code kan opgebouwd worden zodat het team kan meten met geautomatiseerde tests of monitoringtools (zie figuur 7.2). De oplossing kan zo simpel zijn als het uitvoeren van een optimalisatieprogramma op een nieuwe databasequery of het gebruik van een statische analysetool om te controleren of de code voldoet aan de toegankelijkheidsnormen. Een meer holistische benadering, zoals het inbouwen van elk event in de code, zodat productieproblemen snel kunnen worden gelokaliseerd en opgelost, kan geschikt zijn.



Figuur 7.2: Taakbord met de toekomst in gedachten

Naleving van de regelgeving

Het naleven van de regelgeving wordt niet altijd als een kwaliteitskenmerk beschouwd, maar net als de kwaliteitskenmerken die we hebben genoemd, moet je er vanaf het begin rekening mee houden. Regelgeving betekent niet per se stapels documenten, maar er komt meestal wel extra werk bij kijken voor het team en het is verstandig om het al vroeg te plannen.

Organisaties moeten samenwerken met auditors en regelgevende instanties om te begrijpen welke informatie nodig is om naleving aan te tonen. Het is belangrijk dat iedereen dezelfde visie heeft op een adequaat gedisciplineerde aanpak. Als het team bijvoorbeeld geautomatiseerde tests heeft die elke dag worden uitgevoerd en de tests levende documentatie opleveren in de vorm van testresultaten, kunnen die dan worden gebruikt als bewijs om de aanpak of dekking te ondersteunen? We hebben allebei gewerkt met teams die naleving moesten aantonen (medische apparatuur en financieel) en deden dat met heel weinig "extra" werk. Zie hoofdstuk 21, "Agile testen in gereguleerde omgevingen" in *More Agile Testing* voor meer voorbeelden en informatie.

Chapter 7: Testing Quality Attributes

Quality attributes – or as some people like to call them, non-functional requirements – are often overlooked when discussing a new feature or story. A quality attribute defines the properties under which a feature must operate. Rather than thinking about them as an "add-on," we prefer to think of them as a constraint the team must consider with every feature or story.

Defining quality attributes

Two main types of quality attributes that need to be considered are development and operational attributes. Development attributes include code maintainability, code reuse, and testability – the "how" we develop our code. That's internal or technology-facing quality and is owned by the software delivery team.

When people talk about quality attributes, they usually mean the operational attributes. Ellen Gottesdiener and Mary Gorman classify some of the quality attributes in Figure 7.1 as operational or development attributes.

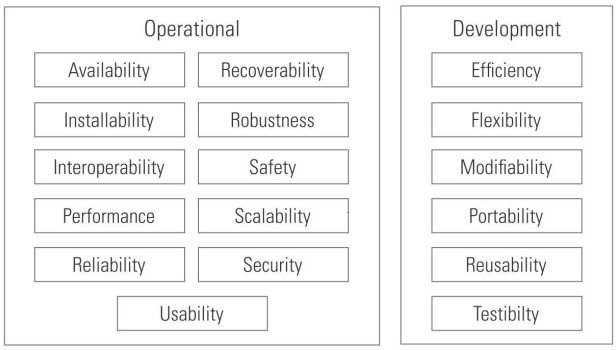


Figure 7.1: Quality attributes meta model

Many of these quality attributes are technology-facing (see Chapter 9: Agile Testing Quadrants for an explanation). If the business stakeholders don't understand them well, the delivery team can help them set the appropriate quality levels for each attribute.

Mitigating risks by collaborating early

Every product or organization has unique needs and risks that need to be assessed. By considering what quality attributes are important to your customers, the team can talk about the risks for the product. For example, Janet worked on a team where reliability was the most important quality attribute (although not the only one). The team asked themselves, "What do we need to be able to prove reliability?" When answering the question, the organization realized they needed to invest in a complete reliability testing environment

where the build could be deployed at the end of every iteration to run a complete set of automated tests along with some exploratory tests. The team worked to get the automated tests and simulations working with every story.

Some teams think about their quality attributes after they deliver the functionality, and they create stories for the "non-functional" requirements. This is generally not a good idea because it may mean having to go back and re-do the architecture or code design. These attributes may in fact be higher priority than functional or behavioral requirements. Valuable functionality in some areas doesn't overcome lack of security or performance in some business domains. Teams that wait too late in the cycle to test these attributes (often during the endgame just before release) hit a total roadblock. These types of issues are usually design issues and can't be fixed that late in the release cycle.

As a delivery team, be proactive. Don't wait for the product owner to start the conversation. She's probably thinking about feature capabilities and taking quality attributes for granted. As a team, consider which aspects of quality are most valuable to customers and the business. A good way to start is by drawing a context diagram of the proposed new feature to see how it interacts with other capabilities or systems. Knowing the dependencies and potentially fragile areas early means there is enough time to make the right design decisions and ensure the team has the necessary technical knowledge. The team might plan to do a spike (an experiment or investigation story) to explore potential designs and architecture.

Release or feature planning offer great opportunities to ask business stakeholders questions like these:

- What's the worst thing that can happen after we release this capability? Does that make it high risk?
- Is it ok if the system or a system capability is down for some amount of time? If not, what is the maximum time or percentage of time it can be down?
- For a web-based app, what browsers might customers use?
- Can we assume that customers will be using mobile devices? Would that include phones and tablets, and does that mean both Apple and android? What about ……?
- How will we know if the feature is successful once we release it?

Planning for pre-release testing

Some quality attributes might require more testing immediately before the release when all components of the feature set are connected. For example, the team may do load or performance testing in a staging environment to get a final baseline. If the team uses blue/green deploys[31] in a cloud infrastructure, they may do this testing on the idle production environment.

Teams can use release feature toggles and other techniques to hide new features from customers until they test various quality attributes in production. Once they are confident in the level of quality for all aspects of the feature, they can toggle the feature on. (See more on testing in production in Chapter 8: Testing in DevOps.)

[31] https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html

Planning for later learning

Asking that last question – How will we know if the feature is successful once we release it? – is a great way for the team to talk about the purpose of the new feature and how they can measure whether it meets the desired goals. Provisioning data for analytics tools needs to be planned into the stories for each feature. At the story level, think about how the team can monitor the system to measure usage and quality of the attribute. The team may need new tools for appropriate logging and monitoring.

At the task level, the programmers should be thinking about instrumenting the code so that the team can measure with automated tests or monitoring tools (see Figure 7.2). The solution might be as simple as running an optimizer on a new database query or using a static analysis tool to check that the code meets accessibility standards. A more holistic approach, such as instrumenting every event in the code so that production issues can be quickly pinpointed and fixed, may be appropriate.

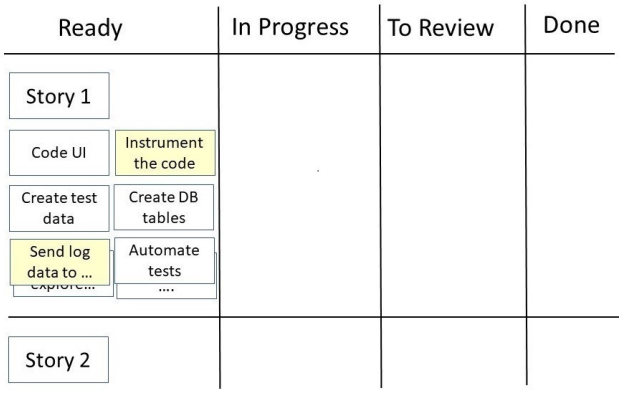


Figure 7.2: Task board with future in mind

Regulatory compliance

Regulatory compliance isn't always deemed a quality attribute, but like the quality attributes we've mentioned, you need to consider it from the beginning. Compliance does not necessarily mean stacks of documents, but there is usually extra work involved for the team, and it's wise to plan early.

Organizations need to work together with auditors and regulatory agencies to understand what information is required to show compliance. It is important that everyone has the same vision of an appropriately disciplined approach. For example, if the team has automated tests that run every day and the tests provide living documentation in the form of test results, can those be used as evidence to support the approach or coverage? Both of us have worked with teams that needed to show compliance (medical devices and financial) and did it with very little "extra" work. See Chapter 21, "Agile Testing in Regulated Environments" in *More Agile Testing* for more examples and information.

Hoofdstuk 8: Testen binnen devops teams

Algemeen gesproken gaat 'software development' over het continue proces van in productie zetten van nieuwe wijzigingen aan de software voor de gebruikers. In het verleden waren dit voornamelijk manuele taken. Tegenwoordig bestaan er nieuwe hulpmiddelen en technieken om software producten te ontwikkelen en om ze te naar een testomgeving en productieomgeving te brengen. Het basis principe blijft hetzelfde: de development teams hebben veel verschillende test activiteiten met als doel het vertrouwen winnen in de gemaakte wijzigingen aan hun product in productie. Men beschikt momenteel over een nieuwe terminologie voor deze test activiteiten, uiteraard blijven ook hier de basis testvaardigheden relevant.

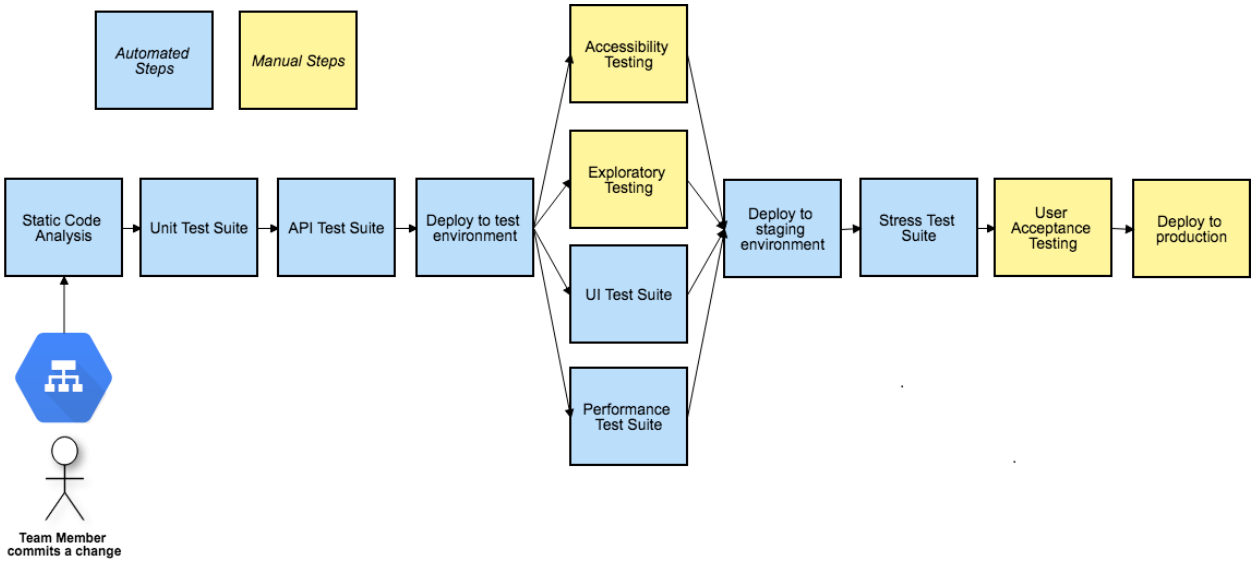
De devops beweging is gegroeid vanuit het idee dat sommige organisaties agile willen werken maar vergeten om hun volledige operationeel personeel te betrekken binnen de transitie. Daarnaast is het ook het resultaat van de verandering naar cloud gehoste toepassingen en infrastructuur als code vervangende 'command line interface'. Rollen worden aangepast en aan de operationele specialisten leert men programmeren. De developers nemen verantwoordelijkheid over hun eigen code zelf wanneer deze al in productie staat in plaats van het over de haag te gooien richting het operationeel personeel.

De test analisten passen ook hun eigen vaardigheden en activiteiten aan. Zo zullen zij proberen om op verschillende manieren bij te dragen aan het team door bijvoorbeeld geautomatiseerde testsets te ontwikkelen die betrouwbare en waardevolle informatie kunnen aanleveren. Daarnaast kunnen ze ook helpen met het optimaliseren van de pijnlijn van de opleveringen en het testen van de infrastructuurcode om zo een betrouwbare uitvoering te garanderen.

Hoofdstuk 23 in *More Agile Testing* gaat meer in detail over hoe operationele specialisten het opleverteam kunnen helpen om de kwaliteit te verbeteren bijvoorbeeld door het opzetten van test omgevingen, het helpen implementeren van geautomatiseerde test kaders, het genereren van test data en vele andere taken.

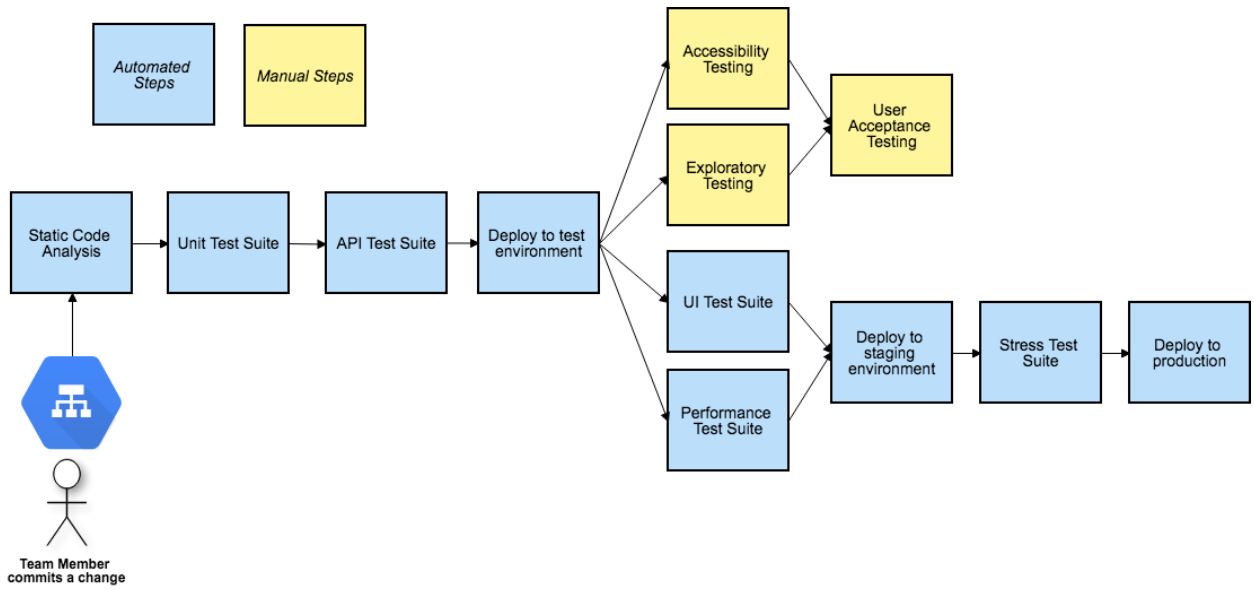
Het voortdurend opleveren en naar omgevingen brengen met een pijplijn

Teams die 'continious delivery (CD)' gebruiken hebben telkens een kandidaat als bruikbare release elke keer men een nieuwe wijziging aan de code bibliotheek aanbrengt en wanneer deze succesvol door de implementatie pijplijn gaat. De pijplijn start met continue integreren, dit kan geautomatiseerde testsuites op verschillende test levels omvatten zoals unit testen, API testen en het volledig testen van de workflow door middel van de gebruikerinterface. Dit kan verschillende stappen bevatten zoals statische code-analyse voor geautomatiseerde implementaties in de verschillende test omgevingen. Op basis van deze gegevens kunnen de zakelijke stakeholders beslissen om deze 'release kandidaat' in productie te zetten. Dit kan op meerdere momenten per dag. Figuur 8.1 toont een voorbeeld van een continue leveringspijplijn.



Figuur 8.1: Continue leveringspijplijn

'Continious deployment (ook CD)' is hetzelfde proces met als verschil dat elke succesvolle releasekandidaat automatisch wordt geïmplementeerd in productie. Figuur 8.2 toont een voorbeeld van het continue naar een omgeving brengen met een pijplijn.



Figuur 8.2: Het continue naar omgevingen brengen met een pijnlijn

Dit klinkt even eng als het uitvoeren van testen in productie indien je dit nog nooit gedaan hebt. Maar als je meerdere keren per dag een release uitvoert, hoe kan je dan alle testen doen die je moet doen? Makkelijk: het manueel uitvoeren van testen (menselijk werk), door het ontbreken van geautomatiseerde testen of in het kader van verkennende testen en toegankelijksheidstesten, zijn evenzeer een onderdeel van

de leveringspijplijn net zoals de testen die wel geautomatiseerd zijn.

De sleutel is het herkennen van **het verschil tussen het naar omgevingen brengen van code en het vrijgeven van code**.

Dankzij verschillende technieken zoals het in- en uitschakelen van bepaalde functionaliteiten is het mogelijk om wijzigingen te verbergen voor klanten totdat men alle noodzakelijke testen kan uitvoeren. Het testen zelf kan asynchroon gedaan worden.

'Continuous delivery' of 'continuous deployment' toont aan dat het team op een hoog niveau kan presteren. Noodzakelijk hierbij is dat alle personen binnen het opleverteam en alle zakelijk stakeholders een gedeeld begrip hebben van elke functionaliteit en van elke story die moet opgeleverd worden. Daarnaast beheersen de ontwikkelaars verschillende manieren om bepaalde functionaliteiten voor sommige (of voor alle) gebruikers te verbergen totdat ze klaar zijn om opgeleverd te worden. De leveringspijplijn moet vlot verlopen zodanig dat het mogelijk is om dagelijks of zelfs meerdere keren per dag de code over te brengen naar productie. Om dit te verwezenlijken is een goede infrastructuur zeer belangrijk, met andere woorden: meer beschikbare en kwalitatieve code om op verder te bouwen en om op te testen. Daarnaast zijn er verschillende nieuwe vaardigheden nodig om dit alles te beheersen. Wanneer elk lid van het team zijn 'T-vormige' vaardigheden[32] aanbrengt en iedereen vlot samenwerkt, dan kan het team makkelijker problemen oplossen en werken aan het verkorten van hun feedbackloop van hun eigen pijplijn.

Testen in productie

In het verleden klonk de term "testen in productie" als "zet de code in productie en laat de gebruikers de defecten vinden en laat ze het ons zelf vertellen" of "laten we alles testen in productie en hopen dat we niemand tijdens het gebruik beïnvloeden". Helaas werd dit in het verleden door sommige teams zo gedaan. Tegenwoordig hebben de woorden "testen in productie" een minder beangstigende connotatie. Testen in productie is zelfs in vele gevallen een noodzaak geworden. Maar hiermee bedoelt men helemaal niet dat men een kwalitatief mindere code in productie moet zetten en de gebruikers zelf de bugs moeten laten ontdekken.

Het uitvoeren van testen in productie helpt bedrijven op verschillende manieren. Vaak is het onmogelijk om een testomgeving op te zetten die er uitziet net zoals in productie. Algemeen genomen is er geen makkelijke manier om te weten te komen hoe software zal reageren todat het in productie staat.

Technieken zoals het in- en uitschakelen van functionaliteiten maken het mogelijk dat teams bepaalde functionaliteiten 'aanzetten' voor een specifieke doelgroep met als doel het snel verkrijgen van feedback. Men geeft dit soms namen zoals 'leerrelease' of 'minimaal levensvatbaar product'. Het A/B testen is hiervan de meest gekende test techniek om te testen in productie. Hierbij toont men verschillende ontwerpen aan verschillende mensen en laat men deze personen beoordelen bij welke men het vaakst doorklikt of welke de grootste verkoop kan genereren.

Er bestaan geavanceerde analyse technieken en traceringsmethoden die details tonen over hoe een individuele gebruiker navigeert doorheen de applicatie of men kan geaggregeerde statistieken genereren zoals het percentage van klanten die een bepaalde functie gebruikt. Het verwijderen van ongebruikte functies kan net zo belangrijk zijn als het toevoegen van nieuwe populaire functies. Dit omdat elke regel van code die men schrijft bepaalde onderhoudskosten heeft en risico's toevoegt. Het uitvoeren van testen in productie gaat over monitoren en observeren.

[32] https://lisacrispin.com/tag/t-shaped-skills/

Monitoren en observeren

Het belang van het monitoren van de gezondheid van het systeem in productie bestaat al even lang als software systemen relevante informatie registeren over gebeurtenissen in de systemen zelf. Teams kunnen waarschuwingen instellen voor bepaalde soorten fouten of wanneer bepaalde foutbudgetten overschreven worden; zoals "Plaats een waarschuwing wanneer het aantal 503 fouten stijgt tot met 10% ten opzichte van het gemiddelde." Wanneer men een waarschuwing ontvangt, kunnen de teamleden dit opzoeken in de logbestanden en analyse doen om het probleem in kaart te brengen. Dit met het oog op het vinden van een oplossing voor het probleem.

Professionele testers weten dat het niet mogelijk is om alle fouten die in productie kunnen optreden vooraf op te sporen. De afgelopen jaren gaf dit de aanleiding tot het ontstaan van een nieuwe praktijk met als naam 'observeerbaarheid'. Vaak duid men dit aan als "o11y", wat staat voor de 11 tekens tussen de 'o' en de 'y' in het Engelse alfabet. Teams die het o11y instrument gebruiken, registreren elke gebeurtenis in hun productie code afzonderlijk zodat het kan geanalyseerd worden met de juiste hulpmiddelen indien nodig. Door het gebruik van deze instrumenten en experimenten, kunnen teams informatie bestuderen uit gestructureerde log bestanden, statistieken en andere traceerbare info. Op deze manier leren ze andere zaken over hun product dan deze die ze kunnen leren van de testen in een testomgeving. Dit is een omgeving waar testers, met hun vaardigheden om ongewone patronen te identifiëren en risico's te spotten, hun meerwaarde zichtbaar maken. Zoals Abby Bangser[33] aangeeft, is dit een soort van verkennend testen (exploratory testing) ondersteund door een tool.

Observeerbaarheid stelt teams in staat om snel te reageren wanneer een gebruiker een probleem rapporteert die het team nog nooit eerder gezien heeft of wanneer de systeemstatistieken ongewonen patronen tonen die wijzen op een mogelijk probleem. De gestructureerde loggegevens maken het mogelijk voor het team om gebruikersactiviteiten te traceren en snel te identificeren waar het systeem zich verkeerd begon te gedragen. Hierdoor kunnen teams een wijziging ongedaan maken of een fix in productie zetten. Dit kan gebeuren binnen enkele minuten. Het vermogen om zo snel in te grijpen bij productiestoringen zorgt ervoor dat het team continue kleine wijzigingen durft in productie te zetten zonder grote angst. Dit alles zorgt ervoor dat continue oplevering of implementatie in productie als een veilige manier van werken kan gezien worden.

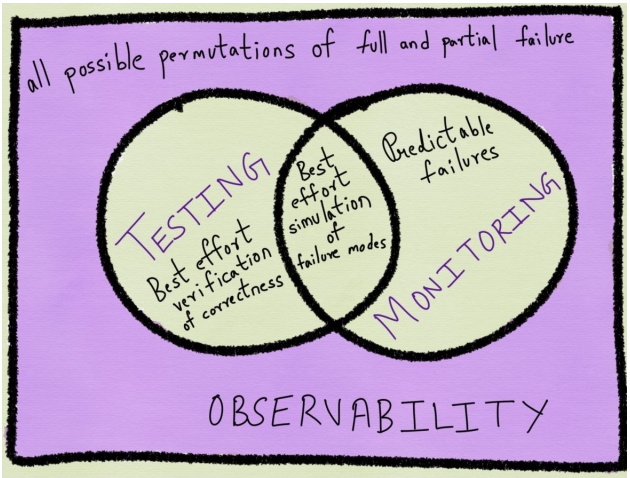
Nieuwe technologie brengt ons nieuwe mogelijkheden

In de afgelopen jaren is "big data"-opslag betaalbaar geworden voor kleine bedrijven. Hierdoor kunnen teams informatie bijhouden over elke gebeurtenis die men detecteert in een testomgeving of in productie. Krachtige technologie, inclusief Artificiële Intelligentie (AI) en Machine Leren (ML) kunnen versneld deze grote hoeveelheid data verwerken en analyseren. Hierdoor kunnen wij snel leren hoe klanten onze producten gebruiken, problemen ontdekken voor klanten ze opmerken en snel oplossingen voorzien voor problemen in productie.

Testen uitvoeren in productie is slechts een deel van de volledige aanpak die teams gebruiken om kwaliteit in te bouwen voor hun product. De teams plannen en voeren nog steeds alle nodige test activiteiten uit samen met het schrijven van de code voor productie. Daarnaast kunnen ze ook het productiegebruik observeren en risicovrije testen uitvoeren in productie om een extra niveau van vertrouwen en betrouwbaarheid toe te

[33] https://club.ministryoftesting.com/t/power-hour-curious-stuck-or-need-guidance-on-devops-or-observability/24963/3

voegen. Figuur 8.3 is een mooi voorbeeld van Cindy Sridharan uit haar artikel *Monitoring and Observability*[34], en geeft weer hoe teams proberen om hun productieomgeving te stimuleren in hun testomgevingen, hoe ze de productieomgeving monitoren om voorspelbare storingen waar te nemen, en observeerbaarheid inzetten om over het hoofd geziene problemen op te vangen die toch gebeuren ondanks alle inspanning in functie van kwaliteit.



Figuur 8.3: Cindy Sridharan, testen, monitoren, and observeerbaarheid

[34] https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

Chapter 8: Testing in DevOps

Software development has always included the process of getting new changes to the software into production for customers to use. In the past, much of this process was manual. There are new tools and practices for creating software artifacts and deploying them into test and production environments. But the basic process is the same. Teams have many testing activities to help them feel confident about the changes they make to their production product. There are new terms for some of this testing, but basic testing skills are still relevant.

The DevOps movement grew out of the idea that some organizations had embraced agile development but left their entire operations staff out of the transition. It's also a result of the move to cloud-hosted applications and infrastructure as code replaced command line interfaces. Roles have adapted. Operations specialists learn how to code. Programmers take responsibility for their code even after it is in production, instead of throwing it over the wall to operations.

Testers also adapt their own skills and activities. They contribute in many ways, such as helping to design the automated test suites that provide reliable and valuable information, optimizing the delivery pipeline, and testing infrastructure code to ensure reliable execution.

Chapter 23 in *More Agile Testing* goes into detail about how operations specialists can help the delivery team improve quality by setting up test environments, helping to implement test automation frameworks, generating test data, and more.

Continuous delivery and deployment

Teams practicing continuous delivery (CD) have a deployable release candidate each time a new change is committed to the code repository and subsequently passes successfully through a deployment pipeline. The pipeline starts with continuous integration, which may include automated test suites at different levels such as unit, API, and full workflow through the user interface. It may include other steps such as static code analysis for automated deployments to various environments. The business stakeholders can decide to deploy the release candidate to production and may do so many times per day. Figure 8.1 shows an example of a continuous delivery pipeline.

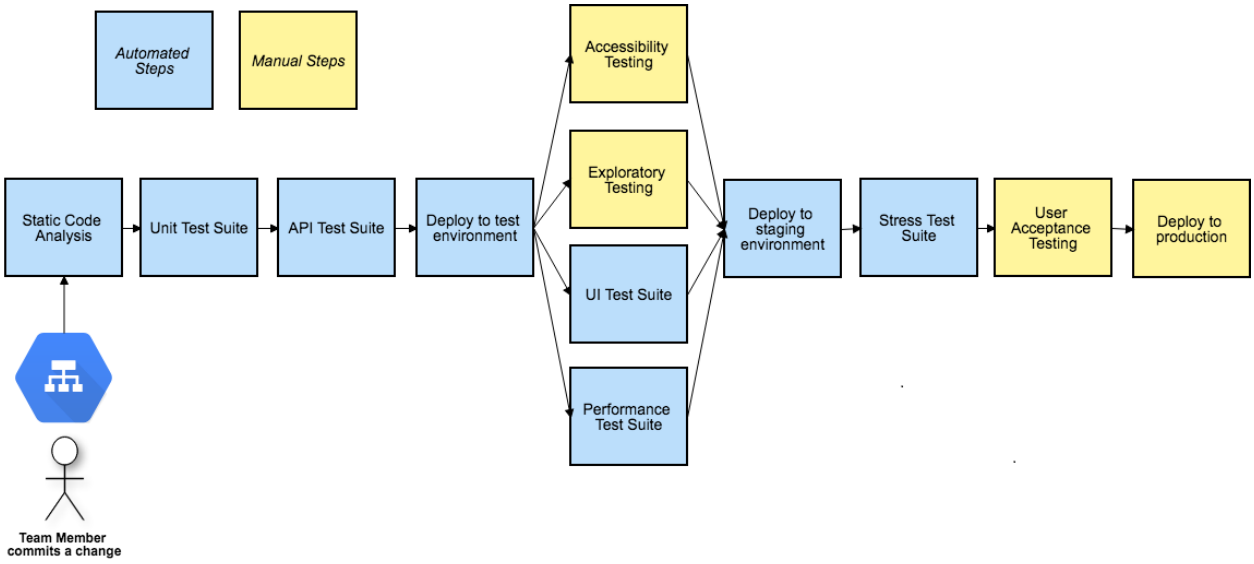


Figure 8.1: Continuous delivery pipeline

Continuous deployment (also CD) is the same process, except that each successful release candidate automatically deploys to production. Figure 8.2 shows an example of a continuous deployment pipeline.

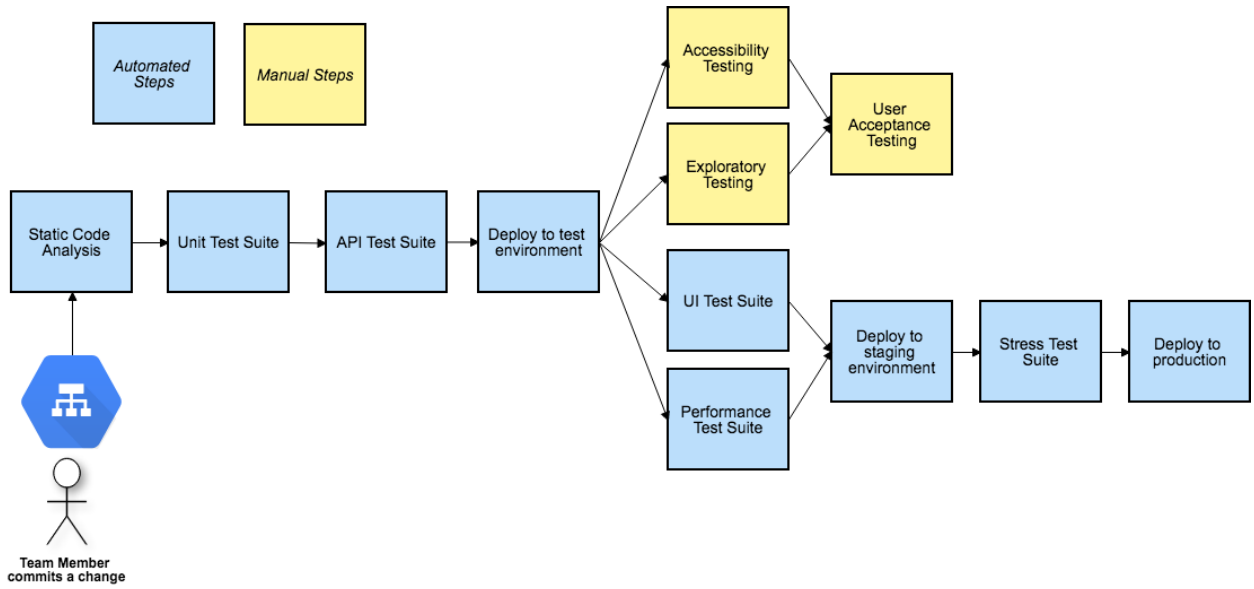


Figure 8.2: Continuous deployment pipeline

This sounds as scary as testing in production if you haven't done it before. If you release multiple times per day, how can you possibly fit in all the testing that you need to do? Human-centric testing, whether tests the team hasn't automated yet or testing activities like exploratory and accessibility testing, is as much a part of a delivery pipeline as automated tests.

The key is recognizing **the difference between deploying and releasing**.

Thanks to techniques such as release feature toggles, it's possible to hide changes from customers until all necessary testing is completed. Testing can be done asynchronously.

Continuous delivery or deployment represents a high standard of achievement for a team. Delivery team members and business stakeholders need to create a shared understanding of each feature and story to be delivered. The developers master ways of hiding features from some (or all) customers until they are ready to release that feature. The delivery pipeline must be fast in order to potentially deploy daily or multiple times per day. That requires a good infrastructure, which means more code to build and test. There are many new skills to master. When each team member brings their T-shaped skills[35] and everyone collaborates, the team can more easily solve problems and continue to shorten their feedback loops represented in the pipeline.

Testing in production

At one time, the term "testing in production" sounded like "Throw the code into production and let the customers find the bugs and tell us," or "Let's test in production and hope that we don't affect anybody's account." Sadly, some teams did just that. Today, the words "testing in production" have a less fearful connotation. Testing in production has become a necessity in many cases, but it does not mean releasing poor quality code and letting customers find the bugs!

Testing in production helps companies in multiple ways. It's usually impossible to create a test environment that looks exactly like production. There is no easy way to really know what the software will do until it's in the production environment.

Techniques such as release feature toggling enable teams to "turn on" specific functionality to specific customers to get quick feedback. Sometimes this is called a learning release or minimal viable product (MVP). A/B testing may be the most well-known testing-in-production technique, showing different designs to different people and judging which lead to the most "click-throughs" or sales.

Sophisticated analytics and tracing can show details such as how an individual user navigates through the application, or aggregate statistics like what percentage of customers are using a specific capability. Removing unused features can be just as important as adding popular new ones, since every line of code has a maintenance cost and adds risk. Testing in production is about monitoring and observing.

Monitoring and observability

The importance of monitoring system health in production has existed as long as software systems have logged pertinent information about events in the system. Teams can set up alerts for certain types of errors or for exceeding an error budget – for example, "Post an alert when the number of 503 errors goes up by 10% over average." When alerted, team members dig into the log files and analytics to investigate the problem so they can resolve it.

Professional testers know that it's not possible to predict all the possible errors that can occur in production! In recent years, this has given rise to a new practice called observability, often referred to as "o11y," representing the 11 characters between the 'o' and the 'y' in the English alphabet. Teams that practice o11y instrument

[35] https://lisacrispin.com/tag/t-shaped-skills/

and log every single event in their production code so it can be analyzed by appropriate tools when needed. Using tools and experiments, teams study information from structured log data, metrics, and traces to learn different things about their product than what can be learned in a test environment. This is an area where testers, with their ability to notice unusual patterns and identify risks, contribute value. As Abby Bangser[36] has said, it is a form of tool-assisted exploratory testing.

Observability lets teams respond quickly when a user reports a problem that the team has never seen before, or when system metrics show unusual patterns indicating a potential problem. The structured log data enables the team to trace user activity and quickly identify where the system started behaving incorrectly. This allows teams to revert a change or deploy a fix, often in a matter of minutes. This ability to recover so quickly from production failures lets the team release these continual small changes to the product fearlessly, and it's what makes continuous delivery or deployment a safe practice.

New technology brings us new capabilities

In the past few years, "big data" storage has become affordable even for small companies. Teams can log information about every event that occurs in any test or production environment. Powerful technology, including artificial intelligence (AI) and machine learning (ML), has sped up processing and analysis of these huge amounts of data. We can quickly learn how customers are using our products, discover problems before they do, and quickly recover from failures.

Testing in production is just one part of any team's approach to building quality into their product. They still plan and execute all the appropriate testing activities along with writing the production code. They can also observe production use and run risk-free tests in production to add another level of confidence and reliability. Figure 8.3 is a wonderful graphic by Cindy Sridharan from her article *Monitoring and Observability*[37], and represents how teams test trying to simulate production, monitor for predictable failures in production, and use observability to catch anything that slips through those other quality-driven efforts.

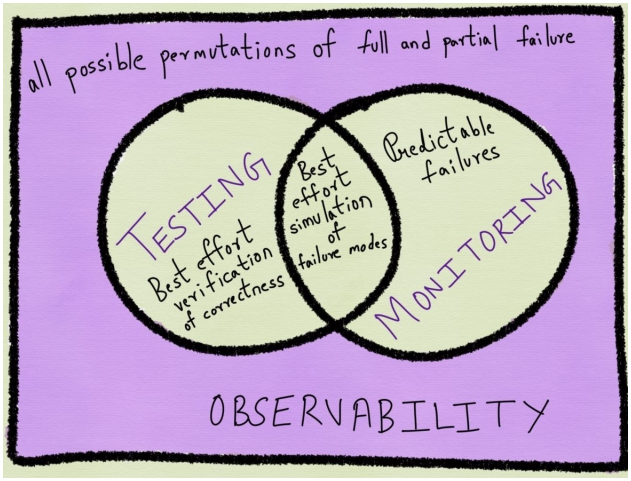


Figure 8.3: Cindy Sridharan's testing, monitoring, and observability

[36] https://club.ministryoftesting.com/t/power-hour-curious-stuck-or-need-guidance-on-devops-or-observability/24963/3
[37] https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

DEEL 3: Handige Modellen

Wij hebben ontdekt dat visuele modellen een essentieel ingrediënt zijn om teams te helpen bij het plannen en uitvoeren van alle noodzakelijke testactiviteiten en het formuleren van een effectieve testautomatiserings-strategie. In deze sectie leggen we uit hoe je de agile testkwadranten kan gebruiken om alle soorten tests te identificeren die nodig zijn voor elke nieuwe functionaliteitsset of story, en om ervoor te zorgen dat ze worden uitgevoerd wanneer ze het meest effectief zijn. Daarna bekijken we hoe we visuele modellen zoals de testautomatiseringspiramide kunnen gebruiken om teamgesprekken te begeleiden die gaan over een realistische automatiseringsstrategie binnen hun specifieke context.

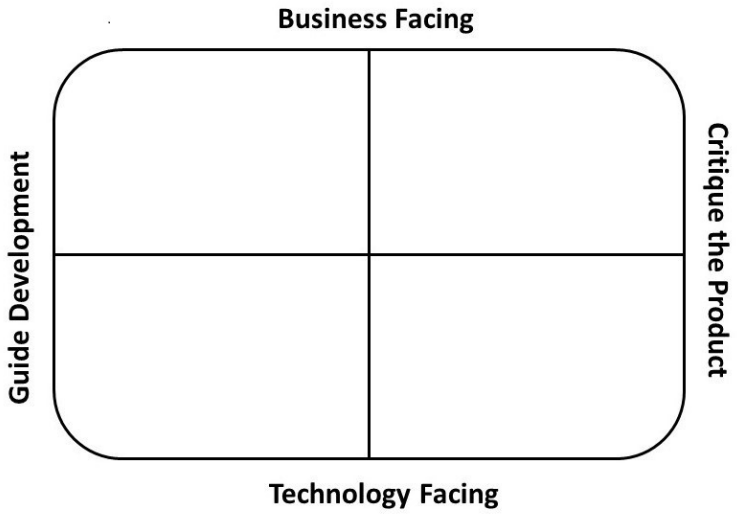
SECTION 3: Helpful Models

We have found visual models an essential ingredient in helping teams plan and execute all necessary testing activities and formulate an effective test automation strategy. In this section, we explain how to use the Agile Testing Quadrants to identify all the types of tests that are needed for each new feature set or story, and make sure they are done when they are the most effective. Then we look at how to use visual models such as the Test Automation Pyramid to guide team conversations about a realistic automation strategy that works for their context.

Hoofdstuk 9: De Agile Testkwadranten

Brian Marick beschreef als eerste de agile testkwadranten (Figuur 9.1) in 2003, toen Lisa en Janet probeerden uit te zoeken hoe over testen te praten met de agile gemeenschap, en specifiek met de testers. Op dat moment behandelden de agilisten enkel "klant testen" en "programmeur testen." Veel teams focusten enkel op functionele acceptatietesten – hoe de functionaliteiten zich moesten gedragen. Vandaag de dag is dit nog steeds een potentiële valkuil voor teams in de overgang naar agile. De kwadranten gaven hen een manier om alle verschillende soorten testen te behandelen die een team mogelijk dient te overwegen. Ze helpen hen om het grote geheel te zien.

De agile testkwadranten zijn een taxonomie van verschillende types van testen. We gebruiken ze als denkgereedschap om teams te helpen bepalen welke testactiviteiten ze zouden kunnen nodig hebben en om zeker te zijn dat ze over de juiste mensen, gereedschappen en omgevingen beschikken om ze uit te voeren.



Figuur 9.1: De Agile Testkwadranten

Testen aan de linkerkant zijn degene die de ontwikkeling in goede banen leiden, degene die geschreven worden voordat code geschreven wordt of terwijl de codering vordert. De testen aan de rechterkant zijn degene die het product beoordelen (evalueren) wanneer de code klaar is. Testen links helpen defecten te voorkomen. Testen rechts vinden defecten in de code of identificeren misschien ontbrekende functionaliteiten.

De bovenste helft van de kwadranten focust op testen die leesbaar zijn door business stakeholders. Deze testen beantwoorden de vraag, "Zijn we het juiste ding aan het bouwen?" De onderste helft omvat testen die geschreven zijn door en voor technische teamleden. Voor de business stakeholders zijn waarschijnlijk de eindresultaten belangrijk, ze zullen niet proberen de testen te lezen. De bovenste helft gaat over externe kwaliteit zoals bepaald door de business. De onderste helft gaat over de correctheid van de interne code of de infrastructuur.

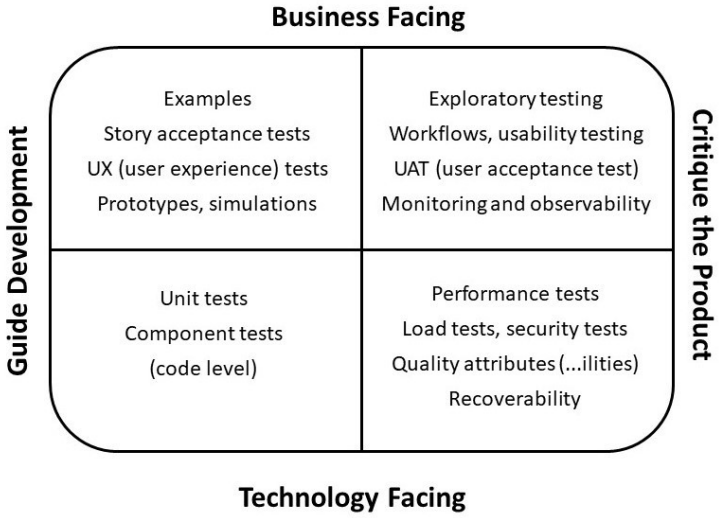
De kwadranten zijn genummerd voor het gemak. De vier kwadranten zijn:

- **Kwadrant 1** (K1): Technologiegerichte testen die de ontwikkeling in goede banen leiden
- **Kwadrant 2** (K2): Businessgerichte testen die de ontwikkeling in goede banen leiden
- **Kwadrant 3** (K3): Businessgerichte testen die het product beoordelen
- **Kwadrant 4** (K4): Technologiegerichte testen die het product beoordelen

Het agile testkwadrantenmodel helpt teams doordenken over de testactiviteiten die nodig zijn om vertrouwen te geven in het product dat ze bouwen. Het helpt ook een gemeenschappelijke testtaal te ontwikkelen met het team en de organisatie wanneer het wordt gebruikt om te communiceren tussen teams. Janet is fan van dit model omdat het niet enkel een holistisch beeld geeft van testen maar dat het ook zichtbaar maakt dat het ganse team verantwoordelijk is voor die testactiviteiten.

Elk team heeft zijn eigen verschillende combinatie van business domein, product, vaardigheden, productmaturiteit, technische stack, regelgevend toezicht, en meer. Het model kan toegepast worden om de testen die nodig zijn in elke context weer te geven. In Figuur 9.2 delen we enkele typische types van testen voor elk kwadrant.

> ***Hint***: *Onthoud: Gebruik deze voorbeelden als richtlijnen. Het is een gereedschap, geen regel, en er zijn heel wat grijze zones. De context van jullie team is uniek, en jullie testkwadranten moeten dat ook zijn. Pas het model toe voor de testen die jullie moeten doen.*



Figuur 9.2: Voorbeelden van testing types voor de kwadranten

Welke testen in welke volgorde?

We hebben de kwadranten 1, 2, 3, 4 genummerd voor het gemak. Omdat het een hele mond vol is om te zeggen "Businessgerichte testen die de ontwikkeling in goede banen leiden," spreken we hierna van "Kwadrant 2" of "K2." De nummers zijn niet bedoeld om aan te geven dat de types van testactiviteiten moeten gebeuren in die

volgorde. Terwijl het team zijn testen plant, dienen ze na te denken over het gepaste moment om elk type testactiviteit uit te voeren.

Een voorbeeld. Een team beslist om met een nieuwe architectuur te starten voor nieuwe functionaliteiten in de toekomst. Ze moeten zeker zijn dat de architectuur op een aangepaste manier zal schalen om voldoende gebruikers aan te kunnen en om een zekere belasting van het systeem aan te kunnen. Het team voert een "spike" uit wat betekent dat ze een beetje weggooicode schrijven, enkel en alleen met als doel de nieuwe architectuur te testen. Ze voeren belastings- en performantietesten uit op de "spike" om te kijken of de architectuur tegemoetkomt aan de vereisten qua antwoordtijden en stabiliteit. Als ze tevreden zijn, verwijderen ze de spike en beginnen te werken aan stories voor een nieuwe functionaliteit, waarbij ze deze keer hun gewone ontwikkelingspraktijken volgen. Als ze niet tevreden zijn, dienen ze mogelijk de architectuur te her-denken en het proces te herhalen.

Voor de ontwikkeling van functionaliteiten starten de meeste teams waarschijnlijk in K2. Misschien tekenen ze prototypes op papier en tonen ze aan potentiële klanten. Het oplever- en business team heeft misschien een story-mapping of specificatie workshop om functionaliteiten te behandelen en in stories te snijden. Tijdens de backlog verfijning of story-gereedheid workshops kan het team activiteiten zoals example mapping gebruiken om meer details naar te boven te brengen.

Zodra het team start aan de ontwikkeling van een story, werken ze aan testen van K1, door unittesten te schrijven als onderdeel van testgedreven ontwikkeling. Tegelijkertijd kunnen ze werken aan storytesten in K2. Van zodra voldoende ontwikkeld is van een functionaliteit om te onderzoeken, beginnen ze aan K3 testing. Nochtans als beveiliging de nummer één concern is voor zowel business als de klanten, dan kunnen beveiligingstesten van K4 geprioriteerd worden boven functionele storytesten van K2. Elk team vindt zijn eigen werkwijze, en die kan anders zijn voor verschillende types functionaliteiten of projecten.

De kwadranten gebruiken

Janet en Lisa hebben teams de kwadranten zien gebruiken op veel verschillende manieren. Sommige teams hangen een poster op de muur met de lege kwadranten. Wanneer ze een nieuwe functionaliteit of release plannen, bespreken ze alle testen die ze nodig hebben en noteren elk type in het gepaste kwadrant. Ze gebruiken het om hen eraan te herinneren wat ze moeten doen of wat ze zouden kunnen vergeten zijn.

Kwadrant 1

Teams kunnen de kwadranten ook gebruiken om te bespreken welke testen ze moeten automatiseren. K1 testen worden typisch geautomatiseerd door de ontwikkelaars van de productiecode. Veel teams werken volgens testgedreven ontwikkelingspraktijken (TDD), waarbij ze een kleine unittest schrijven voor een klein stukje functionaliteit, en nadien de code toevoegen die ervoor moet zorgen dat de test slaagt. K1 testen zijn ontworpen om snel uit te voeren omdat ze maar een klein stukje code testen en gewoonlijk geen interacties omvatten met andere lagen van de applicatie of databases. Ze geven teams de snelle feedback die nodig is om snel en zonder angst wijzigingen te kunnen maken aan de code.

Kwadrant 2

De businessgerichte testen van K2 die de ontwikkeling in goede banen leiden zijn een belangrijk fundament voor de meeste teams. Product owners, ontwikkelaars, testers, en anderen ontmoeten elkaar vaak om functionaliteiten en stories te plannen. Ze gebruiken daarbij technieken zoals example mapping om business regels uit te klaren voor elke story, samen met de voorbeelden die ze illustreren. Teams die gedragsgestuurde ontwikkeling (BDD), door acceptatietesten gestuurde ontwikkeling (ATDD), of specificatie door voorbeelden (SBE) beoefenen, vertalen deze naar scenarios die gedrag specifiëren als uitvoerbare testen. Deze scenarios kunnen worden geautomatiseerd terwijl de code wordt geschreven.

Kwadrant 3

Testen in K3 zijn eerder mens-centraal, en leren of de ontwikkelde stories en functionaliteiten de beoogde waarde opleveren aan klanten. Automatisatie kan daarbij gebruikt worden om dit soort testen te vereenvoudigen door het voorzien van data of status. Het is een vaak gebruikte werkwijze om exploratory testen te doen in productie, gebruik makend van feature toggles om nieuwe functionaliteiten te "verbergen" voor klanten tot het testen is voltooid. K3 testen omvatten testen in productie zoals monitoring analyses om te leren wat er echt gebeurt en hoe klanten de functionaliteiten gebruiken. Wat geleerd wordt van K3 testen keer vaak terug naar K2, en resulteert dan in het maken van nieuwe stories of functionaliteiten.

Kwadrant 4

Veel K4 testen zijn gebaseerd op automatisatie en gereedschappen, maar sommige aspecten kunnen bijkomende testen vereisen. Bijvoorbeeld, de geautomatiseerde gereedschappen om toegankelijkheidstesten uit te voeren, zijn nog steeds niet zo effectief als manuele exploratory testen voor dit kwaliteitskenmerk. Resultaten van deze testen keren vaak terug als K1 activiteiten wanneer het team het code ontwerp wijzigt om verschillende kwaliteitskenmerken te verbeteren. Prestatiemonitoring en fouten in productie, of herstelbaarheidstesten kunnen ook beschouwd worden als een type van testen dat in K4 thuishoort. De technologie van vandaag heeft het haalbaar en veilig gemaakt om in productie te testen. Dit betekent niet dat we klanten bugs voor ons laten ontdekken maar dat we met zekerheid leren hoe de code zich gedraagt in een echte productie-omgeving. (Details over testing in productie zijn te vinden in Hoofdstuk 8.)

"Klaar" definiëren

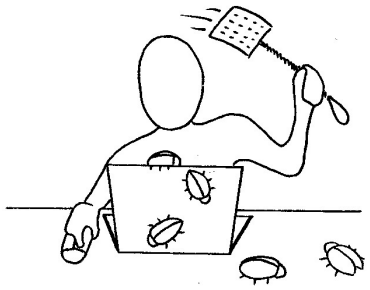
Gebruik de kwadranten om te bepalen wat "Klaar" betekent voor jouw team. Veel teams hebben het moeilijk wanneer ze proberen te beslissen welke testen ingesloten moeten worden in die definitie. In Hoofdstuk 3 spraken we over niveaus van detail, en die niveaus worden hier toegepast. In plaats van "Klaar" te definiëren, sporen we teams aan om meer specifiek te zijn en te spreken van "**Story Klaar**". Testen die "Story Klaar" bepalen, omvatten waarschijnlijk alles van K1, alles van K2, en misschien wat van K3 zoals exploratory testen.

Als we een stap verder gaan en "**Functionaliteit Klaar**" definiëren, nemen we alle testen voor stories, maar misschien ook testen zoals gebruikersacceptatietesten (UAT) en exploratory testen van de functionaliteit. We

bevelen aan om alle kwaliteitskenmerken die niet konden getest worden op story niveau uit te voeren op het niveau van de functionaliteit.

Je kan zelfs "**Release Klaar**" definiëren. Dat zou alle testen omvatten die van toepassing is op jouw context, vanuit elk kwadrant.

> ***Hint**: Onthoud, wanneer we spreken van "klaar-heid" van een story, een functionaliteit, of een release, leggen we niet op dat deze testen in een vaste volgorde moeten uitgevoerd worden. Het verdient eerder aanbeveling te overwegen welke testen de ontwikkeling in goede banen leiden (fouten vermijden) en dewelke het product beoordelen (tekorten in de code ontdekken).*



Figuur 9.3: Zoeken en "oplossen" van fouten

De modellen vinden die passen in jouw context

Veel teams vonden de agile testkwadranten nuttig bij het plannen van hun testactiviteiten. Andere hebben het kwadrantenmodel aangepast om beter te voldoen aan hun noden. Er zijn veel nuttige variaties op de kwadranten. Je kan Hoofdstuk 8 van *More Agile Testing* downloaden (in het Engels) van agiletester.ca[38], om verschillende variaties op de kwadranten te vinden. Bekijk onze bronnenlijst voor meer informatie.

Welk model ook het beste werkt voor jullie, zorg ervoor dat je het zichtbaar houdt en gebruik het om conversaties te stimuleren over hoe jullie continu jullie testen kunnen verbeteren.

[38] https://agiletester.ca

Chapter 9: The Agile Testing Quadrants

Brian Marick first wrote about the agile testing quadrants (Figure 9.1) in 2003, when Lisa and Janet were trying to figure out how to talk about testing to the agile community, especially the testers. At that time, most of the agile folks were discussing only "customer tests" and "programmer tests." A lot of teams were focused only on functional acceptance tests – how the features should behave. Today, this is still a potential pitfall for teams transitioning to agile. The quadrants gave us a way to discuss all the different types of testing that a team might need to consider. They help us see the big picture.

The agile testing quadrants are a taxonomy of different types of testing. We use it as a thinking tool to help teams discuss what testing activities they might need and make sure they have the right people, resources, and environments to perform them.

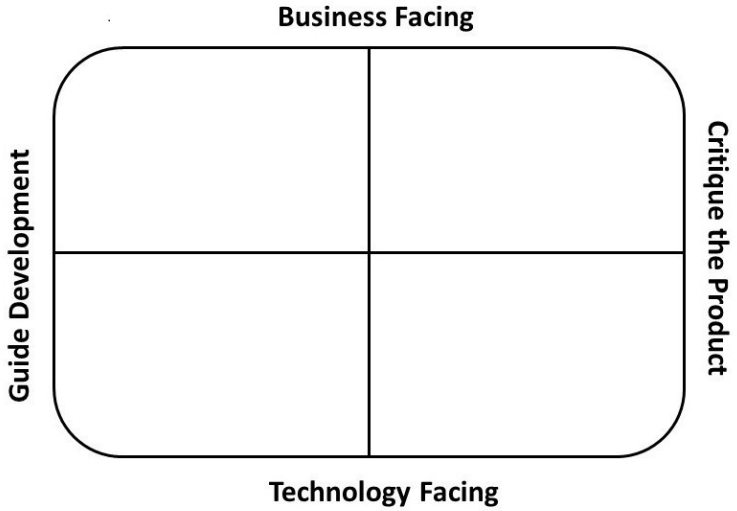


Figure 9.1: Agile Testing Quadrants

Tests on the left-hand side are those that guide development, the ones that are written before coding happens or concurrently as coding proceeds. The tests on the right-hand side are those that critique (evaluate) the product after coding is complete. Tests on the left help prevent defects. Tests on the right find defects in the code or perhaps identify missing features.

The top half of the quadrants focuses on tests that are readable by business stakeholders. These tests answer the question, "Are we building the right thing?" The bottom half includes tests that are written by and for technical team members. The business stakeholders probably care about the end results, but they would not try to read the tests. The top half is about external quality as defined by the business. The bottom half is about internal code or infrastructure correctness.

The quadrants are numbered for ease of reference. The four quadrants are labeled as:

- **Quadrant 1** (Q1): Technology-facing tests that guide development

- **Quadrant 2** (Q2): Business-facing tests that guide development
- **Quadrant 3** (Q3): Business-facing tests that critique the product
- **Quadrant 4** (Q4): Technology-facing tests that critique the product

The agile testing quadrants model helps teams think through testing activities that are needed to give confidence to the product they are building. It also helps to build a common testing language with the team and with the organization if used to help communicate across teams. Janet's favorite thing about this model is that it not only represents a holistic view into testing but also makes the whole team's responsibility for testing activities visible.

Each team has its own distinct combination of business domain, product, skills, product maturity, technical stack, regulatory oversight, and more. The model can be applied to represent the testing required in each context. In Figure 9.2, we share some typical types of tests that might be found in each quadrant.

> ***Hint***: *Remember: Use these examples as a guideline. It's a tool, not a rule, and there are lots of gray areas. Your team's context is unique, and your quadrants should be too. Apply the model to the testing you need to do.*

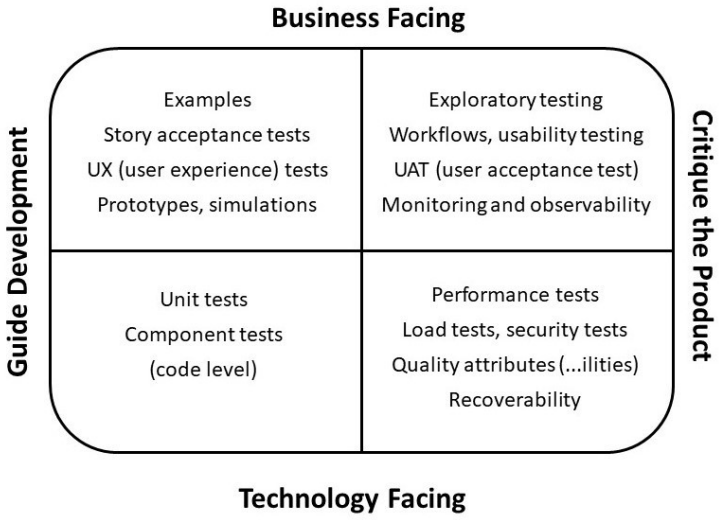


Figure 9.2: Examples of testing types for the quadrants

What tests in what order?

We have numbered the quadrants 1, 2, 3, 4 simply for ease of reference. It's a mouthful to say "Business-facing tests that guide development," so we say, "Quadrant 2" or "Q2." The numbers are not intended to represent that the types of testing activities should be done in that order. As the team plans their testing, they will think about the appropriate time to do each testing activity.

Here's an example. A team decides to start using a new architecture for the new features going forward. They need to be sure that the architecture will scale appropriately to accommodate a certain number of users and a potential load on the system. The team does a "spike," which means they write some throw-away code solely for the purpose of testing out the architecture. They do load and performance testing on the "spike" to see if it meets response time requirements and remains stable. If they're satisfied, they delete the spike and start working on stories for a new feature, this time following their usual development practices. If they're not satisfied, they may re-think the architecture and repeat the process.

For feature development, most teams probably start in Q2. They might draw prototypes on paper and show them to potential customers. The delivery and business team might have a story-mapping or specification workshop to discuss features and slice them into stories. During backlog refinement or story-readiness workshops, the team can use activities such as example mapping to extract more details.

Once the team starts developing a story, they work on tests from Q1, writing unit tests as part of test-driven development. They may simultaneously be working on story tests in Q2. If enough of a feature is delivered to explore, they may move into Q3 testing. However, if security is the number one concern for the business and customers, security testing in Q4 may be prioritized over functional story testing from Q2. Each team finds their own way of working, and it may change for different types of features or projects.

Using the quadrants

Janet and Lisa have seen teams use the quadrants in many ways. Some teams put a poster on the wall with the blank quadrants. As they plan a new feature or release, they talk about all the testing they'll need and write each type in the appropriate quadrant. They use it as a reminder about what they need to do or what they may have forgotten.

Quadrant 1

Teams can also use the quadrants to talk about what tests should be automated. Q1 tests are typically automated by the developers writing the production code. Many teams practice test-driven development (TDD), writing a small unit test for some small piece of functionality, then the code to make that test pass. Q1 tests are designed to run quickly since they test such a small area of code and generally don't include interaction with other layers of the application or databases. They give teams the fast feedback needed to make changes to the code quickly and fearlessly.

Quadrant 2

The business-facing tests that guide development in Q2 are an important foundation for most teams. Product owners, developers, testers, and others meet frequently to plan features and stories. They may use techniques such as example mapping to elicit business rules for each story along with the examples that illustrate them. Teams practicing behavior-driven development (BDD), acceptance test-driven development (ATDD), or specification by example (SBE) turn these into scenarios that specify behavior as executable tests. These scenarios can be automated as the code is written.

Quadrant 3

Tests in Q3 tend to be human-centric, learning whether the delivered stories and features provide the intended value to customers. Automation may be used to facilitate this testing via data or state setup. It's becoming more common for teams to do exploratory testing in production, using release feature toggles to "hide" new features from customers until testing is complete. Q3 tests include forms of testing in production such as monitoring analytics to learn what really happens and how customers use the features. Information learned from Q3 testing feeds back into Q2, often resulting in creating new stories or features.

Quadrant 4

Many Q4 tests depend on automation and tools, but some may require additional testing. For example, the automated tools for accessibility (often shortened to "a11y," representing the starting and ending letters and the number of letters in between) testing are still not as effective as manual exploratory testing for those capabilities. Results of these tests often feed back into Q1 activities as the team changes the code design to improve various quality attributes. Monitoring performance and errors in production, or testing for recoverability, can also be considered a type of testing that falls into Q4. Today's technology has made it feasible and safe to test in production. This doesn't mean that we let customers find bugs for us but that we learn for sure how the code behaves in a true production environment. (Details on testing in production can be found in Chapter 8.)

Defining "Done"

Use the quadrants to define what "Done" means to your team. Many teams struggle when trying to decide what testing should be included in that definition. In Chapter 3, we talked about levels of detail, and those levels apply here. Instead of defining "Done," we encourage teams to be more specific and call it "**Story Done**." Testing that can be included in "Story Done" would likely be all of Q1, all of Q2, and perhaps some of Q3 like exploratory testing.

Go one step further and define "**Feature Done**," which would include all testing for stories, but perhaps also includes tests like user acceptance testing (UAT) and exploratory testing at the feature level. We recommend that all quality attributes that could not be tested at the story level should be performed at the feature level.

You may even choose to define "**Release Done**." That would include every test that applies in your context, from every quadrant.

> ***Hint**: Remember, when we talk about "done-ness" at story, feature, and release levels, we are not mandating that tests be done in a certain order. Rather, consider which tests guide development (prevent defects) and which ones are critiquing the product (finding defects in the code).*

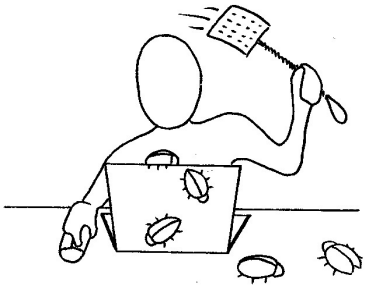


Figure 9.3: Finding and "fixing" bugs

Find the models that fit your context

Many teams have found the agile testing quadrants useful in planning their testing activities. Others have adapted the quadrants model to better suit their needs, and there are many useful variations on the quadrants. You can find Chapter 8 from *More Agile Testing* available for download on agiletester.ca[39], and it includes several adaptations of the quadrants. Check our resources list for more.

Whatever model works best for you, be sure to keep it visible and use it to stimulate conversations about how to continually improve your testing.

[39] https://agiletester.ca

Hoofdstuk 10: Visualiseren van een testautomatiseringsstrategie

Veel teams hebben nog steeds geen geautomatiseerde regressietesten. Sommige teams voelen dat ze het proces onder de knie hebben, maar als ze nieuwe technologie in hun product gebruiken, merken ze dat hun bestaande testautomatisering gereedschappen hier niet mee kunnen omgaan. Ze worstelen constant met de balans tussen waarde en onderhoudskosten. Waar een individueel team ook op zijn testautomatisering reis is, is het nuttig om een stap terug te zetten en te denken op prioriteiten en verbeteringen, waarop vervolgens kan worden gefocust. Dit hoofdstuk is niet bedoeld om een uitgebreide introductie te geven over testautomatisering, maar om teams te laten zien hoe visuele modellen kunnen helpen om hun testautomatiseringsstrategieën te ontwerpen.

Gebruik maken van visuele modellen

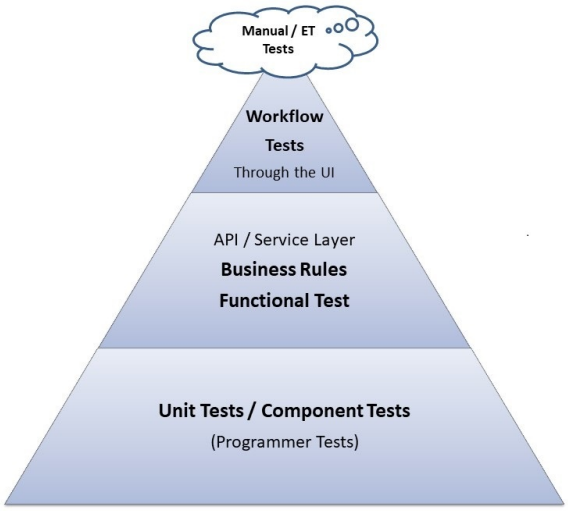
Het betrekken van het gehele team in het formuleren van een strategie om de automatiseringsbehoeften in kaart te brengen en het uitvoeren van die strategie is de sleutel om te slagen met automatisering. Visuele modellen leiden deze gesprekken.

De testkwadranten, besproken in hoofdstuk 9, kunnen deze teams helpen om hun teststrategieën te plannen als ze zowel deze testtypes die nodig zijn als de vaardigheden, gereedschappen en infrastructuur, die ze nodig zullen hebben om dit voor elkaar te krijgen. In dit hoofdstuk kijken we naar aanvullende modellen, die teams kunnen helpen om een succesvolle automatiseringsstrategie te vinden.

> ***Hint***: *Onthoud, dat dit denk gereedschappen zijn om te worden gebruikt voor het starten van gesprekken over hoe jouw team testen wilt automatiseren..*

De klassieke testautomatiseringspiramide

Mike Cohns testautomatiseringspiramide heeft veel teams geholpen sinds de eerste jaren na 2000. We hebben het wat aangepast vanaf toen (Figuur 10.1) om onze intentie duidelijk te maken, door een wolk in te voegen aan de top om aan te geven dat niet alle regressietesten kunnen worden geautomatiseerd. Vaak hebben we een mens centrale testen nodig, die onderzoekende testen (ET) bevatten.

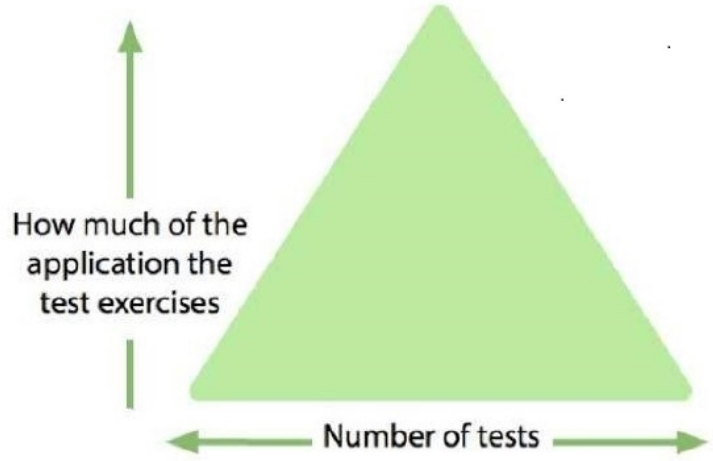


Figuur 10.1: Klassieke Testautomatiseringspiramide

Sommige regressies vinden alleen plaats wanneer twee of meer lagen van de applicatie zijn betrokken. Dit zou workflow testen door de UI vereisen die de server, database, en/of extern systeem erbij betrekken. In de meeste contexten is het beste om het aantal end-to-end workflow testen te minimaliseren. Deze testen zijn langzaam, zijn vaak het meest breekbaar en vereisen meestal het meeste onderhoud. De wolk op de top van de piramide bevat mens centrale activisten zoals exploratory testen en andere taken, die niet kunnen worden geautomatiseerd.

De testautomatiseringspiramide helpt ons te bedenken om “testen lager te duwen”, de regressietesten tot één deel van de applicatie te maximaliseren en degenen die meerdere lagen van het systeem erbij betrekken, te minimaliseren. Wanneer teams testen plannen, kunnen ze naar de piramide kijken om te zien welke testen het meest geschikt kunnen worden geautomatiseerd.

De klassieke piramide is niet bedoeld dat er een bepaald nummer of percentage van geautomatiseerde testen op elk niveau is. Seb Rose vormde het model een beetje anders zoals getoond in Figuur 10.2.



Figuur 10.2: Seb Roses versie van de piramide, aantal testen ten opzichte van testdekking.

Sebs model verduidelijkt dat wat een test duurder maakt is het aantal lagen van de applicatie, die nodig is om het uit te voeren. Het is bijvoorbeeld mogelijk om een unit niveau test te hebben van de UI die niet andere lagen van de applicatie erbij te betrekken. Het is mogelijk om TDD te gebruiken voor elke geïsoleerde laag van de applicatie ongeacht of het de server, de API, de UI of de microservice is.

Er zijn veel aanpassingen van het piramidemodel (en ja, de klassieke is echt een driehoek) over de jaren. Hoofdstuk 15 van "More Agile Testing" bevat verscheidene aanpassingen inclusief die van Alister Scott en Sharon Robson.

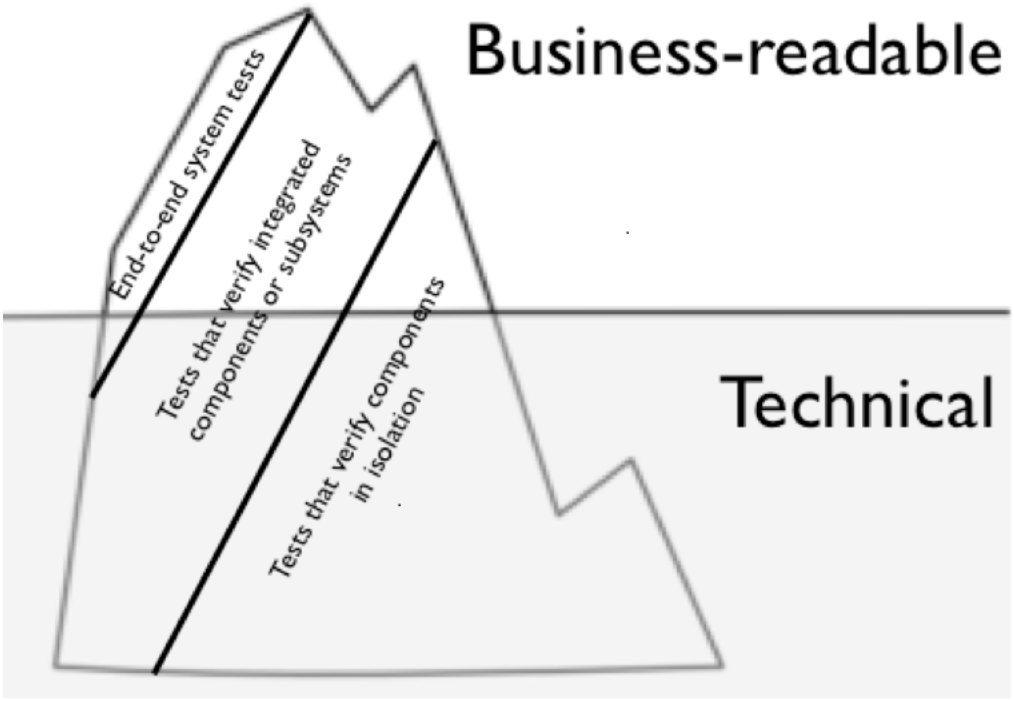
Teams die geautomatiseerde testen hebben, kunnen de "vorm" van hun piramide tekenen om te visualiseren waar hun huidige testen passen. Veel teams starten met meestal UI testen en een "ondersteboven" piramide of "ijshoorn[40]." Anderen hebben wellicht een zandloper. Geen van deze vormen zijn strikt verkeerd, maar als de huidige automatisering niet aan de behoeften van het team voldoet, kunnen plaatjes helpen om te zien welke veranderingen nodig zijn.

Gesprekken over een visueel model helpen teams die net aan het starten zijn met automatisering inspanningen, om te beslissen over hun eventuele doel en hun eerste prioriteiten.

Geautomatiseerde testen als levende documentatie

De tijd, kosten en moeite, die in het automatiseren van verschillende soorten van testen op verschillende lagen van de applicatie, zijn niet enige overwegingen bij het samenstellen van de automatiseringsstrategie. Een andere overweging is om rekening te houden met degene, die in staat zijn om de tests te lezen en begrijpen. Seb Roses testautomatisering ijsberg (Figuur 10.3) herinnert ons aan één van de meest waardevolle eigenschappen van geautomatiseerde testen is de levende documentatie die het ons geeft. Ze zijn altijd bijgewerkt, omdat het team hen de gehele tijd succesvol doet slagen, zodat je op elk gegeven moment kan vertellen, wat jouw systeem doet.

[40] https://watirmelon.blog/testing-pyramids/



Die delen van de ijsberg boven de waterlijn zijn testen, die leesbaar zijn voor de business, terwijl degene eronder dat niet zijn (zijn geschreven in een technische taal) De hoeveelheid testen zal variëren per team; bijvoorbeeld, Lisa werkte in een team, wiens product was bedoeld voor andere oplever teams. Iedereen in het team inclusief de product owner kon de geautomatiseerde regressietesten geschreven in low-level code begrijpen. Ze hadden geen "bedrijf leesbare" testen. In andere gebieden is het kritisch dat bedrijf stakeholders de acceptatietesten kunnen begrijpen. Dit model helpt ons te herinneren om te denken over wat er nodig is in de context van het team.

Het model uitbreiden

Zelfs de meest ijverige geautomatiseerde regressietesten dekking zou falen om de regressiefouten te identificeren. Geen enkele testomgeving is precies hetzelfde als productie. Sommige bugs worden gevonden in productie zoals besproken in hoofdstuk 8. In haar boek "A Practical Guide In DevOps" presenteert Katrina Clokie haar DevOps filter. Het is een handig plaatje, dat unit testen alleen kleine fouten eruit filteren, terwijl verschillende lagen van integratietesten en end-to-end testen voortschrijdend grotere detecteren. Om de volledig gevormde bugs te vinden hebben teams logs, alarmeringen en monitoring nodig voor hun productiesysteem.

Gedeelde verantwoordelijkheid

We moedigen teams aan om de verantwoordelijkheid van het testen van de bedrijfsregels en hogere niveaus van integratie op API niveau te delen. Testers zouden ook inzicht moet hebben in de unit – en

componenttesten geschreven door de ontwikkelaars. Als teamlid is het belangrijk om de applicatie van jouw team en de interne werking te begrijpen bij het benaderen van jouw automatiseringsstrategie.

Omdat het automatiseren van testen door de UI de neiging heeft om meer tijd te consumeren, is er de verleiding om dit over te dragen aan een apart automatiseringsteam of de testers in het team volledige verantwoordelijkheid hiervoor te geven. Wij raden aan, dat de ontwikkelaars die efficiënte onderhoudbare code schrijven, samen te werken met de testers, die goed zijn in het specificeren van testgevallen door zowel de UI als de andere lagen boven de basislaag van de piramide. Onthoud, dat net als de andere modellen is de testautomatiseringspiramide een gids. Teams die alle leden in alle rollen samenbrengen om vragen te stellen, praten over de antwoorden, op het whiteboard tekenen en experimenten ontwerpen, hebben het beste succes met automatisering. Visuele modellen zoals de voorbeelden in dit hoofdstuk helpen teams te praten waarom ze testen automatiseren, wat de grootste automatisering pijnpunten zijn, hoe mensen met verschillende vaardigheden kunnen helpen en wat hun volgende experimenten zouden zijn. Volgens onze ervaring werkt een stap bij stap benadering het beste.

Chapter 10: Visualizing a Test Automation Strategy

Test automation is a constant challenge for software teams everywhere. Many teams still have no automated regression tests. Some teams feel they've mastered the process, but then they update their product to incorporate new technology that their existing automation tools can't handle. They often struggle to maintain a balance between value and maintenance cost.

Wherever an individual team is on its automation journey, it's helpful to take a step back and think about priorities and what improvements to focus on next. This chapter is not meant to give you an extensive introduction into automation but to show how using visual models can help teams design their automation strategy.

Using visual models

Getting the whole team involved in formulating a strategy for addressing different automation needs and executing that strategy is key to succeeding with automation. Visual models help guide these conversations.

The agile testing quadrants model covered in Chapter 9 can help teams plan their automation strategy as they discuss the different types of testing that are needed, as well as what skills, tools, and infrastructure they will need to complete it. In this chapter, we look at some additional models that can help teams find a successful automation strategy.

> ***Hint***: *Remember, these are thinking tools, to be used to start conversations about how your team wants to automate tests.*

The classic test automation pyramid

Mike Cohn's test automation pyramid has helped many teams since the early 2000s. We've adjusted it slightly since then (Figure 10.1) to make our intent clear, including the cloud bubble on top to represent that not all regression tests can be automated. Sometimes we need human-centric tests, which include exploratory tests (ET).

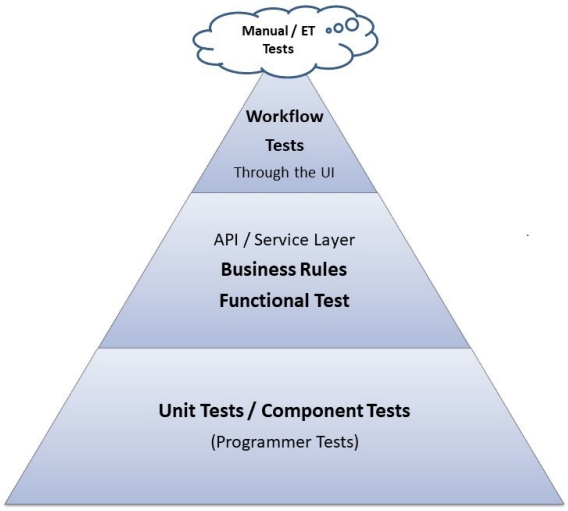


Figure 10.1: Classic Test Automation Pyramid

This model helps teams understand that in most contexts, it pays to automate tests at the most granular level of the application as possible, to provide adequate protection against regression failures. Teams that practice test-driven development (TDD) build up a solid base of unit- and component-level tests that help guide code design. These tests run very fast, so they give the team quick feedback.

With most applications, testing interactions between different layers of the architecture is required. For example, business logic usually requires interaction with the database. Doing as much automation of this type at the service or API level without going through a user interface (UI) is generally the most efficient way.

Some regressions only occur when two or more layers of the application are involved. That may require workflow tests through the UI that involve the server, database, and/or an external system. In most contexts, it's best to minimize the number of end-to-end workflow tests. These tests run slowly, are often the most brittle, and usually require the most maintenance. The cloud at the top of the pyramid includes human-centric activities such as exploratory testing and other tasks that can't be automated.

The test automation pyramid helps us think of ways to "push tests lower," maximize the regression tests that are isolated to one part of an application, and minimize those that involve multiple parts of the system. When teams plan testing for a feature, they can look at the pyramid to see where each test can most appropriately be automated.

The classic pyramid is not meant to imply that there is a certain number or percentage of automated tests at each level. Seb Rose envisions the model a bit differently, as shown in Figure 10.2.

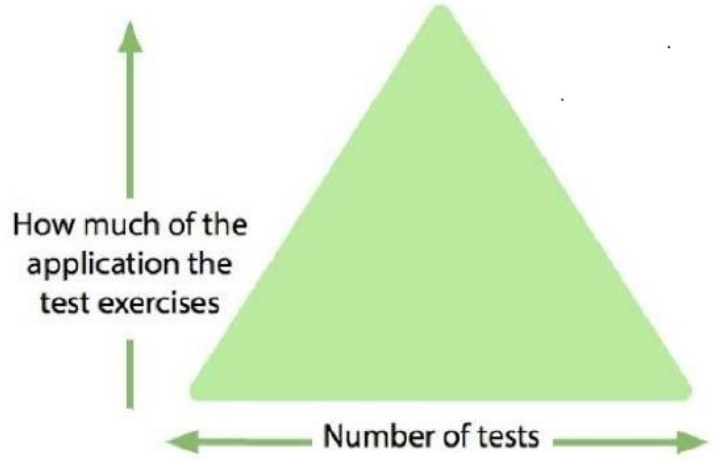


Figure 10.2: Seb Rose's version of the pyramid, number of tests vs. test coverage

Seb's model clarifies that what makes a test more expensive is the number of layers of the application it requires to execute. For example, it's possible to have a unit-level test of the UI that does not involve any other layers of the application. It's possible to use TDD for each isolated layer of the application, whether it's the server, the API, the UI, or a microservice.

There have been many adaptations of the pyramid model (and yes, the classic one is really a triangle) over the years. Chapter 15 of *More Agile Testing* includes several adaptations, including those from Alister Scott and Sharon Robson.

Teams that have automated tests can draw the "shape" of their pyramid to visualize where their current tests fit. Many teams start out with mostly UI tests and an "upside-down" pyramid or "ice cream cone[41]." Others may have an hourglass. None of these shapes are necessarily wrong, but if the current automation doesn't meet the team's needs, the visuals can help picture what changes are needed.

Conversations around a visual model help teams that are just starting their automation efforts decide their eventual goal and their first priorities.

Automated tests as living documentation

The time, cost, and effort that goes into automating different types of tests at different levels of the application are not the only considerations when putting together an automation strategy. Another consideration is remembering who needs to be able to read and understand the tests. Seb Rose's Test Automation Iceberg (Figure 10.3) reminds us that one of the most valuable attributes of automated tests is the living documentation that they provide. They're always up to date because the team keeps them passing all the time, so you can tell at any given time exactly what your system does.

[41] https://watirmelon.blog/testing-pyramids/

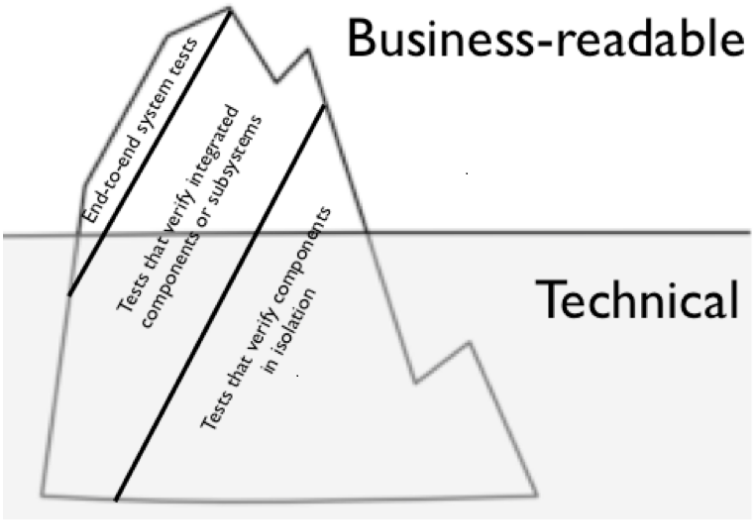


Figure 10.3: Seb Rose's test automation iceberg

Those portions of the iceberg above the waterline are tests that are business readable, while those below are not (are written in a technical language). The amount of tests will vary by team; for example, Lisa worked on a team whose product was intended for other delivery teams. Everyone on the team including the product owner could understand the automated regression tests written in low-level code. They didn't need "business-readable" tests. In other domains, it's critical that business stakeholders can understand the acceptance tests. This model helps remind us to think about what's needed in the team's context.

Extending the model

Even the most diligent automated regression test coverage may fail to identify some regression failures. No test environment is exactly like production. Some bugs may be found via "testing in production," as discussed in Chapter 8.

In her book *A Practical Guide to Testing in DevOps*, Katrina Clokie presents her DevOps bug filter. It's a helpful visual showing that unit tests can only filter out the small bugs, while different levels of integration and end-to-end tests detect progressively bigger ones. To find the fully formed bugs, teams need logging, alerting, and monitoring for their production system.

Shared responsibility

We encourage teams to share the responsibility of testing the business rules and higher levels of integration at the API level. Testers should also have visibility into the unit and component tests written by developers. As a team member, it's important to understand your team's applications and the inner workings when approaching your automation strategy.

Because automating tests through the UI tends to be more time-consuming, there's a temptation to hand that off to a separate automation team or have the testers on the team take full responsibility for it. We recommend

that the developers, who are good at writing efficient, maintainable code, work together with the testers, who are good at specifying test cases, to automate tests through the UI as well as all other layers above the base level of the pyramid.

Remember, like all the other models, the test automation pyramid is a guide. Teams that get members in all roles together to ask questions, chat about the answers, draw on the whiteboard, and design experiments have the best success with automation. Visual models like the examples in this chapter help teams talk about why they're automating tests, what their biggest automation pain points are, how people with different skills could help, and what their next experiments should be. In our experience, a step-by-step approach works best.

DEEL 4: Agile testing vandaag

De grondbeginselen van agile testen - zoals het gebruik van de teambrede aanpak, het begeleiden van de ontwikkeling met voorbeelden en het samenwerken tussen verschillende rollen om kwaliteit in te bouwen - zijn vandaag net zo effectief als 20 jaar geleden. Is er iets dat we moeten veranderen om de uitdagingen van nu aan te gaan? In deze sectie delen we wat enkele toonaangevende agile testers zien veranderen aan de testing rol. We sluiten af met een aantal ingrediënten om teams te helpen slagen met hun agile testen.

SECTION 4: Agile Testing Today

The basics of agile testing – such as using the whole-team approach, guiding development with examples, and collaborating across roles to build quality in – are as effective today as they were 20 years ago. Is there anything we need to change to meet today's challenges? In this section, we share what some leading agile testing practitioners see changing for the role of testers. We'll wrap up with a bunch of ingredients to help teams succeed with agile testing.

Hoofdstuk 11: De nieuwe rol van de tester

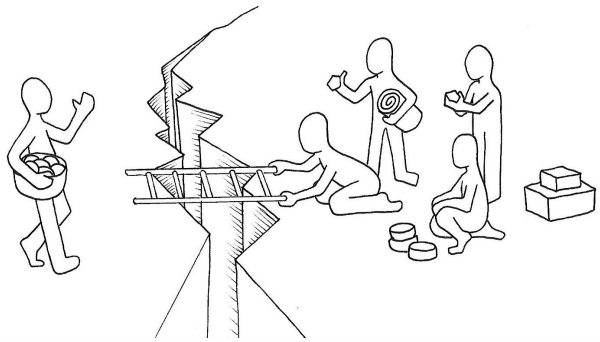
Veel teams worstelen met slechts één tester als onderdeel van het team – of zelfs erger nog, een tester die meer dan één opleverteam ondersteund. Als de tester de enige is die de testactiviteiten uitvoert, creëert het over het algemeen een knelpunt. Agile teams die elke iteratie wijzigingen naar productie doorvoeren (of zelfs vaker als ze aan continuous delivery doen) kunnen het zich niet veroorloven om slechts één tester te hebben die alle testen doet.

We kunnen de toekomst niet voorspellen, maar het is informatief om rond te kijken hoe de rol van een tester aan het veranderen is. We vroegen andere ervaren agile test coaches en trainers om erachter te komen wat hun ideeën zijn over de veranderende rol van een tester. Deze verschillende perspectieven kunnen je helpen begrijpen hoe testers en teams zich kunnen aanpassen en kunnen helpen bouwen aan een kwaliteitscultuur.

Testers zijn kwaliteitslijm voor een team

Alex Schladebeck – Duitsland

Toen ik begon, was de rol van tester in een team (als de tester al een deel van het team was) vaak alleen verantwoordelijk voor UI-automatisering en handmatig testen. Over de laatste 12 jaar heb ik een enorme diversificatie van de rol gezien, en altijd in een contextafhankelijke manier op basis van hoe het team opereerde. Ik zie testers werken aan meer automatiseringstaken, zelfs pairen op eenheidsniveau met ontwikkelaars. Ik zie dat ze bij alle procespunten betrokken zijn. Ik zie ze organiseren van mob-testsessies met het team om verkennende testen uit te voeren. Ik zie ze campagne voeren voor betere feedback loops. Ik heb zelfs testers zien beginnen met het repareren van bugs of het implementeren van functionaliteiten.



Maak verbindingen

Voor mij is deze diversificatie en rolvervaging een enorme vrijheid en een grote verantwoordelijkheid. Het betekent dat we ons moeten afvragen: “Hoe kan ik, mijn vaardigheden en mijn leerpotentieel het best passen

in de context van dit team?" We worden heel erg 'kwaliteits- en communicatielijm', identificeren en opvullen van de gaten gaten in elk team.

Ik ben de term "embedded kwaliteits ingenieur" of "embedded kwaliteitsadviseur" gaan gebruiken voor deze rol. Het probleem met de titel "tester" is dat het de naam bevat van een van de vele activiteiten die we doen, dus jij hoort vragen als: "Als iedereen betrokken is bij testen, waarom hebben we dan een tester nodig?" of uitspraken als: "De ontwikkelaars automatiseren tests, dus we hebben geen testersrol nodig." Testen is slechts één van de vele dingen dat een tester doet. Naar mijn mening moeten we tegen het idee vechten dat een agile team moet bestaan uit "chimeras": een mix van verschillende rollen, of een Swiss-Army-knife-teamlid dat alles kan: vereisten, UX, testen, beveiliging, front- en backend. Agile teams moeten divers en multifunctioneel zijn, en dat betekent dat we mensen nodig hebben met verschillende achtergronden, interesses en specialisaties, zonder dat iemand een silo of bottleneck is. Ik denk dat dit een haalbare balans is.

Ik zie de rol van een tester als bestaande uit meerdere activiteiten. Er zijn dingen die altijd al deel uitmaakten van de rol, zoals werken met belanghebbenden, expertise inbrengen om activiteiten te testen, koppelen met developers en andere testers, ondersteunen van de product owner, organiseren en het aanpassen van de algehele kwaliteitsstrategie, het verkrijgen van goede testgegevens, en risico's identificeren. Ik denk dat er in de toekomst nog meer taken overgenomen of ondersteund zullen worden door testers. Sommige hiervan kunnen zijn: samenwerken met het team om te zorgen voor testbaarheid en waarneembaarheid voor testen en monitoren in productie, vragen stellen over het productie systeem om te onderzoeken hoe het wordt gebruikt, onze prestaties aan te scherpen en het aanleren van vaardigheden voor verkennende testen, waardoor het team zich kan concentreren op waarde (en soms op minimalisme, d.w.z. de waarde van iets niet te doen). Ik zie ook dat testers 'teamgezondheid' aan hun kwaliteitsattributen beginnen toe te voegen, lettend op de communicatie en stressniveaus van de ploeg als geheel. Dit zijn immers zaken die de kwaliteit enorm kunnen beïnvloeden. Kortom, ik denk dat de rol van tester belangrijk blijft. Zij zijn de persoon in het team wiens belangrijkste prioriteit kwaliteit is. Ze zijn ook de persoon met de passie voor kwaliteit en testen, en zij pleiten en supporteren voor hen.

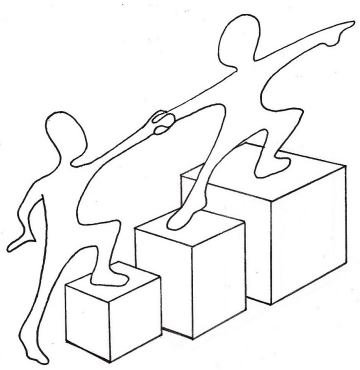
De professionele reis van een agile tester

Paul Carvalho – Canada

Als ik agile teams coach, help ik het hele team om te leren samenwerken en voortbouwen op elkaars sterke punten. Een product owner brengt kennis van het bedrijfsleven en de industrie, een programmeur brengt sterke codeer- en ontwikkelingsvaardigheden, een ontwerper brengt inzichten in het perspectief en de ervaring van de gebruiker, en een tester brengt ook iets van waarde op tafel.

De professionele reis van een tester begint met afstand te nemen van toevalligheden en willekeurig testen, tot een doordacht ontwerp van testen door middel van modellen, technieken en andere gespecialiseerde vaardigheden en kennis. Agile zet een sterke focus op de nauwe samenwerking met anderen, dus testers moeten uit hun hoofd komen wanneer ze aan testen denken en hun stem vinden om te helpen andere teamleden te begrijpen de vele manieren om kwaliteit te genereren met informatie over de systemen in ontwikkeling. Een geweldige agile tester brengt meer tijd door met een whiteboard-marker in de hand en koppelt met anderen teamleden dan iets anders. Het gaat erom de rest van het team te helpen om meer over het systeem te zien en te begrijpen, voordat het systeem wordt gebouwd. "Bouw kwaliteit in" is meer dan

een slogan; het is een realiteit die geweldige testers kunnen helpen bij het inschakelen van goed presterende samenwerkende teams.



Mentor

De professionele reis van een agile tester verloopt van willekeurig, lukraak testen, tot het begrijpen van de technieken en het ontwerp van goed testen, tot een stem vinden en deze ideeën uiten, zodat ze kunnen helpen de prestaties te verhogen van de rest van uw team.

Probeer als tester ieders begrip van de systemen te vergroten en te laten groeien door middel van doordachte verkenning. Als je het idee loslaat dat jij degene moet zijn die de testgevallen of testautomatisatie moet schrijven, bedenk waar uw unieke vaardigheden en inzichten kunnen helpen om het team te dragen.

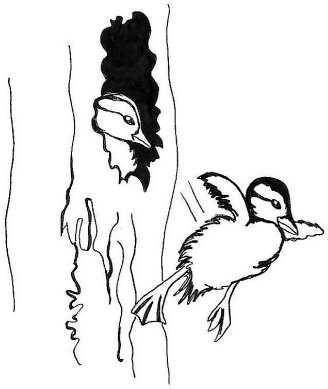
Het fascinerende pad om te evolueren als tester

Claudia Badell – Uruguay

Als testers kunnen wij bijdragen en waarde toevoegen vanuit verschillende perspectieven: als facilitators en evangelisten naar testen en kwaliteit in een productteam, als coaches, als testadviseurs, als experts in bepaalde soorten testen (bruikbaarheidstesten, toegankelijkheid onder andere testen, beveiligingstesten, prestatietesten) en meer. Testers worden niet meer gezien als poortwachters voor kwaliteit, dus wij mogen gezien en gewaardeerd worden als pleitbezorgers van kwaliteit.

Tegenwoordig is het steeds vaker dat testers deel uitmaken van het ontwikkelingsteam en bijdragen leveren vanaf het begin van het ontwikkelingsproces. Mijn ervaring is de rol van een tester in deze context evolueert. Naast het uitvoeren van testwerkzaamheden ter ondersteuning van handmatige testen en geautomatiseerde controles, werken testers samen om bruggen te bouwen binnen het team om tot een gemeenschappelijk begrip te komen en betrokkenheid bij de testen. Ook het definiëren, opvolgen en bijsturen van teststrategieën die door het hele team moeten worden toegepast. Bovendien delen en evangeliseren zij hun kennis van testen binnen het team.

Naarmate technologie, methodologieën en processen evolueren en teams en gemeenschappen volwassen worden, denk ik dat het belangrijk voor ons is om een proactieve houding aan te nemen om zich aan dergelijke veranderingen aan te passen. De toekomst brengt nieuwe uitdagingen en kansen. Afhankelijk van de context, kunnen verschillende vaardigheden en testactiviteiten nodig zijn, maar in mijn ervaring zijn er kernvaardigheden die nodig zijn om gelijke tred te houden met software ontwikkeling.



Experimenteer en ontdek nieuwe kansen

Deze kernvaardigheden zijn:

- wees leergierig en probeer nieuwe experimenten uit om de teststrategieën in het team te verbeteren.
- wees een uitstekende vragensteller gedurende de hele levenscyclus van het product. Afhankelijk van het type informatie dat we verzamelen, kunnen vragen op verschillende manieren geformuleerd worden. Ze kunnen mondeling of schriftelijk gemaakt worden, dus duidelijke en goed gestructureerde communicatieve vaardigheden zijn belangrijk. Ze kunnen ook programmatisch gemaakt worden; bijvoorbeeld als we wilden bepaalde antwoorden tussen twee services willen controleren, technisch vaardigheden zijn belangrijk.
- modelleringsvaardigheden hebben als een manier om te begrijpen wat te testen en het definiëren van teststrategieën die de verschillende aspecten omvatten die nodig zijn.
- een zekere mate van technische kennis hebben om samen te werken bij het definiëren van de testbaarheidsaspecten van de oplossing terwijl de software in ontwikkeling is – bijvoorbeeld ter ondersteuning van unit testing en geautomatiseerde integratie testen.
- een delende en samenwerkende houding hebben.

We zitten in een spannende tijd waarin we een deel van onze toekomst vorm kunnen geven. Hoe ben jij je erop aan het voorbereiden?

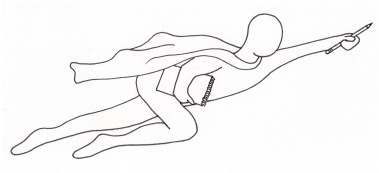
Wees alles wat je kan zijn

Mike Talks – Nieuw-Zeeland

In de afgelopen zes jaar is mijn testteam verdwenen van één enkele groep die aan één watervalproject tegelijk werkt tot individuen die in meerdere teams werken. Er wordt veel gesproken over een teambrede aanpak van kwaliteit en testen, maar testers als specialisten worden gezien als leiders in deze ruimte. Dat betekent het creëren van een first-pass-benadering bij een nieuw verhaal of functionaliteit, maar het is ook belangrijk om een discussie met het grotere team te faciliteren over deze benaderingen om feedback te krijgen en de aanpak te verkennen. Het betekent ook dat een testtaak te zwaar is voor testers om alleen te voltooien, ze kunnen helpen bij het organiseren van een taakverdeling onder de leden van een bereidwillig team.

Ik vind een veel voorkomende kandidaat hiervoor het testen van verschillende browsers/apparaten. Wij testen vaak functionaliteiten tijdens de iteratie op onze kernapparaten, maar bezoeken af en toe ons product op een veel breder scala van apparaten. Dit is waar het team en hun frisse ogen kunnen helpen, en een klein beetje organisatie van de tester kan een enorm verschil maken.

Hoewel de meeste gesprekken over het testen van een product plaatsvinden binnen uw team, is het ook handig om "bij te praten" met anderen in dezelfde disciplines om ideeën te delen en te zien wat er in andere teams werkt. Dit kan ook evolueren in het begeleiden van individuen te helpen omgaan met specifieke problemen alsook de moed om meer nieuwe ideeën uit te proberen.



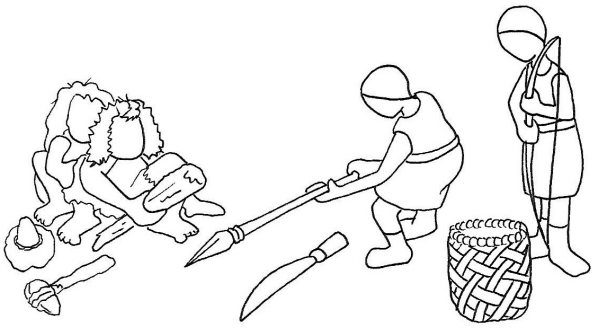
Wees alles wat je kan zijn

Begin met een gesprek

Kathleen Naughton – Verenigde Staten

Het testen van software is tegelijkertijd geëvolueerd en stond het stil. Het is geëvolueerd tot het punt dat sommige organisaties het aantal testers verminderde of zelfs elimineerde in hun teams. Het stond stil omdat diezelfde organisaties moeite hebben gehad om de speciale vaardigheden te waarderen die een tester aan teams toevoegd.

In mijn ervaring, waar testers zijn verminderd of geëlimineerd, hebben de teams dit geprobeerd op te vangen door dichter bij TDD-praktijken te komen, zodat ze een groot aantal geautomatiseerde unit-tests hebben. Ze proberen wat intra-team testen (elkaars code testen) te doen en zijn sterk afhankelijk van hun continue leveringspijplijn om hun geautomatiseerde testen uit te voeren. Wat vaak gemist wordt zijn de integratie- en gebruikerservaringstesten die essentieel zijn voor hoogwaardige producten. Als er testers binnen deze organisaties zijn, ze zijn aanwezig om handmatige tests uit te voeren nadat de code reeds klaar is. Alle inzichten of suggesties van deze testers hebben de neiging om gedeprioriteerd te worden in de product backlog in uitstel voor functionaliteitenontwikkeling.



Wees relevant en beïnvloed de mensen om je heen

Ik geloof dat een essentiële vaardigheid die testers nodig hebben om relevant te zijn, is code kunnen lezen en begrijpen. Hierdoor kunnen ze begrijpen wat unit testen of andere geautomatiseerde tests controleren en maken de identificatie van testlacunes mogelijk die de tester kan invullen. Het laat ook vermindering toe voor van testoverlapping. Als er al een eenheid of integratie test aanwezig is, kan de testaanpak meer gericht zijn op de eindgebruiker activiteiten die niet gedekt zijn. Een andere essentiële vaardigheid die nodig is, kan volgens mij worden beschouwd als een zachte vaardigheid. De vaardigheid van het hebben van gesprekken met programmeurs over hun unit- en integratietesten is krachtig. Deze gesprekken kunnen leiden tot beïnvloeding van ontwerpbeslissingen die op hun beurt opleveringen van hogere kwaliteit mogelijk maken. Kennis brengen over hoe cruciale gesprekken te voeren stelt het hele team in staat betere software produceren.

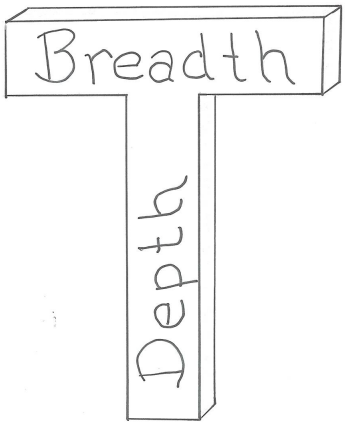
De wereld heeft niet meer controleurs nodig

Aldo Rall – Nieuw-Zeeland

Testen is in de loop der jaren geëvolueerd, en de industrie heeft testtechnische vaardigheden en praktijken ontwikkeld, echter niet zo vlot. Ik denk dat we een tijdperk zijn ingegaan waarin verlichting op de proef wordt gesteld, en er is een grote verschuiving in hoe organisaties en testers nu denken over de testende rol.

Ik zie een toenemende beweging in de richting van het belang van vaardigheden. De meer vaardigheden een individu heeft, hoe waardevoller ze worden voor een team of organisatie. Die personen met vaardigheden die verder gaan dan alleen test technische vaardigheden kunnen meer bijdragen dan iemand die beschikt over een basisset van testontwerp- en uitvoeringsvaardigheden. Dit idee wordt benadrukt door het denken van generaliseren van specialisten als besproken door Scott Ambler[42] of T-vormige vaardigheden zoals besproken door Lisa en Janet in hun boeken. Als jij je carrière toekomstbestendig wilt maken, je testengineeringvaardigheden wilt opbouwen, maar ook vaardigheden buiten de traditionele testwereld. Vergeet de titels maar; over tien jaar zal het voor niemand veel betekenen als jij “testingenieur”, “testspecialist”, “tester”, “test analist” of “verificatie-engineer” genoemd werd. Vaardigheden zijn uiteindelijk meer waardevoller dan verzamelde functietitels; Ik heb dit zeker als waarheid gevonden in mijn eigen carrière.

[42] http://www.agilemodeling.com/essays/generalizingSpecialists.htm



T-vormige vaardigheden

Ik zou kijken naar een holistische benadering die wordt gekenmerkt door inclusiviteit "en" gesprekken. Het beste voorbeeld dat ik kan bedenken is om je verzameling van vaardigheden te combineren op een unieke manier die past bij uw specifieke context, wetende dat zelfs dat zal veranderen. Hoe kun je een set test technische en analytische vaardigheden combineren in een bepaalde situatie? Hoe kun je onderhandelingsvaardigheden combineren met leervaardigheden? Hoe kun je verschillende vaardigheden combineren op het gebied van testtechnieken om een betere testdekking te verkrijgen? Denk "en."

Ik geloof dat één van de belangrijkste vaardigheden die een moderne beoefenaar moet hebben is het vermogen om een context te begrijpen, de veranderende aard ervan te begrijpen, en overeenkomstig aan te passen. De echte meesters van de toekomstige werkplek zullen degenen zijn die een context kunnen observeren, en de combinatie van de gepaste vaardigheden toe te passen, en vervolgens voortdurend de combinatie van vaardigheden bij te stellen naarmate de context evolueert. Die intuïtie heeft tijd nodig om zich te ontwikkelen, en het is een constant veranderend domein. Het leren van nieuwe vaardigheden zal de bekwaamheid en de waarde van zo'n persoon verhogen bij de organisaties en teams.

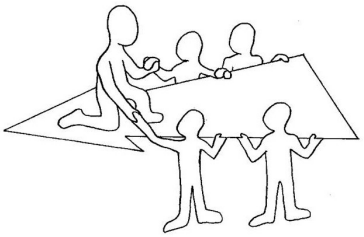
We brengen veel verschillende vaardigheden mee dan alleen test technische vaardigheden. Ik zou willen suggereren dat we overwegen (door onbeschaamd te lenen van anderen) een veelzijdige aanpak. Noem het de "Holistische agile test vaardigheden" of "De tien denkhoeden van agile testen", of wat dan ook logisch is voor jou. Sommige van deze facetten kunnen zijn

- **Consultant**: Ja, soms zullen we in ons team moeten "raadplegen" of met een ander team om problemen op te lossen.
- **Test engineering specialist**: We moeten onze test uien kennen van onze sjalotten. We hebben goede en solide testengineeringvaardigheden nodig.
- **Agile geleerde**: Blijf studeren en leren over agile, breng ideeën naar het team en experimenteer.
- **Coach**: We hebben geweldige kansen om het team te coachen of zelfs collega's (binnen en buiten het team). Dit is een levensvaardigheid, in mijn mening.
- **Mentor**: Soms moeten we iemand begeleiden bij het testen.
- **Facilitator**: Soms moeten we gewoon in de rol stappen van facilitator voor een beslissing, discussie, uitleg, enz.
- **Veranderingsagent**: We kunnen soms verandering en onrust teweegbrengen, pleiten voor een nieuwe praktijk/techniek/methode om mee te experimenteren en van te leren.

- **Leider**: Ja, het kan soms nodig zijn om op te treden en de leiding te nemen in/namens het team.
- **Leraar**: Dat spreekt voor zich, vooral als er een tekort aan testvaardigheden in het team/organisatie is.
- **Bedrijfsdomein geleerde / verdediger van gezond verstand / grote geheel denker**: Soms is het goed om even weg te gaan en het bos los te zien van de bomen.

De gedachten van Lisa en Janet

We hopen dat je het leuk vond om de gedachten van andere mensen te lezen over wat ze beschouwen als de rol van een tester. Nu zullen we de onze delen. We hebben testers aangemoedigd om hun niet-tester-teamgenoten testvaardigheden te helpen leren. Wanneer het hele team de verantwoordelijkheid neemt voor kwaliteit en testen, heeft elk teamlid enige basis nodig in testvaardigheden. Om dit te bereiken, hebben we vaardigheden geïdentificeerd die we aanraden die testspecialisten zouden moeten leren.



Gebruik uw denkvaardigheden

- **samenwerkingsvaardigheden** om actief deel te nemen aan praktijken zoals mindmapping of voorbeeldmapping
- **faciliteringsvaardigheden** om teamleden te helpen beter te communiceren, faciliteren van bijeenkomsten zoals retrospectives, faciliteren van workshops tot het helpvan van niet-testers testvaardigheden aan te leren
- **onderwijsvaardigheden** om hun kennis te delen met andere teamleden
- **coachingvaardigheden** om het team te helpen problemen te identificeren en experimenten te ontwerpen voor verbeteringen
- **communicatieve vaardigheden** om effectief feedback te geven en te ontvangen (zie hoofdstuk 4, "Denkvaardigheden voor testen", in *More Agile Testen* voor aanvullende informatie)

We zien een groeiende behoefte aan testers om op te treden als testconsulent voor hun teams. Veel teams hebben een lage verhouding tussen toegewijde testers en ontwikkelaars en andere niet-testrollen. Wij testers kunnen meer waarde toevoegen door iedereen te helpen die de competenties nodig heeft om essentiële testactiviteiten uit te voeren. Iedereen in het team zal meer nadenken over testen en zich zo meer bewust zijn van de noodzaak om vanaf het begin kwaliteit in te bouwen.

Chapter 11: A Tester's New Role

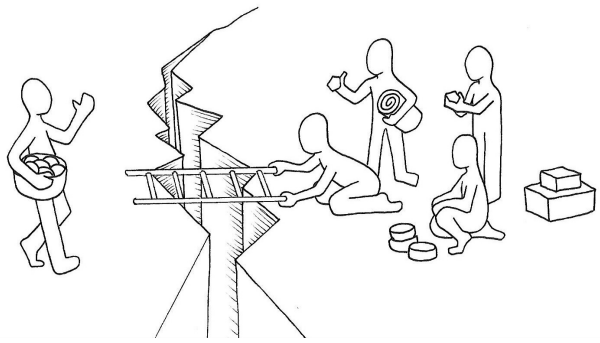
Many teams struggle with only one tester as part of the team – or even worse, a tester who supports more than one delivery team. If the tester is the only one doing testing activities, it generally creates a bottleneck. Agile teams who deliver changes to production at every iteration (or even more frequently if they're implementing continuous delivery) cannot afford to have a single tester to do all the testing.

We can't predict the future, but it is informative to look around to see how a tester's role is changing. We asked other experienced agile testing coaches and trainers to find out what their thoughts are about the changing role of a tester. These different perspectives may help you understand how testers and teams can adapt and help build a quality culture.

Testers are quality glue for a team

Alex Schladebeck – Germany

When I started out, the tester role in a team (if the tester was even a part of the team) was often to be solely responsible for UI automation and manual testing. Over the last 12 years, I've seen a huge diversification of the role, and always in a context-dependent way based on how the team operated. I see testers working on more automation tasks, even pairing at the unit level with developers. I see them being involved at all process points. I see them organizing mob testing sessions with the team to perform exploratory testing. I see them campaigning for better feedback loops. I've even seen testers start to fix bugs or implement features.



Make connections

For me, this diversification and role-blurring is a huge freedom and a great responsibility. It means that we have to ask ourselves, "How do I, my skills, and my potential to learn best fit into this team's context?" We become very much "quality and communication glue," identifying and filling in the gaps in any team.

I've started using the term "embedded quality engineer" or "embedded quality consultant" for this role. The problem with the title "tester" is that it contains the name of one of the many activities we do, so you hear questions like, "If everyone is involved in testing, why do we need a tester?" or statements like, "The developers are automating tests, so we don't need a tester role." Testing is just one of the many things that a

tester does. In my opinion, we need to fight against the idea that an agile team should consist of "chimeras": a mix of different roles, or a Swiss-Army-knife team member that can do everything: requirements, UX, test, security, front and backend. Agile teams need to be diverse and cross-functional, and that means we need people with different backgrounds, interests, and specializations, without any one person being a silo or bottleneck. I think that's a balance that is achievable.

I see the tester role as consisting of multiple activities. There are things that have always been part of the role, like working with stakeholders, bringing expertise to test activities, pairing with developers and other testers, supporting the product owner, organizing and adapting the overall quality strategy, getting good test data, and identifying risk. I think there will be even more tasks that can be taken on or supported by testers in the future. Some of these might be: working with the team to ensure testability and observability for testing and monitoring in production, asking questions of the production system to explore how it is being used, honing our performance and teaching skills for exploratory testing, helping the team to focus on value (and sometimes on minimalism, i.e., the value of something not done). I also see testers starting to add "team health" to their quality attributes, looking out for the communication and stress levels of the team as a whole. These are, after all, things that can affect quality greatly.

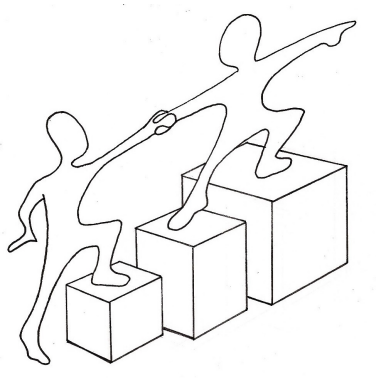
In short, I believe that the tester role remains important. They are the person in the team whose main priority is quality. They are also the person with the passion for quality and testing, and they advocate and champion for them.

An agile tester's professional journey

Paul Carvalho – Canada

When I coach agile teams, I help the whole team learn to work together and build upon each others' strengths. A product owner brings business and industry knowledge, a programmer brings strong coding and development skills, a designer brings insights into the user's perspective and experience, and a tester brings something of value to the table as well.

A tester's professional journey starts with moving away from accidental and random testing, to thoughtful design of tests through models, techniques, and other specialized skills and knowledge. Agile puts a strong focus on working closely with others, so testers need to get out of their heads when thinking about testing and find their voice to help other team members understand the many ways to generate quality information about the systems in development. A great agile tester spends more time with a whiteboard marker in hand and pairing with other team members than anything else. It's about helping the rest of the team to see and understand more about the system, before the system is built. "Build Quality In" is more than a slogan; it is a reality that great testers can help enable on high-performing collaborative teams.



Mentor

An agile tester's professional journey progresses from random, haphazard testing, to understanding the techniques and design of good testing, to finding a voice and expressing these ideas so they may help elevate the rest of your team's performance.

As a tester, aim to enhance and grow everyone's understanding of the systems and solutions through thoughtful exploration. If you let go of the notion that you must be the one to write test cases or program automation, think about where your unique skills and insights can help carry your team.

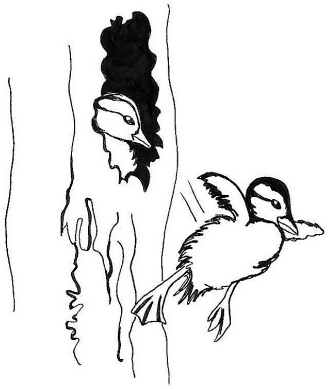
The fascinating path of evolving as testers

Claudia Badell – Uruguay

As testers, we can contribute and add value from different perspectives: as facilitators and evangelists toward testing and quality in a product team, as coaches, as test consultants, as experts at certain types of testing (usability testing, accessibility testing, security testing, performance testing, among others), and more. Testers are no longer seen as gatekeepers for quality, so we can be seen and valued as quality advocators.

Nowadays, it is more and more frequent that testers are part of the development team and make contributions from the beginning of the development process. In my experience, the role of a tester in this context is evolving. Besides performing testing activities to support manual testing and automated checks, testers collaborate to build bridges within the team in order to reach a common understanding and engagement about testing. They also define, follow up, and adjust the testing strategies to be applied by the whole team. In addition, they share and evangelize their knowledge of testing within the team.

As technology, methodologies, and processes evolve and teams and communities mature, I believe it is important for us to have a proactive attitude to adapt to such changes. The future will bring new challenges and opportunities. Depending on the context, different skills and different testing activities may be needed, but in my experience, there are core skills that are necessary to keep pace with software development.



Experiment and find new opportunities

These skills are:

- be eager to learn and try new experiments to enhance testing strategies in the team.
- be an excellent question asker throughout the product lifecycle. Depending on the type of information that we gather, questions can be formulated in different ways. They can be made verbally or in writing, so clear and well-structured communication skills are important. They can also be made programmatically; for example, if we wanted to check certain responses between two services, technical skills are important.
- have modeling skills as a way of understanding what to test and defining testing strategies that cover the different aspects that are needed.
- have some degree of technical knowledge to collaborate in defining testability aspects of the solution while the software is being developed – for example, to support unit testing and automated integration testing.
- have a sharing and collaborative attitude.

We are in an exciting time where we can shape part of our future. How are you getting prepared for it?

Be all that you can be

Mike Talks – New Zealand

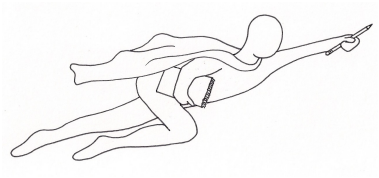
Over the last six years, my test team has gone from a single group working on one waterfall project at a time to individuals working across multiple teams.

There's a lot of talk about a whole-team approach to quality and testing, but testers as specialists are looked to lead in this space. That means creating a first-pass approach at a new story or feature, but it is also important to facilitate a discussion with the larger team about these approaches to gain feedback and explore the approach. It also means if a testing task is too onerous for testers to complete alone, they can help organise a division of labor among willing team members.

I find a frequent candidate for this is cross browser/device testing. We often test stories in the iteration using our core devices but will occasionally visit our product on a much broader range of items. This is where the

team and their fresh sets of eyes can help, and a little bit of organization from the tester can make a huge difference.

Although most conversations about testing a product happen within your team, it's also useful to "catch up" with others in the same disciplines to share ideas and what's been working on other teams. This can also turn into mentoring to help individuals deal with specific problems as well as become more daring to try new ideas.



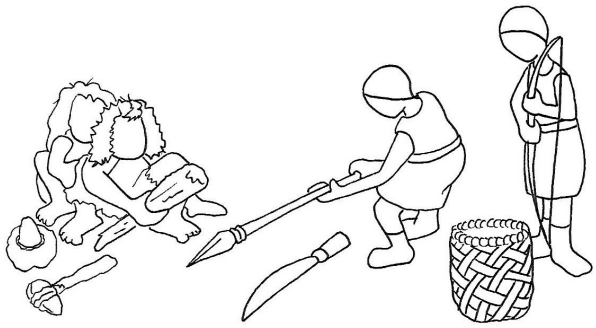
Be all you can be

Start with a conversation

Kathleen Naughton – United States

Software testing has simultaneously evolved and stood still. It has evolved to the point that some organizations have reduced the number or even eliminated testers on their teams. It has stood still because these same organizations have struggled to value the specialty skills a tester brings to teams.

In my experience, where testers have been reduced or eliminated, the teams have tried to fill in by moving closer to TDD practices so that they have a large number of automated unit tests. They try to do some intra-team testing (testing each other's code) and rely heavily on their continuous delivery pipeline to execute their automated tests. What often ends up being missed are the integration and user experience tests that are essential for high-quality products. If there are testers in these organizations, they are present to do manual testing after the code is finished. Any insights or suggestions made by these testers tend to be deprioritized in the product backlog in deferment to feature development.



Be relevant and influence those around you

I believe an essential skill that testers need to be relevant is being able to read and understand code. This allows them to understand what unit tests or other automated tests are checking and enables the identification of testing gaps that the tester can fill. It also allows for reduction in test overlap. If there are already unit

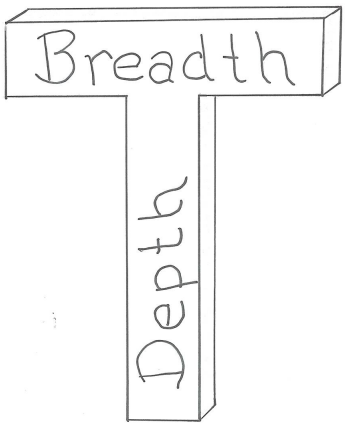
or integration tests present, the testing approach can be more targeted at the end-user activities that are not covered. Another essential skill I believe is needed might be considered a soft skill. The skill of having conversations with programmers about their unit and integration tests is powerful. These conversations can lead to influencing design decisions that in turn enable higher-quality deliverables. Bringing knowledge about how to have crucial conversations enables the whole team to produce better software.

The world doesn't need more checkers

Aldo Rall – New Zealand

Testing has evolved over the years, and the industry has developed test engineering skills and practices, although not smoothly. I think we have entered an era of testing enlightenment, and there is a big shift in how organizations and testers now think of the testing role.

I observe an increasing movement toward the importance of skills. The more skills an individual has, the more valuable they become for a team or organization. Those individuals with skills that span beyond test engineering skills only are the ones that can contribute more than someone who has a basic set of test design and execution skills. This idea is emphasized by the thinking about generalizing specialists as discussed by Scott Ambler (http://www.agilemodeling.com/essays/generalizingSpecialists.htm) or T-Shaped Skills as discussed by Lisa and Janet in their books. If you want to future-proof your career, build your test engineering skills, as well as skills outside the traditional world of testing. Forget about titles; in ten years' time it is not going to mean much to anyone if you were called "test engineer," "test specialist," "tester," "test analyst," or "verification engineer." Skills are ultimately more valuable than collected job titles; I have certainly found that true in my own career.



T-shaped skills

I would look at a holistic approach characterized by inclusive "and" conversations. The best example I can think of is to combine your collection of skills in unique ways that suits your specific context, knowing that even that will change. How can you combine a set of test engineering and analysis skills in a given situation? How can you combine a set of negotiating skills with teaching skills? How can you combine different test engineering skills together to obtain better test coverage? Think "and."

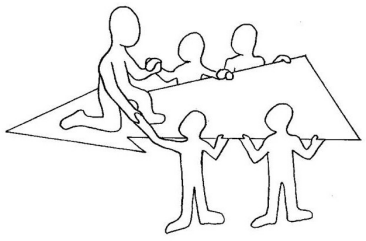
I believe one of the key skills a modern practitioner must have is the ability to understand a context, understand its changing nature, and adjust accordingly. The true masters of the future workplace will be those who will be able to observe a context, apply the fit-for-purpose combination of skills, and then continually adjust the combination of skills as the context evolves. That intuition takes time to develop, and it is a constantly changing domain. Learning new skills will enrich the capability and the value that such a person brings to organizations and teams.

We bring many different skills than just test engineering skills. I would like to suggest that we consider (by unashamedly borrowing from others) a multifaceted approach. Call it the "Holistic agile testing skills" or "The ten thinking hats of agile testing," or whatever else makes sense to you. Some of these facets might be:

- **Consultant**: Yes, sometimes we will have to "consult" inside our team or with another team to help solve problems and issues.
- **Test engineering specialist**: We have to know our testing onions from shallots. We require good and solid test engineering skills.
- **Agile scholar**: Keep studying and learning about agile, bring ideas to the team, and experiment.
- **Coach**: We have great opportunities to coach the team or even co-workers (inside and outside the team). This is a life skill, in my opinion.
- **Mentor**: Sometimes we have to mentor someone in testing.
- **Facilitator**: Sometimes we just need to step into the role of facilitator for a decision, discussion, explanation, etc.
- **Change agent**: We may sometimes bring about change and upheaval, advocate a new practice/technique/method to experiment with and learn from.
- **Leader**: Yes, we may sometimes be required to step up and perform leadership in/on behalf of the team.
- **Teacher**: That goes without saying, especially if there is a shortage of testing skills in the team/organization.
- **Business domain scholar / defender of common sense / big picture thinker**: Sometimes it is good to step away and see the forest from the trees.

Lisa's and Janet's thoughts

We hope you've enjoyed reading other people's thoughts about what they consider to be a tester's role. Now we'll share ours. We've encouraged testers to help their non-tester teammates learn testing skills. When the whole team takes responsibility for quality and testing, every team member needs some grounding in testing skills. To accomplish this, we've identified skills we recommend that testing specialists should learn.



Use your thinking skills

- **collaboration skills** to participate actively in practices like mind mapping or example mapping
- **facilitation skills** to help team members communicate better, facilitate meetings such as retrospectives, facilitate workshops to help non-testers learn testing skills
- **teaching skills** to share their knowledge with other team members
- **coaching skills** to help the team identify problems and design experiments for improvements
- **communication skills** to give and receive feedback effectively (see Chapter 4, "Thinking Skills for Testing," in *More Agile Testing* for additional information)

We see a growing need for testers to act as test consultants for their teams. Many teams have a low ratio of dedicated testers to developers and other non-testing roles. We testers can add more value by helping everyone have the competencies needed to do essential testing activities. Everyone on the team will think more about testing and be more conscious of the need to build quality in from the start.

Hoofdstuk 12: Ingrediënten voor succes

Elk agile software delivery-team bewandelt zijn eigen leertraject. Ons doel is om voortdurend ons vermogen te verbeteren om waarde te leveren aan onze klanten en op regelmatige basis, met behoud van de gewenste kwaliteitsstandaard van het bedrijf. Elk team doet dit met een unieke combinatie van het bedrijfsdomein, softwareproduct, technologiestack, framework en praktijken.

In de loop der jaren hebben we ontdekt dat tussen al deze verschillen, bepaalde ingrediënten voor succes zitten waar elk team van kan profiteren.

Succesfactoren

In ons eerste boek, *Agile Testing*, bevatte ons samenvattend hoofdstuk zeven succesfactoren waarvan we dachten dat ze nodig waren (hoewel niet voldoende) om succesvol te zijn in het leveren van een kwaliteitsproduct. Het is gemakkelijk overweldigend te zijn door het plannen en uitvoeren van testactiviteiten gedurende korte delivery cycli. Hieronder vindt u een korte lijst van de belangrijkste succesfactoren en kern agile testpraktijken om uw teams te begeleiden.

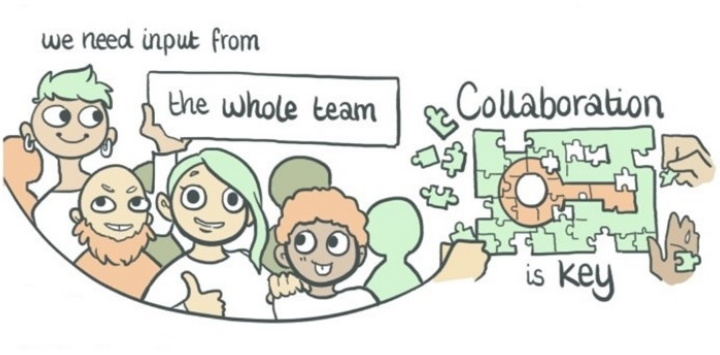
"Teambrede aanpak"

Elisabeth Hendrickson leerde ons dat "testen een activiteit is, geen fase." Testen is een integraal onderdeel van softwareontwikkeling, samen met coderen en zoveel andere activiteiten. Met dit perspectief is het gemakkelijk voor iedereen om te helpen met testtaken mocht het nodig zijn.

Testers kunnen andere teamleden vaardigheden leren, zoals het uitlokken van concrete voorbeelden van gewenst en ongewenst gedrag van bedrijfsdeskundigen, het evalueren van verschillende kwaliteitsattributen of het doen van onderzoekende testen.

Programmeurs kunnen testers helpen de systeemarchitectuur te begrijpen om beter te testen of ze zelfs een basis van coderen aan te leren. Elk teamlid kan sommige van zijn diepgaande vaardigheden overdragen aan andere teamleden, ongeacht de rol.

Wanneer teams zich realiseren dat testen en kwaliteit een teamprobleem zijn, kunnen ze hun diverse vaardigheden integreren en een sfeer van vertrouwen en veiligheid ontwikkelen, alsook een leeromgeving creëren waar ze kunnen experimenteren en continu verbeteren.



Drawing by Constance Hermit

Agile test mentaliteit

Testers zijn niet langer de 'kwaliteitspolitie' die de 'go/no-go' bepalen. Testers of teamleden die testactiviteiten uitvoeren kunnen de risico's en de impact van testresultaten uitleggen, zodat het bedrijf een weloverwogen beslissing kan nemen over de uitrol naar productie.

Als teamlid met een agile-testmentaliteit betekent dit dat je leergierig bent en meer wil weten over alles om je te helpen bij het uitvoeren van uw job. Het betekent dat je de agile principes en waarden toepast. Het betekent ook samenwerken met de technische en zakelijke teamleden, waarbij je het grote geheel in gedachten houdt terwijl je de kleine functionaliteiten incrementeel toevoegt. Je bent gefocust op het voorkomen van bugs, op deze manier hoef je later niet zoveel tijd te spenderen aan het vinden van bugs.

Drawing by Constance Hermit

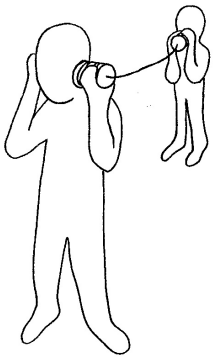
Automatiseer uw regressietesten

Er zijn een paar dingen om te onthouden wanneer uw team begint te automatiseren. Het is een teamprobleem, dus denk aan "het hele team" en werk samen om te automatiseren op alle niveaus. Programmeurs zijn goed in het schrijven van code, testers zijn goed in het specificeren van de testen en mensen met andere gespecialiseerde vaardigheden in het team kunnen helpen met testgegevens, infrastructuur en meer. De testautomatisering piramide is een goed visueel model om de automatisering van het team te vormen en de strategie te laten evolueren. Door de testen eenvoudig en gemakkelijk onderhoudbaar te houden, kan een team werken aan het hebben van voldoende regressietests om hen vertrouwen te geven om de applicatie uit te rollen.

Testautomatisering is een controle om er zeker van te zijn dat je niet bent vergeten iets te veranderen, d.w.z. het is een veranderingsdetector. Een goede automatisering strategie geeft u de tijd om verkennende tests uit te voeren om problemen te vinden voordat uw klant dat doet.

Geef en ontvang feedback

Succesvolle softwareontwikkeling is afhankelijk van snelle feedback. Teams moeten meteen weten of een wijziging een onbedoelde storing heeft veroorzaakt. Zij willen weten hoe klanten reageren op een nieuwe functionaliteit. Testers staan centraal bij het creëren en het blijvend verkorten van de verschillende feedbackloops, waaronder: geautomatiseerde tests maken, verkennend testen en het observeren van het productiegebruik om te leren hoe klanten het product gebruiken.



Mensen hebben ook feedback voor zichzelf nodig, zodat ze meer manieren kunnen vinden om waarde toe te voegen. Luister- en observatievaardigheden staan centraal.

> ***Hint***: *Als je samenwerkt met andere teamleden, vraag ze dan welke hiaten jij kan opvullen en hoe je effectiever kan bijdragen.*

Bouw een basis van kerntechnieken

Er zijn kerntechnieken die effectief zijn gebleken bij het helpen van teams om kwaliteit in hun product in te bouwen.

- Elk team heeft **continue integratie** nodig om succesvolle software te kunnen leveren met een frequente cadans in de tijd. Elke keer dat een teamlid een wijziging begaat in de broncode, zou er een automatisch een bouwproces moeten starten dat alle codewijzigingen integreert en deze verifieert met de geautomatiseerde tests. Elk team heeft een deployment pipeline nodig om een release-kandidaat te maken en deze in te zetten voor een test of productieomgeving. Zelfs ook als deze handmatige fasen bevat.
- Veel teams hebben nog steeds moeite om betrouwbare **testomgevingen** te hebben die zoveel mogelijk op productie lijken en hen in staat stellen om eenvoudig bepalen welke buildversie er uitgerold is. De cloud infrastructuur van vandaag biedt nog meer mogelijkheden om tijdelijke test omgevingen aan te maken om een specifieke buildversie te testen en automatisch nieuwe versies uit te rollen in permanente testomgevingen.
- Technische schuld is als kredietkaartschuld; het blijft groeien als alleen het minimumbedrag of een deel van de rente wordt betaald. De technische schuld blijft groeien als het team geen tijd neemt om de code te refactoren, creëren van adequate geautomatiseerde regressietesten, upgrade van frameworks, en het beheren van andere noodzakelijke infrastructuur. Na een tijdje kan zelfs de kleinste codewijziging grote risico's met zich meebrengen en veel tijd vereisen aan handmatige testtaken. Investeer de tijd in het **managen van technische schuld** in de code en in uw geautomatiseerde tests.

- Kleine, frequente veranderingen zijn over het algemeen minder riskant dan grote, zeldzame. Er kan minder fout gaan en mislukkingen kunnen sneller gediagnosticeerd worden. Teams die grote functionaliteiten opdelen in kleine '**lerende releases**" en zelfs kleinere stories hebben meer kans om op tijd te leveren wat hun klanten willen.
- **Progammeren en testen zijn onderdeel van één proces**. Dit is de kern van de teambrede aanpak van testen en kwaliteit. Testen en programmeren gebeurt samen, hand in hand, van feature-idee tot evaluatie van de functie in de productie.
- **Synergie tussen praktijken** komt van het doen van al deze kernpraktijken samen. Test-driven Development, collectieve code eigendom en continue integratie zorgen voor consistentie en snelle feedback. Refactoring is afhankelijk van het al dan niet hebben van geautomatiseerde regressietesten. Binnen agile-praktijken is dit uitgetest en werkend. Deze praktijken zijn dan ook ontworpen om dit allemaal samen te doen.

Samenwerken met klanten

Testers spreken de domeintaal van zakelijke stakeholders en de technische taal van de leden van het opleverteam. De juiste mensen samenkrijgen wanneer een gesprek nodig is over hoe een functie zich moet

gedragen of hoe een ontwerp eruit moet zien is ook belangrijk. Testers kunnen product owners helpen bedrijfsregels te formuleren voor elk verhaal en deze illustreren met concrete voorbeelden. Dit is een van de meest waardevolle manieren waarop testers bijdragen aan teams.

Kijk naar het grote geheel

Terwijl agile teams zich concentreren op kleine veranderingen en kleine stukjes functionaliteit, moeten ze op elk moment ook het grote geheel in gedachten houden. Testers zijn getalenteerd in het identificeren van welke delen van het systeem beïnvloed worden door een bepaalde kleine verandering, en ze hebben het perspectief van een klant.

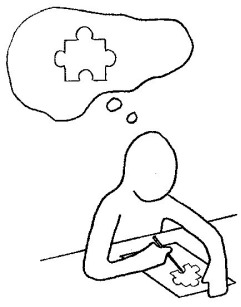
Het agile testkwadrantenmodel helpt het team het grote geheel in gedachten te houden bij het plannen van testactiviteiten. Verkennende testen zijn een voorbeeld van een testactiviteit die onverwachte gevolgen van een nieuwe toepassingsmogelijkheid kan ontdekken. We hebben goede manieren om te analyseren hoe klanten ons product gebruiken. Dit helpt ons allemaal focus houden op het leveren van de juiste waarde.

Praktijken voor het opbouwen van vertrouwen

In ons tweede boek, *More Agile Testing*, hebben we een aantal test kernpraktijken geïdentificeerd die teams helpen het vertrouwen op te bouwen dat nodig is om regelmatig veranderingen in de productie uit te rollen. Deze praktijken zijn vooral belangrijk naarmate meer teams evolueren naar continuous delivery of continuous deployment.

Gebruik voorbeelden

Concrete voorbeelden van hoe een applicatie zich zou moeten gedragen helpt iedereen in het team de bedrijfsregels te begrijpen. Deze voorbeelden kunnen omgezet worden in testen die de ontwikkeling begeleiden. Ze kunnen zo worden geautomatiseerd dat het team weet wanneer de story of functionaliteit klaar is. De geautomatiseerde testen worden onderdeel van de regressietestsuites die snelle feedback geven of een nieuwe wijziging het bestaande productiegedrag heeft beïnvloed. Voorbeelden helpen teams op het juiste pad te blijven.



Verkennende testen

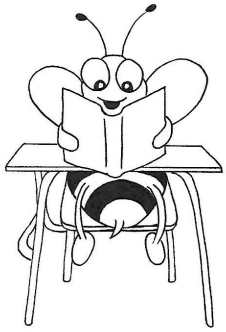
Door regressietests te automatiseren, blijft er meer tijd over voor verkennende testen, een van de beste manieren om de “onbekende onbekenden” te vinden die ernstige productiestoringen kunnen veroorzaken. Programmeurs kunnen leren verkennende testen te doen op elke story voordat ze hun werk als ‘af’ beschouwen. Dit is nog een snelle feedbacklus. Iedereen in het team kan verkennende testvaardigheden leren en gebruiken. Ze zullen niet alleen onverwachte problemen identificeren, maar vinden ook ontbrekende mogelijkheden die worden teruggekoppeld naar nieuwe functionaliteiten.

Functionaliteit testen

Het is essentieel om op alle detailniveaus te testen. Omdat agile teams zich focussen op het verhaal, moeten ze onthouden om ook op functionaliteit niveau te testen. Een belangrijk onderdeel hiervan is het identificeren van wat de functionaliteit werkelijk hoeft te omvatten. Testers kunnen helpen erachter te komen wat waardevol is voor de klanten door vragen te stellen over wat het bedrijf zou moeten laten vallen ten voordele van het leveren van andere zeer waardevolle functionaliteiten.

Voortdurend leren

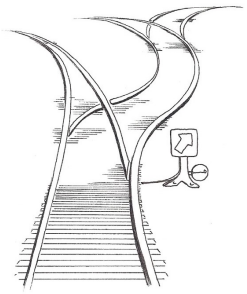
Teamsucces hangt af van de psychologische veiligheid, vertrouwen en tijd om te leren. Het team moet samenwerken om het grootste obstakel te identificeren voor het leveren van de gewenste kwaliteitsniveau, of het nu onvoldoende test automatisering, feedback die te lang duurt of het bouwen van functionaliteiten die niemand wilde. Dan kunnen ze kleine experimenten ontwerpen om te beginnen met het overwinnen van dat obstakel. Testers kunnen het team helpen kwaliteit in te bouwen door testvaardigheden over te dragen. Andere teamleden kunnen testers helpen hun T-vormige vaardigheden te vergroten zodat ze op meerdere manieren een meerwaarde zijn.



Contextgevoeligheid

Elk team werkt binnen zijn unieke context. De grootte van het bedrijf, het zakelijke domein en zijn regelgevende omgeving, de betrokken technologie, infrastructuurbehoeften - dit zijn slechts enkele overwegingen voor een team als het overweegt hoe het zijn capaciteit kan verbeteren om regelmatig waarde aan klanten te

leveren. Gebruik geen tool of oefening omdat het is wat Google of Facebook doet - gebruik wat geschikt is voor uw context.



Hou het realistisch

Testers blinken uit in het geven van feedback. Het kan moeilijk zijn om slecht nieuws te leveren. Maar het is belangrijk om realistisch te blijven. Als een wijziging riskant is en het team heeft dat risico niet voldoende gemitigeerd met testen en andere activiteiten, moeten stakeholders dit weten. Testers kunnen fungeren als raadgevers om iedereen in het team te helpen hun testvaardigheden te verbeteren en in staat zijn om bezorgdheden over de kwaliteit zichtbaar te maken voor het bedrijf. Wanneer het team knelpunten ervaart bij het testen, maak dit dan zichtbaar, maak er een teamprobleem van om op te lossen. Het kan verleidelijk zijn om problemen te verdoezelen om de stakeholders tevreden te houden, maar ze zullen niet gelukkig zijn als de klanten de pijn ervaren.

Het is ook belangrijk voor het moraal van het team om te weten dat ze ‘nee’ kunnen zeggen. Bijvoorbeeld: “Nee, we kunnen geen stories meer opnemen”, of “Nee, we kunnen geen nieuwe story toevoegen, tenzij je één van de andere verwijdert.” Houd het realistisch!

Paden naar succes

We weten dat er veel testers en teams zijn die stress voelen, vooral bij teams die nog steeds aan het omvormen zijn naar het gebruik van de agile ontwikkelingswaarden, principes en praktijken. Bijvoorbeeld, het management is er nog niet achter hoe hun rol moet veranderen en kan frequentere leveringen eisen en onrealistische deadlines opleggen. De testen moeten nog gebeuren en in teveel gevallen dragen testers nog steeds volledige verantwoordelijkheid voor alle testactiviteiten. We denken graag dat er een magische applicatie is die al onze problemen zal oplossen, maar we weten dat dit niet de realiteit is!

Testproblemen omzetten in problemen voor het hele leveringsteam is essentieel om te leren hoe je kwaliteit in je producten kunt inbouwen en duurzaam succes kan behalen. Deze belangrijke succesfactoren en vertrouwenwekkende praktijken bieden een kader om het team te helpen beslissen over de volgende stappen op weg naar verbetering.

In onze ervaring duurt het jaren voordat een leveringsteam hun gewenste prestatie- en kwaliteitsniveau behaald. We kunnen een achtste sleutel toevoegen aan deze succesfactor: geduld! Regelmatige teamretrospectieven (we raden aan voor minstens één per week bij nieuwe teams) zijn ook essentieel bij het identificeren van het grootste kwaliteitsgerelateerde probleem en het ontwerpen van een klein experiment om te beginnen

dit te verbeteren. De diverse vaardigheden en ervaringen in een multifunctioneel team maakt het oplossen van deze problemen veel gemakkelijker.

Een voorbeeld

Het team is gefrustreerd omdat de product owner een hoog percentage afwijst van de stories die ze opleveren. Het constante herwerk, soms dagen nadat het team dacht dat de story "af" was, vertraagt hen. Cyclustijd – de tijd vanaf het moment dat ze aan een story beginnen te werken tot wanneer het wordt opgeleverd in productie – is veel langer dan ze zouden willen. Welke belangrijke succesfactor kan helpen?

De teambrede aanpak is duidelijk. Laten we het hele team, of een representatieve groep inclusief alle rollen, samenkomen om het te bespreken. Wat vertrouwen opbouwen zou helpen? Wanneer de product owner een story weigert, is dat meestal omdat het gedrag van dat deel van de toepassing, niet het gedrag is dat hij/zij voor ogen had. Het team heeft de vereisten verkeerd begrepen. Het team leert voortdurend bij (één van de vertrouwenwekkende praktijken), en één van de testers heeft zojuist geleerd over example mapping. Ze besluiten te experimenteren met example mapping om te kijken of het een beter gedeeld begrip van de story zal opleveren. Zij veronderstellen dat example mapping het afwijzingspercentage voor de stories met 20% zal verminderen in de komende twee weken, wat zal resulteren in een besparing van 10% op de gemiddelde cyclus tijd.

Ze meten en experimenteren om te zien of hun hypothese waar is. In dit echt voorbeeld, was het experiment een succes. De doelen voor de vermindering van het afwijzingspercentage en de cyclustijd werden overschreden. Binnen de twee maanden werden het afwijzingspercentage en de cyclustijd beide met 50% verminderd. Het team vond meer voordelen van example mapping omdat het hielp bij het specificeren van de scenario's om de ontwikkeling te sturen in de vorm van gedragsgestuurde ontwikkelingstesten. Maar als het experiment zou mislukt zijn, zou het team een nieuw experiment ontwerpen, geleid door de succesfactoren en vertrouwenwekkende praktijken.

Wanneer onze eigen teams het gevoel hebben vast te zitten met een probleem, vallen we terug op de sleutel succesfactoren en vertrouwenwekkende praktijken. Dit samen met de tien principes voor agile testen uit hoofdstuk 1, om ons te helpen bij het plannen van onze volgende stappen. Ze zullen ons begeleiden tijdens onze leerreis terwijl we ons vermogen verbeteren om regelmatig kleine, waardevolle wijzigingen bij onze klanten door te voeren.

Bedankt voor het lezen, en we hopen dat je succesvol bent in uw eigen reis.

Chapter 12: Ingredients for Success

Each agile software delivery team travels its own learning journey. Our goal is to continually improve our ability to deliver value to our customers frequently, while maintaining our business's desired standard of quality. Each team is doing this with a unique combination of business domain, software product, technology stack, frameworks, and practices.

Over the years, we have found that among all these differences, certain ingredients for success benefit every team.

Success factors

In our first book, *Agile Testing*, our summary chapter comprised seven success factors we thought were necessary (although not sufficient) to be successful in delivering a quality product. It is easy to get overwhelmed by planning and executing testing activities during short delivery cycles. Below is a short list of key success factors and core agile testing practices to guide your teams.

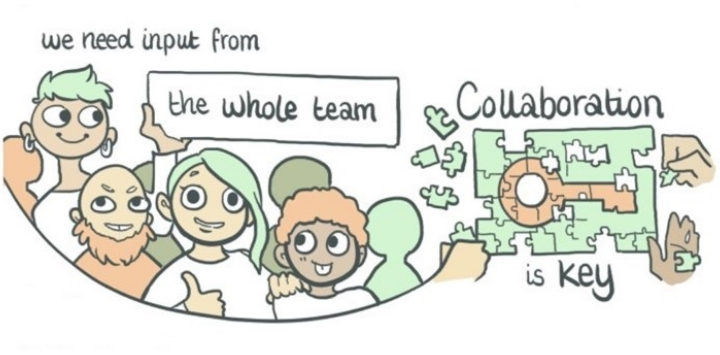
"Whole-team approach"

Elisabeth Hendrickson taught us that "testing is an activity, not a phase." Testing is an integral part of software development, along with coding and so many other activities. With this perspective, it is easy for everyone to help with testing tasks as necessary.

Testers can teach other team members skills like eliciting concrete examples of desired and undesired behavior from business experts, evaluating different quality attributes, or doing exploratory testing.

Programmers can help testers understand the system architecture to get better testing or even teach them basic coding constructs. Each team member can transfer some of their deep skills to other team members, regardless of role.

When teams realize that testing and quality are a team problem, they can incorporate their diverse skillsets and develop an atmosphere of trust and safety, as well as create a learning environment where they can experiment and continually improve.



Drawing by Constance Hermit

Agile testing mindset

Testers are no longer the "quality police," determining "go/no-go" decisions. Testers or team members who are performing testing activities can explain the risks and impacts of test results so that the business can make an informed decision about releasing to production.

As a team member with an agile testing mindset, it means you're inquisitive and want to learn more about everything to help you do your job. It means that you apply agile principles and values. It means collaborating with the technical and business team members, keeping the big picture in mind as you put the small feature increments together. You're focused on bug prevention, so you don't have to spend so much time finding bugs later.

Drawing by Constance Hermit

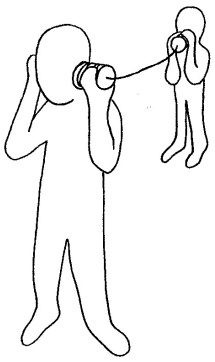
Automate your regression tests

There are a few things to remember when your team starts to automate. It is a team problem, so think "whole team" and collaborate to automate at all levels. Programmers are good at writing code, testers are good at specifying tests, and people with other specialized skills on the team can help with test data, infrastructure, and more. The test automation pyramid is a good visual model to form and evolve the team's automation strategy. By keeping the tests simple and easy to maintain, a team can work toward having enough regression tests to give them confidence about releasing.

Test automation is a check to ensure that you haven't forgotten to change something, i.e., it is a change detector. A good automation strategy gives you the time to perform exploratory testing to find issues before your customer does.

Provide and obtain feedback

Successful software development depends on fast feedback. Teams need to know right away if a change has caused an unintended failure. They want to know how customers react to a new feature. Testers are central to creating and continuing to shorten the various feedback loops, including creating automated tests, engaging in exploratory testing, and observing production usage to learn how customers use the product.



People also need feedback for themselves so they can find more ways to add value. Listening and observing skills are key.

> ***Hint***: *As you collaborate with other team members, ask them what gaps you can fill and how you can contribute more effectively.*

Build a foundation of core practices

There are core practices that have proven effective in helping teams build quality into their product.

- Every team needs **continuous integration** to successfully deliver software at a frequent cadence over time. Each time a team member commits a change to the source code control repository, it should kick off a build process that integrates all code changes and verifies them with automated tests. Every team

has a deployment pipeline to create a release candidate and deploy it to a test or production environment – even if it includes manual stages.
- Many teams still struggle to have reliable **test environments** that resemble production as much as possible and allow them to easily control which build version is deployed. Today's cloud infrastructure provides even more options to create temporary test environments to test a specific build version and automatically deploy new versions to permanent test environments.
- Technical debt is like credit card debt; it continues to grow if only the minimum amount or part of the interest is paid. The technical debt continues to grow if the team doesn't take time to refactor code, create adequate automated regression tests, upgrade frameworks, and manage other necessary infrastructure. Over time even the smallest code change poses large risks and requires much time spent with manual testing tasks. Invest the time to **manage technical debt** in the code and in your automated tests.

- Small, frequent changes are generally less risky than large, infrequent ones. Less can go wrong, and failures can be quickly diagnosed. Teams that slice big feature ideas into small "**learning releases**" and even smaller stories are more likely to deliver what their customers want in a timely manner.
- **Coding and testing are part of one process**. This is the core of the whole-team approach to testing and quality. Testing and coding happen together, hand in hand, from feature idea to evaluating the feature in production.
- **Synergy between practices** comes from doing all these core practices together. Test-driven development, collective code ownership, and continuous integration ensure consistency and fast feedback. Refactoring depends on having automated regression tests. Agile practices are tried and true and are designed to be done all together.

Collaborate with customers

Testers speak the domain language of business stakeholders and the technical language of delivery team members. Getting the right people together when a conversation is needed about how a feature should behave or how a design should look is also important. Testers can help product owners articulate business rules for each story and illustrate them with concrete examples. This is one of the most valuable ways testers contribute on teams.

Look at the big picture

While agile teams focus on small changes and small slices of features at any given time, they also need to keep the big picture in mind. Testers are talented at identifying what parts of the system might be affected

by a particular small change, and they have a customer's perspective.

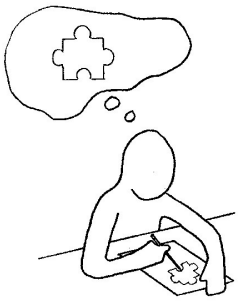
The agile testing quadrants model goes a long way toward helping the team keep the big picture in mind as they plan testing activities. Exploratory testing is an example of a testing activity that can uncover unexpected consequences of a new application capability. We have good ways to analyze how customers are using our product. This all helps us focus on delivering the right value.

Confidence-building practices

In our second book, *More Agile Testing*, we identified some core testing practices that help teams build the confidence needed to frequently release changes to production. These practices are especially important as more teams move toward continuous delivery or continuous deployment.

Use examples

Concrete examples of how an application capability should behave help everyone on the team understand the business rules. These examples can be turned into tests that guide development. They can be automated so the team knows when it's done with a story or feature. The automated tests become part of regression test suites that provide quick feedback on whether a new change has affected existing production behavior. Examples help teams stay on track.



Exploratory testing

Automating regression tests leaves more time for exploratory testing, one of the best ways to find the "unknown unknowns" that could cause dire production failures. Programmers can learn to do exploratory testing on each story before they deem their work "finished." This is another fast feedback loop. Everyone on the team can learn and use exploratory testing skills. They will not only identify unexpected problems but will also find missing capabilities that feed back into new feature ideas.

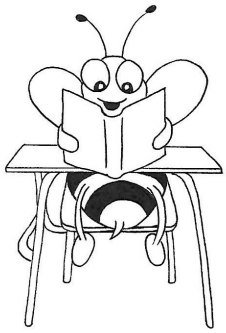
Feature testing

It's essential to test at all levels of detail. Because agile teams focus on the story, they need to remember to also test at the feature level. An important part of this is identifying what the feature really needs to include.

Testers can help figure out what is valuable to customers by asking questions concerning what the business should drop in favor of delivering other highly valuable features.

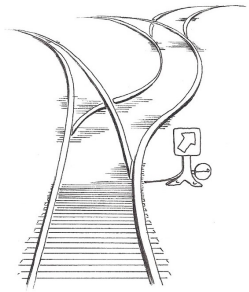
Continual learning

Team success depends on psychological safety, trust, and time to learn. The team needs to work together to identify the biggest obstacle to delivering their desired level of quality, whether it's inadequate test automation, feedback that takes too long, or building features that nobody wanted. Then they can design small experiments to start overcoming that obstacle. Testers can help the team learn to build quality in by transferring testing skills. Other team members can help testers ramp up their T-shaped skills so they can contribute in more ways.



Context sensitivity

Every team is working within its unique context. The size of the company, the business domain and its regulatory environment, the technology involved, infrastructure needs – these are just some considerations for a team as it considers how to improve its ability to deliver value to customers frequently. Don't adopt a tool or practice because it's what Google or Facebook does – use what is appropriate for your context.



Keep it real

Testers excel at providing feedback. It can be difficult to deliver bad news. But it's important to stay grounded in reality. If a change is risky and the team hasn't adequately mitigated that risk with testing and other activities, business stakeholders need to know. Testers can act as consultants to help everyone on their team

improve their testing skills and be able to make quality concerns visible to the business. When the team experiences bottlenecks with testing, make it visible, make it a team problem to solve. It can be tempting to gloss over issues to keep the business executives happy, but they won't be happy if customers experience pain.

It is also important for team morale to know they can say no. For example, "No, we can't take in any more stories," or "No, we can't add a new story unless you remove one of the others." Keep it real!

Paths to success

We know there are a lot of testers and teams out there who feel stressed, especially on teams that are still transitioning to using agile development values, principles, and practices. For example, management hasn't yet figured out how their role needs to change and may demand more frequent deliveries and impose unrealistic deadlines. The testing still must be done, and in too many cases, testers still bear total responsibility for all testing activities. We'd like to think there is some magic silver bullet tool out there that will solve all our problems, but we know that's not reality!

Turning testing problems into problems for the whole delivery team to address is vital to learning how to build quality into your products and achieving sustainable success. These key success factors and confidence-building practices provide a framework to help the team decide its next steps along a path to improvement.

In our experience, it takes years for a delivery team to achieve their desired level of performance and quality. We might add an eighth key success factor: patience! Frequent team retrospectives (we recommend at least one a week for new teams) are also key in identifying the biggest quality-related problem and designing a small experiment to start chipping away at it. The diverse skills and experience in a cross-functional team makes solving those problems much easier.

An example

The team is frustrated because the product owner rejects a high percentage of the stories they deliver. The constant re-work, sometimes days after the team thought the story was "finished," is slowing them down. Cycle time – the time from when they start working on a story to when it is deployed to production – is much longer than they'd like. What key success factor can help?

The whole-team approach is obvious. Let's get the whole team, or a representative group including all roles, together to discuss it. What confidence-building practice would help? When the product owner rejects a story, it's usually because the behavior of that part of the application is not what she wanted. The team misunderstood the requirements. The team is continually learning (one of the confidence-building practices), and one of the testers has just learned about example mapping. They decide to experiment with example mapping to see if it will build better shared understanding of each story. They hypothesize that example mapping will reduce story rejection rate by 20% over the next two weeks, resulting in a 10% savings of average cycle time.

They measure and experiment to see if their hypothesis is true. In this real example, the experiment was a success. The goals for reduction in rejection rate and cycle time were exceeded. Within two months, rejection rate and cycle time were both reduced by 50%. The team found more benefits from example mapping as it

helped with specifying scenarios to guide development in the form of behavior-driven development tests. But if the experiment had failed, the team would have designed another experiment, guided by the success factors and confidence-building practices.

When our own teams feel stuck on a problem, we fall back on the key success factors and confidence-building practices, along with the ten principles for agile testing in Chapter 1, to help us plan our next steps. They will guide us along our learning journey as we improve our ability to get small, valuable changes to our customers frequently and sustainably.

Thank you for reading, and we hope you are successful in your own journey.

Begrippenlijst

Ad hoc testen: Een informele testactiviteit waarbij men bugs zoekt op een ongestructureerde manier zonder plan vooraf.

Context diagram: Een diagram op hoog niveau dat alle externe entiteiten weergeeft die kunnen interacteren met een systeem, inclusief andere systemen, omgevingen en activiteiten.

Klant: Extreme programming (XP) gebruikt de term "klant" om te verwijzen naar een business stakeholder, product persoon of eind- gebruiker die bijeenkomt met het ontwikkelteam om prioriteiten te stellen, vragen te beantwoorden en besluiten te nemen over gewenste functionaliteit. In moderne agile teams kan de term verwijzen naar een of meer business stakeholders, leden van een productteam, eind gebruikers en/of iedereen die helpt bij het ontwikkelen en accepteren van opgeleverde stories.

Eindspel: Het eindspel is de tijd vóór release, wanneer het ontwikkelteam de laatste hand legt aan het product. Niet een periode om bugs of technische achterstanden te fixen, maar een mogelijkheid om samen te werken met groepen buiten het ontwikkelteam om de software naar productie te brengen. Voorbeelden van eindspelactiviteiten zijn aanvullende testen van database migratie en installatie.

Iteratie: Timebox gebruikt voor planning, met de intentie dat een "potentieel opleverbaar product" beschikbaar is op het einde van de iteratie. De Scrum term voor iteratie is "sprint". Plannen in tijdblokken van twee weken is een tegenwoordig een gangbare praktijk, zelfs bij teams die continous delivery toepassen en vaker naar productie gaan.

Kanban: Een aanpak om te plannen die is afgeleid van Lean manufacturing waarbij teams werken op een flow gebaseerde manier. Zij gebruiken limieten voor werk in uitvoering (WIP) en pakken nieuw werk op dat klaar staat (Ready) wanneer een lege plek beschikbaar komt. Het team plant, wanneer nodig, een paar nieuwe stories tegelijk.

Leer Release: De eerste release(s) opgeleverd aan een klant, voor feedback om van te leren en aan te passen. **The Learning Release**[43].

Mind map: Een visueel diagram gebruikt als gereedschap voor brainstorming, vooral wanneer meer mensen tegelijk samenwerken. Het begint met een concept, idee of onderwerp als startpunt, waarbij nieuwe ideeën worden verbonden met dit startpunt en met elkaar. Mind maps kunnen goed werken voor test planning en andere activiteiten.

Pairing: (pair programmeren, pair testen) Twee personen die zij aan zij werken aan hetzelfde werkstation, bij voorkeur met twee gespiegelde monitoren, twee toetsenborden en twee muizen, om samen productie- of testcode te schrijven of andere testactiviteiten uit te voeren. Beschikken over twee personen, elk met verschillend perspectief en vaardigheden, helpt om problemen in een vroeg stadium op te vangen en betere oplossingen te bereiken. Bij de strikte stijl van pairing is één persoon, de navigator, vrij om te observeren en suggesties te doen, terwijl de andere persoon optreedt als bestuurder. De navigator en bestuurder rollen worden regelmatig gewisseld.

Toestandsdiagram: Een visuele techniek gebruikt als abstracte beschrijving van het **gedrag**[44] van een

[43] https://medium.com/@Ardita_K/the-learning-release-70374d2450b3
[44] https://nl.wikipedia.org/wiki/Gedrag

systeem[45] in reactie op diverse gebeurtenissen. Dit gedrag wordt geanalyseerd en weergegeven als aaneenschakeling van gebeurtenissen die kunnen plaatsvinden in één of meer toestanden.

Test-driven Development (TDD): Bij test-driven development, schrijft en automatiseert de programmeur een kleine unittest die initieel faalt. Pas daarna wordt een minimale hoeveelheid programmacode geschreven waardoor de test slaagt. De structuur van de programmacode wordt waar nodig verbeterd om te voldoen aan acceptabele standaarden. De productiecode wordt zo stap voor stap geschreven. TDD, ook bekend als Test-driven Design, is meer een code ontwerp praktijk dan een testactiviteit. TDD helpt om robuuste, gemakkelijk onderhoudbare code te schrijven.

[45] https://nl.wikipedia.org/wiki/Systeem

Glossary

Ad hoc testing: An informal testing activity where one looks for bugs in a non-structured way without any advance plan.

Context diagram: A high-level diagram that represents all external entities that may interact with a system, including other systems, environments, and activities.

Customer: Extreme programming (XP) uses the term "customer" to refer to a business stakeholder, product person, or end user who meets with the programming team to set priorities, answer questions, and make decisions about feature behavior. In contemporary agile teams, the term can represent any and all business stakeholders, product team members, end users, and anyone who helps guide development and accept delivered stories.

Endgame: The endgame is the time before release when the delivery team applies the finishing touches to the product. Not a bug-fix or "hardening" period, it is an opportunity to work with groups outside of development to help move the software into production. Examples of endgame activities include additional testing of database migrations and installation.

Iteration: Time box used for planning, with the intent that there is a "potentially shippable product" by the end. The Scrum term for this is "sprint." Planning in two-week timeframes is a common practice today, even in teams doing continuous delivery and deploying to production more often.

Kanban: A planning approach derived from Lean manufacturing in which teams work in a flow-based manner. They use work-in-progress (WIP) limits, pulling new stories in that are "ready," to fill a newly empty WIP slot. The team plans, as needed, a few new stories at a time.

Learning Release: The first release(s) delivered to a customer in order to get feedback to learn and adjust (https://medium.com/@Ardita_K/the-learning-release-70374d2450b3).

Mind map: A visual diagram used as a brainstorming tool, especially when multiple people are collaborating at once. It starts with a concept, idea, or topic in the root node, with ideas connected to the root node and to each other as they are generated. Mind maps can work well for test planning and other activities.

Pairing (pair programming, pair testing): Two people working side by side at the same workstation, preferably with two mirrored monitors, two keyboards, and two mice, to write production or test code or do other testing activities. Having two people, each with a different perspective and skill set, helps catch problems immediately and reach better solutions. In strong-style pairing, one person, the navigator, is free to observe and suggest ideas, while the other person acts as the "driver"; the driver/navigator roles switch frequently.

State Diagram: A visual technique used to give an abstract description of the **behavior**[46] of a **system**[47] in response to various events. This behavior is analyzed and represented as a series of events that can occur in one or more possible states.

Test-driven Development (TDD): In test-driven development, the programmer writes and automates a small unit test, which initially fails, before writing the minimum amount of code that will make the test pass. The code is refactored as needed to meet acceptable standards. The production code is made to work one test at

[46] https://en.wikipedia.org/wiki/Behavior
[47] https://en.wikipedia.org/wiki/System

a time. TDD, also known as test-driven design, is more of a code design practice than a testing activity, and helps build robust, easily maintainable code.

Bronnen voor verder leren

Algemeen

Agile Testing: A Practical Guide for Testers and Agile Teams, En *More Agile Testing: Learning Journeys for the Whole Team*, Lisa Crispin en Janet Gregory, https://agiletester.ca[48]

Bibliografie van *More Agile Testing*, gedigitaliseerd door Kristine Corbus

- https://testretreat.com/2018/01/28/more-agile-testing-introduction/
- https://testretreat.com/2018/01/29/more-agile-testing-learning-better-testing/
- https://testretreat.com/2018/01/29/more-agile-testing-planning/
- https://testretreat.com/2018/01/30/more-agile-testing-business-value
- https://testretreat.com/2018/02/15/more-agile-testing-test-automation/

Gemeenschappelijk begrip - Samenwerking

Discovery: Explore behavior using examples, Seb Rose en Gáspár Nagy, 2017, http://bddbooks.com/

User Story Mapping: Building Better Products Using Agile Software Design, Jeff Patton, O'Reilly Media, 2014.

Impact Mapping, Gojko Adzic, http://impactmapping.org

"Experiment with Example Mapping," https://lisacrispin.com/2016/06/02/experiment-example-mapping/

"Introduction to Example Mapping," Matt Wynne, https://cucumber.io/blog/example-mapping-introduction/

"Three Amigos Strategy," George Dinwiddie, https://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories

"Our team's first mobbing session," Lisi Hocke, https://www.lisihocke.com/2017/04/our-teams-first-mobbing-session.html

"The Driver-Navigator in Strong-Style Pairing," Maaret Pyhäjärvi, https://medium.com/@maaret.pyhajarvi/the-driver-navigator-in-strong-style-pairing-2df0ecb4f657

Strong-Style Pair Programming and *Mob Programming Guidebook*, Maaret Pyhäjärvi, https://leanpub.com/u/maaretp

Onderzoekend testen

Explore It: Reduce Risk and Increase Confidence with Exploratory Testing, Elisabeth Hendrickson, 2013, https://pragprog.com/book/ehxta/explore-it

Exploratory Testing, Maaret Pyhäjärvi, https://leanpub.com/exploratorytesting

[48] https://agiletester.ca/

DevOps, Monitoring, Waarneembaarheid

A Practical Guide to Testing in DevOps, Katrina Clokie, https://leanpub.com/testingindevops

"Testing in production the safe way," Cindy Sridharan, https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

"Monitoring and observability," Cindy Sridharan, https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c

"Charity Majors on Observability and Understanding the Operational Ramifications of a System," InfoQ interview met Charity Majors, https://www.infoq.com/articles/charity-majors-observability-failure

Google Site Reliability Engineering, https://landing.google.com/sre/

"What is Chaos Engineering[49]?" *Joe Colontonio (inclusief een link naar de podcast met Tammy Butow),* https://www.joecolantonio.com/chaos-engineering/

"Tracing vs Logging vs Monitoring: What's the Difference?" Chrissy Kidd, https://www.bmc.com/blogs/monitoring-logging-tracing/

Test Automatisering

"Keep your automated tests simple and avoid anti-patterns," Lisa Crispin, https://www.mabl.com/blog/keep-your-automated-testing-simple

"Test automation: Five questions leading to five heuristics," Joep Shuurkes, https://testingcurve.wordpress.com/2015/03/24/test-automation-five-questions-leading-to-five-heuristics/

"Powerful test automation practices," deel 1 en 2, Lisa Crispin en Steve Vance, https://www.mabl.com/blog/powerful-test-automation-practices-pt-1, https://www.mabl.com/blog/powerful-test-automation-practices-pt-2

"Test Suite Design," Ashley Hunsberger, https://github.com/ahunsberger/testSuiteDesign

Accelerate: The Science of Lean and DevOps, Nicole Forsgren, *en anderen,* https://itrevolution.com/book/accelerate/

"Analyzing automated test failures," Lisa Crispin, https://www.mabl.com/blog/lisa-webinar-analyzing-automated-ui-test-failures

"The Testing Iceberg," Seb Rose, http://claysnow.co.uk/the-testing-iceberg/

"Lower level automation and testing? Be more precise! The automation triangle revisited again!" Toyer Mamoojee, https://toyerm.wordpress.com/2018/10/16/lower-level-automation-and-testing-be-more-precise-the-automation-triangle-revisited-again

The Team Guide to Software Testability, Ash Winter en Rob Meaney, https://leanpub.com/softwaretestability

[49] https://www.joecolantonio.com/chaos-engineering/

Resources for Further Learning

General

Agile Testing: A Practical Guide for Testers and Agile Teams, and *More Agile Testing: Learning Journeys for the Whole Team*, Lisa Crispin and Janet Gregory, https://agiletester.ca[50]

Bibliography from *More Agile Testing,* digitized by Kristine Corbus

-https://testretreat.com/2018/01/28/more-agile-testing-introduction/

-https://testretreat.com/2018/01/29/more-agile-testing-learning-better-testing/

-https://testretreat.com/2018/01/29/more-agile-testing-planning/

-https://testretreat.com/2018/01/30/more-agile-testing-business-value

-https://testretreat.com/2018/02/15/more-agile-testing-test-automation/

Getting Shared Understanding - Collaboration

Discovery: Explore behavior using examples, Seb Rose and Gáspár Nagy, 2017, http://bddbooks.com/

User Story Mapping: Building Better Products Using Agile Software Design, Jeff Patton, O'Reilly Media, 2014.

Impact Mapping, Gojko Adzic, http://impactmapping.org

"Experiment with Example Mapping," https://lisacrispin.com/2016/06/02/experiment-example-mapping/

"Introduction to Example Mapping," Matt Wynne, https://cucumber.io/blog/example-mapping-introduction/

"Three Amigos Strategy," George Dinwiddie, https://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories

"Our team's first mobbing session," Lisi Hocke, https://www.lisihocke.com/2017/04/our-teams-first-mobbing-session.html

"The Driver-Navigator in Strong-Style Pairing," Maaret Pyhäjärvi, https://medium.com/@maaret.pyhajarvi/the-driver-navigator-in-strong-style-pairing-2df0ecb4f657

Strong-Style Pair Programming and *Mob Programming Guidebook*, Maaret Pyhäjärvi, https://leanpub.com/u/maaretp

Exploratory Testing

Explore It: Reduce Risk and Increase Confidence with Exploratory Testing, Elisabeth Hendrickson, 2013, https://pragprog.com/book/ehxta/explore-it

[50] https://agiletester.ca/

Exploratory Testing, Maaret Pyhäjärvi, https://leanpub.com/exploratorytesting

DevOps, Monitoring, Observability

A Practical Guide to Testing in DevOps, Katrina Clokie, https://leanpub.com/testingindevops

"Testing in production the safe way," Cindy Sridharan, https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

"Monitoring and observability," Cindy Sridharan, https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c

"Charity Majors on Observability and Understanding the Operational Ramifications of a System," InfoQ interview with Charity Majors, https://www.infoq.com/articles/charity-majors-observability-failure

Google Site Reliability Engineering, https://landing.google.com/sre/

"What is Chaos Engineering[51]?" *Joe Colontonio (includes link to podcast with Tammy Butow),* https://www.joecolantonio.com/chaos-engineering/

"Tracing vs Logging vs Monitoring: What's the Difference?" Chrissy Kidd, https://www.bmc.com/blogs/monitoring-logging-tracing/

Test Automation

"Keep your automated tests simple and avoid anti-patterns," Lisa Crispin, https://www.mabl.com/blog/keep-your-automated-testing-simple

"Test automation: Five questions leading to five heuristics," Joep Shuurkes, https://testingcurve.wordpress.com/2015/03/24/test-automation-five-questions-leading-to-five-heuristics/

"Powerful test automation practices," parts 1 and 2, Lisa Crispin and Steve Vance, https://www.mabl.com/blog/powerful-test-automation-practices-pt-1, https://www.mabl.com/blog/powerful-test-automation-practices-pt-2

"Test Suite Design," Ashley Hunsberger, https://github.com/ahunsberger/testSuiteDesign

Accelerate: The Science of Lean and DevOps, Nicole Forsgren, *et al,* https://itrevolution.com/book/accelerate/

"Analyzing automated test failures," Lisa Crispin, https://www.mabl.com/blog/lisa-webinar-analyzing-automated-ui-test-failures

"The Testing Iceberg," Seb Rose, http://claysnow.co.uk/the-testing-iceberg/

"Lower level automation and testing? Be more precise! The automation triangle revisited again!" Toyer Mamoojee, https://toyerm.wordpress.com/2018/10/16/lower-level-automation-and-testing-be-more-precise-the-automation-triangle-revisited-again

The Team Guide to Software Testability, Ash Winter and Rob Meaney, https://leanpub.com/softwaretestability

[51] https://www.joecolantonio.com/chaos-engineering/

Over de auteurs

Jeanet Gregory is een agile testing coach en procesbegeleider bij DragonFire Inc. Haar vakgenoten hebben haar in 2015 uitgeroepen tot de 'Meest Invloedrijke Agile Testing Professional Persoon'. Janet is gespecialiseerd in tonen aan agile teams hoe testpraktijken nodig zijn om producten van goede kwaliteit te ontwikkelen. Ze geeft wereldwijd agile testing cursussen en brengt haar zomers door in de bergen buiten Calgary.

Lisa Crispin verspreidt al twee decennia lang agile plezier in de testwereld en testplezier in de agile wereld. Haar vakgenoten verkozen haar tot de 'Meest Invloedrijke Agile Testing Professional Persoon' in 2012. Haar huidige interesses zijn het helpen van teams om te slagen met continuous delivery en deployment; ze is dan ook een uitgesproken voorstander van testen en werkt bij mabl om toonaangevende praktijken op het gebied van testen in de softwaregemeenschap te verkennen. Lisa woont met haar man, ezels, katten en honden in het prachtige Vermont.

Janet en Lisa zijn auteurs van: *Agile Testing Condensed: A Brief Introduction* (2019), *More Agile Testing: Learning Journeys for the Whole Team* (2014), *Agile Testing: A Practical Guide for Testers and Agile Teams* (2009), de Agile Testing Essentials LiveLessons video cursus, en de 3-daagse training "Agile Testing for the Whole Team" aangeboden door de Agile Testing Fellowship. Samen hebben zij de Agile Testing Fellowship opgericht om een gemeenschap te laten groeien voor alle beoefenaars die kwaliteit hoog in het vaandel dragen.

Janet Gregory: janetgregory.ca[52] Twitter: @janetgregoryca

Lisa Crispin: lisacrispin.com[53] Twitter: @lisacrispin

Agile Testing Fellowship: agiletestingfellow.com[54] agiletester.ca[55]

[52] https://janetgregory.ca/
[53] https://lisacrispin.com/
[54] https://agiletestingfellow.com/
[55] https://agiletester.ca/

About the Authors

Janet Gregory is an agile testing coach and process consultant with DragonFire Inc. Her peers voted her as the Most Influential Agile Testing Professional Person in 2015. Janet specializes in showing agile teams how testing practices are necessary to develop good quality products. She teaches agile testing courses worldwide and enjoys spending her summers in the mountains outside Calgary.

Lisa Crispin has been spreading agile joy to the testing world and testing joy to the agile world for two decades and her peers voted her as the Most Influential Agile Testing Professional Person in 2012. Her current interests include helping teamssucceed with continuous delivery and deployment and she is a testing advocate working at mabl to explore leading practices in testing in thesoftware community. Lisa lives with her husband, donkeys, cats and dogs in beautiful Vermont.

Janet and Lisa are authors of *Agile Testing Condensed: A Brief Introduction* (2019), *More Agile Testing: Learning Journeys for the Whole Team* (2014), *Agile Testing: A Practical Guide for Testers and Agile Teams* (2009), the LiveLessons Agile Testing Essentials video course, and "Agile Testing for the Whole Team" 3-day training course offered through the Agile Testing Fellowship. Together, they founded the fellowship to grow a community of practitioners who care about quality.

Janet Gregory: janetgregory.ca[56] Twitter: @janetgregoryca

Lisa Crispin: lisacrispin.com[57] Twitter: @lisacrispin

Agile Testing Fellowship: agiletestingfellow.com[58] agiletester.ca[59]

[56] https://janetgregory.ca/
[57] https://lisacrispin.com/
[58] https://agiletestingfellow.com/
[59] https://agiletester.ca/

Vertalers

Naam
Geike Hanoulle
Han Toan Lim
Harry Nieboer
Jolien Schoukens
Katrien Verheyden
Kim Lauwers
Peter Wynands
Sven Cipido
Yves Hanoulle