

Hoe innovatie
jouw IT-kwaliteit
beïnvloedt.

Trends in Testing

Artificial Intelligence

Agent skills
gebruiken
om Playwright-
tests te schrijven

p 20

Testautomatisering

Wat is de staat
van Robot
Framework?

p 38

Soft skills

Soft skills: de
strategische
spiergroep van de
Quality Engineer

p 46

Wij zijn Sofius. *Powerhouse* in softwaretesten en kwaliteitsverbetering. Een krachtige combinatie van toonaangevende labels. Gedreven door samenwerking, innovatie en hoogwaardige expertise.

Vooruitkijken in een vakgebied dat nooit stilstaat

De wereld van softwarekwaliteit staat nooit stil. Nieuwe technologieën, werkwijzen en inzichten volgen elkaar in hoog tempo op. Als testconsultancy organisatie hebben we de ambitie om vóórop te lopen in ons vakgebied. We signaleren, adopteren, innoveren en delen de nieuwste trends en ontwikkelingen. Dit doen we door het bezoeken van conferenties, ongelimiteerd te trainen, te presenteren op events, het schrijven van blogs en het organiseren van inhouse kennis-sessies. Dit alles met maar één doel: een significante bijdrage leveren aan de ontwikkeling binnen ons vakgebied én organisaties helpen om softwarekwaliteit naar een hoger niveau te brengen.

Om onze klanten daarin zo goed mogelijk te ondersteunen, volgen we ontwikkelingen binnen ICT en softwaretesten op de voet. We kijken daarbij vooral naar trends die daadwerkelijk impact hebben op de praktijk. Welke ontwikkelingen veranderen de manier waarop we testen? Waar liggen nieuwe kansen? En welke kennis en vaardigheden worden steeds belangrijker?

Jaarlijks onderzoekt Sofius de belangrijkste trends en ontwikkelingen binnen het testvak. De resultaten brengen we samen in dit trendrapport. We hopen organisaties in Nederland hiermee te inspireren en te ondersteunen bij hun uitdagingen van vandaag én morgen.

In dit trendrapport nemen we je mee in de belangrijkste ontwikkelingen van 2026. We laten zien welke trends aan terrein winnen, hoe deze zich ontwikkelen ten opzichte van voorgaande jaren en wat dit betekent voor organisaties en testprofessionals.

Inhoudsopgave

Top 10 Trends in Testing	06
Trend 1 - Artificial Intelligence	08
– AI in de praktijk: lessons learned uit een grote bancaire organisatie	10
– Wanneer AI gebruiken en wanneer niet?	14
– Een AI agent als softwareontwikkelaar: hoe waarborgt een tester de kwaliteit?	18
– Agent skills gebruiken om Playwright	20
De labels van Sofius	26
Trend 2 – Testautomatisering	28
– Hoe weet je of jouw team de juiste Test Automation tools en technieken kiest?	30
– Het testen van mobiele apps op mobiele devices met Appium	34
– Wat is de staat van Robot Framework	38
Trends in Testing 2025	42
Trend 3 – Soft skills	44
– Soft Skills: de strategische spiergroep van de Quality Engineer	46
– Have you lost your mind(set)?	48
– Niet alles wat kan, moet je beloven!	50
Onze onderzoeksopzet	52
Onze expertise	54

Collab.
Test.
Accelerate.



Top 10 Trends in Testing

1. Artificial Intelligence

2. Testautomatisering

3. Soft skills

4. Quality engineering

5. Continuous testing

6. DevOps

7. Contract testing

8. Risk-based testing

9. Testdata

10. Accessibility testing

Software testen in 2026: van slimmer testen naar grip op kwaliteit

Wie de trends van 2025 en 2026 naast elkaar legt, ziet een duidelijke verschuiving. Waar technologie vorig jaar vooral nieuwe mogelijkheden bood om sneller en slimmer te testen, draait het nu steeds meer om de vraag hoe we die technologie verantwoord en beheerst inzetten.

Dat zien we het sterkst bij **Artificial Intelligence (AI)**. In 2025 lag de nadruk vooral op AI als hulpmiddel voor de test engineer: testcases genereren, testscripts onderhouden, defecten voorspellen en regressietesten versnellen. In 2026 krijgt AI een veel bredere impact op het vakgebied. AI wordt niet alleen ingezet om te testen, maar is steeds vaker zélf onderwerp van kwaliteitsbewaking. Denk aan AI-agents die zelfstandig taken uitvoeren, AI-generated code, de betrouwbaarheid van LLM's, compliance en de noodzaak van menselijke controle. De centrale vraag verschuift daarmee van 'Wat kunnen we met AI?' naar 'Hoe zorgen we dat we AI kunnen vertrouwen?'.

Ook **testautomatisering** ontwikkelt zich verder. Automatisering blijft onverminderd belangrijk, maar staat steeds minder op zichzelf. Waar de discussie eerder vooral draaide om frameworks, tooling en onderhoud, raakt testautomatisering steeds sterker verweven met quality engineering, CI/CD, observability en AI-ondersteunde workflows. Automatisering wordt daarmee volwassen, maar tegelijkertijd ook complexer.

Onderliggend zien we nog een derde beweging: **kwaliteit wordt strategischer**. Organisaties experimenteren volop met generatieve AI en nieuwe vormen van automatisering, maar de stap naar grootschalige, beheersbare toepassing blijkt nog lastig. Governance, risicobeheersing, compliance en aantoonbare betrouwbaarheid worden daardoor belangrijker.

Voor testprofessionals betekent dit dat het vakgebied opnieuw verbreedt. Het gaat niet alleen om méér of sneller testen, maar steeds nadrukkelijker om het beheersen van risico's en het aantoonbaar maken van kwaliteit. Juist naarmate technologie een grotere rol speelt, **worden ook sociale vaardigheden belangrijker**. Kritisch denken, effectief communiceren, samenwerken en stakeholders meenemen in kwaliteitsvraagstukken zijn essentieel om technologie op de juiste manier in te zetten en de vertaalslag te maken van technische mogelijkheden naar waarde voor de organisatie.

De tester van 2026 helpt organisaties niet alleen fouten te vinden, maar vooral vertrouwen te creëren in een steeds complexer technologisch landschap.

We zien veel organisaties AI actief inzetten en beschikbaar stellen aan hun medewerkers. Ook zien we een aantal organisaties die AI juist actief blokkeren zodat niemand het kan gebruiken. Dit creëert een tweedeling. Aan de ene kant wordt AI ingezet om specificaties, code en testen te (helpen) schrijven. Aan de andere kant wordt doorgewerkt zoals dat in het verleden altijd gedaan is.

Artificial Intelligence

De beweging naar agentic oplossingen gaat door. Waar chatbots werken op basis van instructies, werken agents op basis van doelen. Deze agents kiezen zelf welke tools (MCP) en 'skills' ze nodig hebben om hun doel te bereiken. Om hun doel te kunnen bereiken hebben agents meer zelfstandigheid en rechten nodig. Veel organisaties durven dit nog niet of slechts beperkt aan.

Op conferenties verschuift het gesprek van "wat kan AI" naar "wat blijft mensenwerk". AI is goed in het uitvoeren van taken, maar niet goed in het leiden van processen of het onderhouden van persoonlijke relaties. Daarnaast brengt het ook significante nadelen in betrouwbaarheid, veiligheid, kwaliteit en zelfs het kennisniveau van de gebruikers. AI vindt steeds meer zijn plek binnen teams als uitvoerder van taken en hulp bij code schrijven, maar niet als vervanger van mensen.

De les? Voor een sterke test mindset blijft de rol van de tester onmisbaar. AI levert de meeste waarde wanneer je het bewust inzet als hulpmiddel. AI werkt het beste voor het uitvoeren van taken onder leiding van mensen. Zo gebruik je de kracht van AI zonder controle uit handen te geven. Testers denken, kiezen en zijn eigenaar. AI voert deze taken uit.

Dit vereist een andere set vaardigheden van testers. Uitvoerende vaardigheden worden minder belangrijk terwijl beoordelen, keuzes maken, en kennis vergaren belangrijker worden.



AI in de praktijk: lessons learned uit een grote bancaire organisatie

Wat gebeurt er als AI niet langer een experiment is, maar onderdeel wordt van het dagelijkse werk? Test automation engineer Lars de Bruijn ervaart het van dichtbij bij een grote bancaire organisatie, waar AI inmiddels breed wordt toegepast. Als enige tester in een team met negen developers ziet hij hoeveel snelheid en efficiëntie AI kan opleveren. Maar ook wat dat betekent voor vaardigheden, werkplezier en de rol van de tester. Een persoonlijke terugblik op de belangrijkste lessen uit de praktijk.



Lars de Bruijn



Van experiment naar dagelijkse praktijk

Binnen de bancaire organisatie waar Lars als Test Automation Engineer werkt, is AI inmiddels verweven met het dagelijkse development- en testwerk. Voor Lars is dat extra waardevol: als enige tester in een team met negen developers helpt AI hem het tempo bij te houden en te voorkomen dat testen een bottleneck wordt. In de praktijk zet hij AI op vijf manieren in.

1. Geautomatiseerde tests schrijven

Lars gebruikt AI-agents en chatfunctionaliteiten om geautomatiseerde testscripts sneller op te zetten en uit te werken. De testaanpak en belangrijkste scenario's bepaalt hij zelf; AI ondersteunt vooral bij de technische uitwerking in Playwright. Zo kan hij in korte tijd meer tests realiseren.

2. Proof of concepts ontwikkelen

Ook bij het onderzoeken van nieuwe mogelijkheden levert AI tijdswinst op. Nieuwe tools verkennen, onbekende code doorgronden en oplossingen uitproberen gaat aanzienlijk sneller. Soms staat er binnen één dag al een bruikbaar proof of concept.

3. Een testautomatiseringsstrategie opstellen

AI fungeert ook als sparringpartner. Bij het ontwikkelen van een nieuwe testautomatiseringsstrategie gebruikte Lars AI om bronnen te verzamelen, informatie te structureren en inzichten uit gesprekken te combineren. Mede daardoor stond er binnen ongeveer twee weken een complete strategie die inmiddels binnen de organisatie wordt toegepast.

4. Werken met AI-agents en skills

Steeds vaker werkt Lars met agents, skills en complete AI-workflows. Daarin kan hij specifieke testkennis vastleggen en beschikbaar maken voor anderen. Zo wordt zijn kennis overdraagbaar en kunnen developers met behulp van deze agents zelfstandig verder wanneer zijn opdracht eindigt.

5. Documentatie maken

Ook het schrijven van (technische en functionele) documentatie wordt versneld. AI analyseert bestaande code en maakt een eerste opzet van documentatie, waarna een collega controleert of de uitleg klopt en begrijpelijk is. Een taak die al snel een uur kostte, kan zo in enkele minuten worden uitgevoerd.

AI bankbreed organiseren

De toepassing van AI blijft niet beperkt tot individuele medewerkers en teams. Ook bankbreed wordt geïnvesteerd in kennisdeling, herbruikbare oplossingen en verantwoord gebruik.

Zo delen medewerkers binnen een interne AI-community tijdens wekelijkse sessies nieuwe ontwikkelingen, demo's en praktijkervaringen. Daarnaast is er een centrale repository met AI-agents en skills die teams kunnen hergebruiken. Ook wordt AI ingezet in geautomatiseerde pipelines voor controles die voorheen handmatig plaatsvonden.

Onder developers wordt AI eveneens steeds meer onderdeel van de workflow. Taken die eerder als omvangrijk werden ingeschat, blijken soms veel sneller uitvoerbaar. Dat heeft niet alleen gevolgen voor het werk zelf, maar ook voor de manier waarop teams plannen en capaciteit inschatten.

Technologie gaat sneller dan de organisatie

De mogelijkheden zijn groot, maar de praktijk kent ook hobbels. Zeker binnen een bancaire organisatie spelen compliance, regelgeving en interne processen een belangrijke rol. Niet alles wat technisch kan, mag ook.

Daarnaast gaat de technologische ontwikkeling soms sneller dan de cultuurverandering. Sommige medewerkers omarmen AI, terwijl anderen terughoudend zijn of zich zorgen maken over wat het voor hun werk betekent. Volgens Lars is het belangrijk om die zorgen serieus te nemen.

Kleine, zichtbare successen kunnen helpen. Een proof of concept dat laat zien dat een taak ineens in een fractie van de tijd kan worden uitgevoerd, maakt de waarde van AI tastbaar. Tegelijkertijd zijn duidelijke kaders nodig: welke tools mogen medewerkers gebruiken, welke gegevens mogen ze delen en hoe gaan ze verantwoord om met bijvoorbeeld kosten en tokenverbruik? Goed AI-beleid beperkt zo niet alleen risico's, maar geeft medewerkers ook ruimte om veilig te experimenteren.

Wat doet AI met het vak van tester?

Misschien wel de interessantste verandering ziet Lars in zijn eigen werk. Waar hij voorheen veel code zelf schreef, laat hij dat inmiddels

grotendeels aan AI over. Hij weet wat er moet gebeuren, kan AI gericht aansturen en de gegenereerde code beoordelen. Maar zelf vanaf nul dezelfde code schrijven gaat na intensief AI-gebruik minder vanzelfsprekend dan voorheen.

Daar staat tegenover dat andere vaardigheden belangrijker worden. Het grotere geheel overzien, verbanden leggen, de juiste context meegeven en kritisch beoordelen of een oplossing daadwerkelijk klopt: de nadruk verschuift van zelf uitvoeren naar regisseren en beoordelen. Daarvoor moet je als tester juist een stevige basis-kennis van testen en techniek hebben. Alleen als je begrijpt hoe een goede oplossing eruitziet, kun je beoordelen of AI die ook daadwerkelijk levert. Daarnaast worden vaardigheden als kritisch denken, goede vragen stellen, context meegeven en risico's herkennen steeds belangrijker.

Die ontwikkeling heeft ook gevolgen voor de samenstelling van teams. Aan het begin van zijn opdracht was het voor Lars nauwelijks haalbaar om als enige tester negen developers bij te houden. Dankzij AI is dat werk veel beter behapbaar geworden. Tegelijkertijd leren developers AI steeds vaker zelf in te zetten voor testwerkzaamheden.

Dat betekent niet dat de tester verdwijnt, maar de rol verandert wel. Het wordt belangrijker om kwaliteit over de volle breedte te overzien, developers te helpen goede testkeuzes te maken en kritisch te beoordelen waar AI wel en niet verantwoord kan worden ingezet. Naast technische kennis worden daarom ook adviseren, communiceren, kennis overdragen en samenwerken steeds belangrijker.

De keerzijde van efficiëntie

Meer snelheid en productiviteit klinken vooral positief. Toch ziet Lars ook een minder voor de hand liggend effect: AI kan een deel van de uitdaging uit het werk halen. Voor professionals die plezier halen uit het zelf oplossen van complexe technische vraagstukken, kan het wennen zijn wanneer AI dat werk grotendeels overneemt. Anderen vinden het juist prettig dat zij minder tijd kwijt zijn aan de uitvoering en zich meer op het denkwerk kunnen richten. Voor managers ontstaat daarmee een nieuw vraagstuk. Als AI steeds meer uitvoerend werk overneemt, hoe zorg je er dan voor dat medewerkers voldoende uitdaging, ontwikkeling en voldoening blijven ervaren?

De impact van AI moet daarom niet alleen worden gemeten in tijdswinst en productiviteit, maar ook in wat het met mensen en hun werk doet.

Verder kijken dan de technologie

De ervaring bij deze bancaire organisatie laat zien hoe snel AI de dagelijkse praktijk kan veranderen. Eén tester kan met behulp van AI een team van negen developers ondersteunen, werkzaamheden die voorheen uren of dagen kostten worden veel sneller uitgevoerd en kennis kan via agents en skills beschikbaar worden gemaakt voor hele teams.

Maar de belangrijkste verandering zit misschien niet in wat AI allemaal kan. AI verandert ook welke vaardigheden professionals nodig hebben, hoe teams worden samengesteld en waar mensen hun uitdaging uit halen. Daarmee wordt de belangrijkste vraag voor organisaties steeds minder óf ze AI gaan gebruiken. Veel interessanter is hoe ze ervoor zorgen dat technologie, organisatie én mensen samen blijven ontwikkelen.

Drie lessen uit de praktijk

Na intensief werken met AI zijn dit de drie belangrijkste lessen die Lars wil meegeven:

1. Geef context en blijf zelf nadenken

AI beschikt over enorm veel kennis, maar kent niet automatisch jouw organisatie, situatie en ervaring. De kwaliteit van de output staat daarom voor een groot deel met de context die je meegeeft. Wees specifiek, gebruik je eigen vakkennis en blijf de uitkomsten kritisch beoordelen. Besteed het werk uit, maar niet je eigen oordeel.

2. Toon de waarde, niet de technologie

Wil je anderen overtuigen van de waarde van AI? Maak dan vooral concreet wat het oplevert. Niet hoe geavanceerd de technologie is, maar hoeveel tijd, capaciteit of kosten je ermee bespaart. Een taak terugbrengen van twee uur handmatig werk naar tien minuten spreekt vaak meer tot de verbeelding dan een technische uitleg over de oplossing.

3. Kijk verder dan efficiëntie alleen

Meer snelheid en productiviteit zijn waardevol, maar AI heeft ook invloed op hoe mensen hun werk ervaren. Als AI juist de werkzaamheden overneemt waar iemand uitdaging, plezier of ontwikkeling uit haalt, kan efficiëntiewinst ook een keerzijde hebben. Kijk daarom niet alleen naar wat AI oplevert voor de organisatie, maar blijf in gesprek met medewerkers over wat het met hun vak, ontwikkeling en werkplezier doet. De beste inzet van AI maakt niet alleen het werk efficiënter, maar zorgt er ook voor dat mensen zich kunnen blijven ontwikkelen.



Sander van Beek



Jochem Schenk

Wanneer AI gebruiken en wanneer niet?

“Kunnen we hier AI voor gebruiken?” is de verkeerde vraag. De vraag die je jezelf zou moeten stellen is: “Kan AI me hierbij helpen?”

AI is voor een testconsultant een hulpmiddel. Een veelzijdig hulpmiddel. Maar hoe bepaal je of AI een hulpmiddel is dat je ook echt beter maakt in je werk?

In dit artikel introduceren we een eenvoudig framework om te beoordelen of het gebruik van AI een goed idee is. Hiervoor gebruiken we vijf metrics: kwaliteit, inspanning, kennis, feedbackloop en risico. Het doel is niet om het gebruik van AI te maximaliseren, maar om de gebruikers ervan beter te maken in hun werk.

De metrics

1. Kwaliteit

Als je AI gebruikt voor ‘x’, gaat de kwaliteit van je product/proces omhoog, omlaag, of blijft het gelijk?

Denk hierbij aan correctheid, onderhoudbaarheid, duidelijkheid, documentatie, en nog veel meer.

2. Effort

Als je AI gebruikt voor ‘x’, gaat de menselijke inspanning omhoog, omlaag, of blijft het gelijk?

Dit is al het werk dat een persoon moet verrichten. Inclusief het schrijven van prompts, corrigeren en het implementeren van de resultaten van de AI in je product/proces.

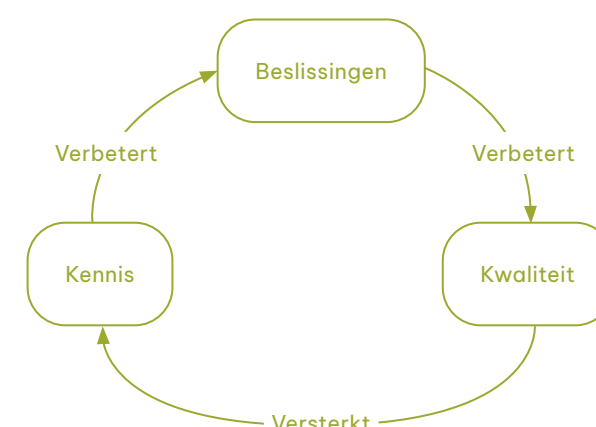
3. Kennis

Als je AI gebruikt voor ‘x’, gaat de kennis van individuele teamleden en het team als geheel op de lange termijn omhoog, omlaag, of blijft het gelijk?

Dit is de belangrijkste metric. Het is afhankelijk van waar je AI gebruikt in je proces. Vroeger werd je een goede engineer door je werk te doen en onderweg te leren. AI kan dit leerproces ondersteunen of ondermijnen. Hetzelfde geldt voor teams. Teams komen niet in de problemen omdat ze te langzaam zijn. Ze komen in de problemen omdat ze hun eigen systemen niet begrijpen.

4. Feedbackloop

Als je AI gebruikt voor ‘x’, creëer je een positieve of negatieve feedbackloop?



Over het algemeen leidt meer kennis tot betere beslissingen, wat leidt tot hogere kwaliteit, wat de kennis versterkt. Of het tegenovergestelde kan gebeuren: minder kennis leidt tot slechtere beslissingen, wat leidt tot slechtere kwaliteit, wat de kennis verder uitholt.

5. Risico

Als je AI gebruikt voor ‘x’, wat is dan het risico in productie: laag, middel of hoog?

Dit soort metrics kunnen erg abstract aanvoelen totdat je ze begint toe te passen. Laten we daarom kijken naar enkele voorbeeldsituaties waarin AI op verschillende manieren wordt gebruikt.

Voorbeeld 1: Een junior developer die AI gebruikt om functionaliteit te bouwen

- **Scenario:** Een junior developer gebruikt AI om productiecode te genereren voor nieuwe functionaliteiten.
- **Kwaliteit:** Over het algemeen dalend. De code werkt meestal, maar subtiele bugs, ontbrekende randgevallen en slechte ontwerpkeuzes sluipen er doorheen. Juniors hebben vaak niet de ervaring om deze problemen betrouwbaar te herkennen.
- **Inspanning:** Minder inspanning op de korte termijn. Meer inspanning op de lange termijn, omdat code moeilijker te onderhouden is als je het niet snapt.
- **Kennis:** Dalend. De junior leert hoe hij prompts moet schrijven, maar niet om een engineer te zijn. Door hun werk uit te besteden aan de AI, missen ze de leermogelijkheden die nodig zijn om te groeien.
- **Feedbackloop:** Negatief. Minder kennis leidt tot meer afhankelijkheid van AI, wat leidt tot nog minder begrip. Met de tijd neemt de AI-afhankelijkheid toe, daalt de codekwaliteit en stopt het leren.
- **Risico:** AI geschreven code komt in productie terecht, wat resulteert in meer (terugkerende) incidenten op alle risiconiveaus.
- **Conclusie:** Een kortetermijnwinst in productiviteit die stiltes langer termijnproblemen creëert op het gebied van kwaliteit, kennis en persoonlijke groei.

Voorbeeld 2: Een junior developer die AI gebruikt als leermiddel

- **Scenario:** Een junior developer werkt samen met AI om code te begrijpen, aanpakken te verkennen en feedback te krijgen. Een senior developer is beschikbaar voor verdere begeleiding en coaching.
- **Kwaliteit:** Stijgend over tijd. De junior bouwt kennis op door actief met AI-antwoorden aan de slag te gaan in plaats van ze blindelings toe te passen. Eenvoudige fouten worden eerder herkend en programmeervaardigheden groeien.
- **Inspanning:** Dalend voor zowel de junior als de senior. AI neemt een gedeelte van de begeleiding voor zijn rekening, zoals het uitleggen van code en het beoordelen van aanpakken. Dit geeft seniors de ruimte voor coaching en het aanleren van concepten die AI niet goed snapt.
- **Kennis:** Stijgend. De junior leert beter programmeren, uitkomsten beoordelen en leert sneller.
- **Feedbackloop:** Positief. Groeiende kennis leidt tot betere implementaties, wat leidt tot een beter product.
- **Risico:** Laag tot gemiddeld, en afnemend over tijd. Reviews door AI vindt fouten vroeg en de senior vindt de rest.
- **Conclusie:** Als je AI op deze manier gebruikt versnelt het de groei, in plaats van dat je deze groei uitbesteedt. De junior ontwikkelt zich sneller, de senior kan zich richten op coaching, en het team bouwt duurzame vaardigheden op in plaats van een afhankelijkheid van AI.

Voorbeeld 3: Een senior developer die AI gebruikt voor leren

- **Scenario:** Een senior developer gebruikt AI om opties te verkennen, codewijzigingen te conceptualiseren en repetitieve code te genereren.
- **Kwaliteit:** Stijgend. Seniors hebben de kennis om slechte ideeën te negeren en goede ideeën te verfijnen. AI vergroot de oplossingsmogelijkheden zonder het beoordelingsvermogen te vervangen.
- **Inspanning:** Dalend. Het uitproberen van ideeën, vergelijken van oplossingen en eerste versies wordt makkelijker. Het echte denkwerk blijft, maar met minder weerstand.

- **Kennis:** Stijgend of neutraal. Sterke bestaande mentale modellen zorgen ervoor dat AI-output wordt beoordeeld maar niet blindelings wordt vertrouwd. Dit kan het kennis versterken.
- **Feedbackloop:** Positief. Betere ontwerpen leiden tot betere systemen, die makkelijker te doorgronden zijn, wat zowel mensen als toekomstig AI-gebruik effectiever maakt.
- **Risico:** Laag tot gemiddeld. Seniors werken aan impactvolle onderdelen, maar voegen ook guardrails toe: tests, reviews en incrementele aanpassingen.
- **Conclusie:** Dit is een goed gebruik van AI. Het versterkt expertise in plaats van het gebrek ervan te maskeren.

Conclusie

Afhankelijk van hoe je AI gebruikt, kan het verschillende uitkomsten opleveren. Wanneer AI alleen wordt ingezet om te versnellen, mis je vaak kennis en begrip van wat je product doet. Dit betekent dat je steeds afhankelijker wordt van AI en uiteindelijk niet meer in staat bent om het te corrigeren wanneer het onvermijdelijk een cruciale fout maakt.

In plaats daarvan zou AI moeten worden gebruikt om je vaardigheden te versterken, je kennisbasis uit te breiden en het menselijk begrip van wat we ontwikkelen te vergroten. Op die manier haal je het beste uit beide werelden.

Veel mensen hebben het over “mens in de loop” als een manier om AI in toom te houden. Dit is niet genoeg. Een niet betrokken mens die AI-output blind goedkeurt, is trage automatisering. Wat je wilt is “brein in de loop”: iemand die actief redeneert, vragen stelt en kennis opbouwt.

Collab.
Test.
Accelerate.





Een AI agent als softwareontwikkelaar: hoe waarborgt een tester de kwaliteit?

Stel je voor: samen met je team werk je aan het ontwikkelen van software en iedereen mag gebruik maken van Artificial Intelligence (AI). Eén van de softwareontwikkelaars besluit een AI-agent te ontwikkelen die zelf initiatief neemt en autonoom optreedt als softwareontwikkelaar. Jouw rol binnen het team is softwaretester. Hoe waarborg je de kwaliteit van de opgeleverde software door de AI-agent? Tegen welke uitdagingen loop je aan en hoe ga je hiermee om? In deze blog ga ik in op wat de opkomst van AI binnen ontwikkelteams betekent voor kwaliteitswaarborging bij het testen van software.

Waar moet je op letten bij het controleren van de kwaliteit van software die met AI is ontwikkeld?

Allereerst is het belangrijk om te begrijpen welk type output het systeem genereert. Bij deterministische output levert dezelfde invoer altijd hetzelfde resultaat op. Bij non-deterministische output kan dezelfde invoer verschillende resultaten opleveren. Beide vragen om een andere manier van testen.

Vooraf bij non-deterministische output is kwaliteit lastiger te beoordelen. Op een open vraag kunnen bijvoorbeeld meerdere antwoorden correct zijn, ook als de formulering verschilt. Kijk daarom niet alleen naar de exacte output, maar vooral naar de relatie tussen input en output. Documenteer welke variatie optreedt en beoordeel vervolgens of verschillende antwoorden inhoudelijk nog steeds correct zijn.

Welke kwaliteitseisen moeten er gesteld worden aan software die met AI is ontwikkeld?

Bij het controleren van de kwaliteit kijk je niet alleen naar het eindresultaat, maar ook naar tussenliggende stappen en herhaling van taken. Bij deterministische output controleer je of

dezelfde invoer steeds hetzelfde resultaat oplevert. Bij non-deterministische output beoordeel je juist of variërende uitkomsten binnen de gestelde grenzen en voorwaarden blijven.

Wanneer fouten worden gedetecteerd, moet de AI-agent de actie stoppen en waar mogelijk corrigeren. Formele regels helpen om fouten te herkennen en de output aan de gestelde kwaliteitseisen te laten voldoen. Belangrijke risico's zijn:

- **Verkeerde redenering:** de AI komt tot een inhoudelijk onjuiste conclusie. Bijvoorbeeld $1+1+1=1$ in plaats van $1+1+1=3$.
- **Bias:** de uitkomst bevat vooroordelen of systematische vertekeningen, bijvoorbeeld doordat bepaalde groepen worden uitgesloten.
- **Prompt-injectie:** een gebruiker geeft instructies waardoor de AI afwijkt van de oorspronkelijke systeem-instructies.
- **Jailbreaking:** veiligheidsbeperkingen worden bewust omzeild. Dit is een vorm van prompt-injectie.
- **Privacylekken:** persoonsgegevens of andere gevoelige data worden onvoldoende beschermd en kunnen via de AI toegankelijk worden.

Hoe test je of de output van AI aan de kwaliteitseisen voldoet?

Toets of de AI voldoet aan vooraf opgestelde formele regels voor verantwoord gebruik. Denk aan kaders voor transparantie, veiligheid, privacy, aansprakelijkheid en risicobeheer.

Voorbeelden van formele regels zijn:

- **Prestatie-kwantificatie:** "Het AI-model laat bij elk antwoord zien hoe groot de kans is op een fout. Het rapporteert dit aan de hand van accuratesse, precisie, recall, F1-score en de verhouding false positives en false negatives. Dit ondersteunt de kaders voor transparantie door duidelijke prestatie-metingen te bieden en veiligheid door foutmarges te rapporteren."
- **Hallucinatie preventie:** "Het AI-model moet informatie vergelijken met één of meerdere verifieerbare bronnen om kans op hallucinaties of niet onderbouwde conclusies te minimaliseren en inconsistentie te signaleren. Dit waarborgt veiligheid door betrouwbare informatie te leveren en privacy

door het vermijden van het verspreiden van onjuiste data waaronder persoonsgegevens."

- **Regelconsistentie:** "Het AI-model mag nooit X zeggen wanneer voorwaarde Y geldt. Dit ondersteunt risicobeheer door gevaarlijke uitkomsten te voorkomen en aansprakelijkheid door consistentie te waarborgen."
- **Formatvalidatie:** "Het AI-model moet JSON-schema checks en API-contractvalidatie toepassen om het format te valideren en inconsistenties daarop te signaleren. Dit ondersteunt privacy door data-integriteit te waarborgen en veiligheid door correcte formaten te gebruiken."
- **Beperkingsschendingen:** "Het AI-model moet zich houden aan lengte, vocabulaire, en verboden tokens. Dit ondersteunt risicobeheer door onbeheerste uitvoer te voorkomen en veiligheid door het voorkomen van ongeautoriseerd delen van gevoelige informatie."
- **Contra feitelijke tests:** "Het AI-model moet gelijke beslissingen geven bij minimale wijziging in formulering. Dit waarborgt aansprakelijkheid door eerlijke resultaten en transparantie door vergelijkbare uitkomsten."
- **Veiligheidsfilters:** "Het AI-model mag nooit verboden concepten gebruiken. Dit ondersteunt privacy door gevoelige inhoud te blokkeren en veiligheid door ethische richtlijnen te respecteren."

Om deze regels te toetsen zijn verschillende tools beschikbaar, zoals Promptfoo, DeepTeam, PyRIT en Garak. Hiermee kun je prompts testen en beoordelen welke variant het gewenste antwoord, gedrag of format oplevert.

Daarnaast kunnen deze tools de output evalueren op bijvoorbeeld nauwkeurigheid, volledigheid en relevantie. Ze worden ook ingezet voor Red Teaming: het bewust proberen te breken, misleiden of omzeilen van AI-software om kwetsbaarheden te ontdekken voordat deze in productie ontstaan. Denk aan verkeerde redenering, bias, prompt-injectie, jailbreaking en privacylekken.

Bij Red Teaming probeer je bewust software te breken, misleiden of omzeilen om kwetsbaarheden te ontdekken vóórdat ze in productie voor problemen zorgen.

Agent skills gebruiken om Playwright-tests te schrijven

Hoe agent skills en instructies mij helpen om E2E-tests te genereren zonder zelf code te schrijven.



Tim Lolkema

Agentic engineering heeft het afgelopen jaar een enorme vlucht genomen. Claude Code werd in mei 2025 publiek beschikbaar en sindsdien zagen we onder andere de introductie van skills, sub-agents en verschillende alternatieven, zoals OpenCode en Codex. Deze ontwikkeling is niet meer weg te denken en verandert de manier waarop we code schrijven.

Werken met AI-agents is een vaardigheid op zich. Met dit artikel hoop ik je te helpen die vaardigheid verder te ontwikkelen.

Zonder duidelijke instructies leidt 'vibe coding' al snel tot resultaten die er goed uitzien, maar inhoudelijk niet kloppen. Kijk bijvoorbeeld naar dit voorbeeld:

```
// What AI generated:
page.getByRole("button", { name:
t("checkout.payment_button")
}).click();

// What actually existed:
page.getByRole("link", { name:
t("checkout.complete_purchase")
}).click();
```

Op het eerste gezicht zag de code er prima uit: correcte Playwright-syntax, een translation function en een toegankelijke rol. Maar AI baseerde zich op hoe een checkoutknop er waarschijnlijk uit zou zien en welke translation key logisch leek, zonder eerst de daadwerkelijke implementatie te controleren.

Gelukkig kunnen we daar iets aan doen.

Door goede agent-instructies te combineren met een gedetailleerde skill voor het schrijven van E2E-tests, hooks en self-validation, worden de resultaten een stuk beter.

Wat zijn agent skills?

Een skill is een herbruikbare set instructies die een agent leert hoe hij een bepaalde taak moet uitvoeren. Deze wordt opgeslagen in een map met daarin een SKILL.md-bestand met

metadata, zoals de naam en beschrijving, en de instructies zelf.

```
.claude/
├── skills/
│   └── write-e2e-test/
│       └── SKILL.md
```

Skills kunnen door de gebruiker worden gestart, vergelijkbaar met 'slash commands', maar een agent kan ook zelf besluiten een skill te gebruiken. Bij het opstarten ziet de agent alleen de naam en beschrijving van beschikbare skills. Pas wanneer een skill nodig is, worden de volledige instructies aan de context toegevoegd. Dat maakt het efficiënt in het gebruik van tokens.



Je kunt agent skills zien als een stapsgewijs proces dat je een AI-agent meegeeft om een specifieke taak uit te voeren.

In mijn skill beschrijf ik een stapsgewijs proces van planning tot implementatie, met als eindresultaat een werkende E2E-test.

"Agent Skills zijn een lichtgewicht, open format om de mogelijkheden van AI-agent uit te breiden met gespecialiseerde kennis en workflows."

Combineer skills met AGENTS.md-files

Skills zijn bedoeld voor specifieke taken. Maar een agent heeft daarnaast algemene kennis over het project nodig. Denk aan de projectstructuur, goede en slechte werkwijzen en instructies voor het uitvoeren van de applicatie en tests.

Daarvoor gebruiken we AGENTS.md-files. Werk je met Claude Code, dan kun je hiervoor CLAUDE.md-files gebruiken.

Deze bestanden kunnen op verschillende niveaus in het project worden geplaatst. Algemene instructies staan bijvoorbeeld in het AGENTS.md-file in de root, terwijl specifieke packages hun eigen instructies krijgen.



Structureer je instructies in verschillende lagen met geneste AGENTS.md-files.

Tips voor effectieve AGENTS.md-files

Beschouw deze bestanden niet als statisch. Blijf ze aanpassen op basis van het gedrag van je agent. Zet er ook niet zomaar alles in. Wanneer de context te groot wordt, kan een agent moeite krijgen om te bepalen wat echt belangrijk is.

Het doel is om je agent voldoende context te geven om zelfstandig te werken, zonder hem te overspoelen met instructies.

Zie AGENTS.md als een README voor agents: een vaste plek waar je AI coding agents de context en instructies geeft die ze nodig hebben om aan je project te werken.

Hoe werkt de skill?

1. Bepaal de testrequirements

Eerst moet de agent duidelijk krijgen wat er getest moet worden. Hij stelt vragen over de scope, stelt relevante testcases voor en vraagt waar het succes van de test moet worden gevalideerd.

Moet de test bijvoorbeeld alleen een succesvolle aankoop controleren, of ook foutscenario's? Moet worden gekeken naar wat de gebruiker ziet, naar de status in de backend of naar allebei?

Door dit vooraf te bespreken, ontstaat een beter plan voordat er ook maar één regel code wordt geschreven.

2. Onderzoek de daadwerkelijke implementatie

Vervolgens moet de agent de applicatie onderzoeken in plaats van aannames te doen.

Voor mijn project betekent dit dat de agent onder andere de daadwerkelijke translation files, frontendcomponenten en relevante GraphQL-calls bekijkt. Zo beschikt hij over de echte toegankelijke namen, rollen, labels en API-gedragingen die nodig zijn voor een robuuste Playwright-test.

Ontbreekt in een component bijvoorbeeld een toegankelijk label of een goede rol? Pas dan liever het component aan dan dat je het probleem omzeilt met een kwetsbare selector. Betere toegankelijkheid levert zowel gebruikers als tests een betere interface op.

3. Maak eerst een plan

Voordat de agent begint met implementeren, presenteert hij de voorgestelde tests, wat iedere test controleert en welke aanpak hij wil volgen.

Ik kan dat plan vervolgens nog aanpassen voordat er code wordt geschreven. Op dat moment zijn wijzigingen immers nog eenvoudig door te voeren.



Dan nu het interessante deel: we gaan de skill daadwerkelijk gebruiken.



4. Implementeer volgens de projectafspraken

Na goedkeuring schrijft de agent de test volgens de bestaande werkwijze binnen de repository. In mijn geval betekent dat onder andere:

- Waar mogelijk testdata en teststatus via API-calls opzetten.
- Toegankelijke locators gebruiken, met voorkeur voor role en label.
- Wachten op relevante netwerkrequests.
- Tests onafhankelijk en parallel uitvoerbaar houden.
- De translation utilities van het project gebruiken in plaats van hardcoded UI-tekst.

Niet ieder project heeft precies deze regels nodig. Het belangrijkste is dat je de agent de afspraken meegeeft die voor jouw codebase relevant zijn.

5. Valideer voordat het werk wordt opgeleverd

Tot slot voert de agent de nieuwe test meerdere keren afzonderlijk uit. Zo worden verkeerde locators, ontbrekende waits, translation mismatches en timingproblemen ontdekt voordat de test als afgerond wordt gepresenteerd.

```
playwright test -g "Should  
[test title]" --repeat-each=5  
--reporter=line
```

Met de -g-optie voeren we alleen de specifieke test uit. Door de line reporter te gebruiken, krijgt de agent de resultaten direct terug. Een test die er alleen correct uitziet, is niet voldoende. Hij moet ook daadwerkelijk betrouwbaar werken.



ESLint-regels kunnen als een soort politieagent fungeren en de agent in de juiste richting sturen.

Gebruik hooks met strikte ESLint-regels

Het doel van goede agent-instructies en een duidelijke workflow is om meer voorspelbaarheid te creëren binnen agentic coding.

Strikte lintingregels kunnen daarbij helpen. Een linter kan ongewenste patronen herkennen, projectafspraken afdwingen en controleren op formatting. Met hooks kun je ervoor zorgen dat een agent deze regels automatisch volgt.

Skills ondersteunen hooks die op verschillende momenten kunnen worden uitgevoerd. Zo kun je bijvoorbeeld altijd een lintingcontrole uitvoeren voordat het resultaat aan de gebruiker wordt getoond. Genereert de AI-agent code die niet aan de regels voldoet? Dan faalt de validation hook en kan de agent de fout automatisch herstellen voordat het eindresultaat wordt gepresenteerd.

Ik raad aan om de Playwright ESLint-plugin met de aanbevolen instellingen te gebruiken. Je kunt de regels vervolgens verder aanscherpen. In mijn project moeten testtitels bijvoorbeeld altijd beginnen met 'Should'.

Hoe meer ongewenste code je automatisch met linting kunt onderscheppen, hoe eenvoudiger het wordt om AI-agents in één keer goede tests of functionaliteiten te laten schrijven. Zie je een agent regelmatig code schrijven die je niet wilt? Kijk dan eens of je daar een ESLint-regel voor kunt maken.

Hoe kun je hier zelf mee aan de slag?

Iedere codebase is anders. Neem mijn skill file en agent-instructies daarom niet één op één over. De achterliggende principes zijn wél breed toepasbaar:

1. Begin met duidelijke requirements. Een goede eerste fase leidt tot een beter plan en uiteindelijk een betere implementatie.
2. Geef de agent de juiste context. Maakt hij steeds dezelfde fouten, bijvoorbeeld met translation keys of toegankelijke rollen? Wijs hem dan op de bestanden waarin hij de juiste informatie kan vinden.
3. Voeg self-validation toe. Niets is zo frustrerend als een gegenereerde test die vervolgens niet werkt. Door self-validation en hooks onderdeel te maken van de workflow, wordt het resultaat van een agent voorspelbaarder en betrouwbaarder.

De belangrijkste les: laat een AI-agent niet zomaar code genereren, maar geef hem de kennis, werkwijze en controles mee die nodig zijn om tot een goed resultaat te komen.

Bekijk ter inspiratie mijn volledige SKILL.md- en AGENTS.md-files:

<https://github.com/tlolkema/skills-explained>



Collab. Test. Accelerate.



Onze vestigingen

Je vindt de locaties van onze labels verspreid door het hele land, zodat we dicht bij onze medewerkers en klanten zijn.

De labels van Sofius

Met onze 6 labels en meer dan 300 professionals op het gebied van softwaretesten werken wij elke dag aan het succes van onze klanten. Samen brengen we mensen, softwaresystemen en bedrijven v rder.



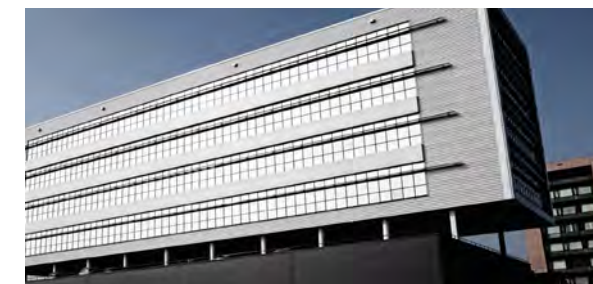
Bartosz ICT

Grensverleggend specialist in software testen. Vanuit een onbegrensde drive, kennis van het vak en een voorliefde voor het toevoegen van waarde.



PTWEE

Specialist op het gebied van testen van complexe enterprise software zoals SAP of Salesforce. Met een pragmatische aanpak helpt PTWEE organisaties om beter en slimmer te testen.



Squerist

Specialisten in verbeteren. Of het nu gaat om software testen, Agile & projectmanagement of business optimalisatie: we leveren kennis, ervaring en resultaat waarmee organisaties echt stappen zetten.



deTesters

Een hecht bedrijf van maximaal 20 senior professionals met maar   n focus: test automation. Wij steken veel tijd in kennisontwikkeling en delen onze kennis graag op conferenties en Meetups.



TestCoders

Specialisten die werken vanuit een eigen visie op het vakgebied: Next level Test Automation. TestCoders zijn versnellers, kennisdelers en pleziermakers.



TechChamps

Specialisten in strategische testautomatisering, met focus op wat   cht waarde levert voor de klant. Ons motto? We kiezen niet voor TestAllTheThings() omdat het kan, maar voor TestTheRightThings() en TestTheThingsRight().

Testautomatisering wordt steeds goedkoper om te maken. Door de normalisering van AI is code schrijven goedkoper en toegankelijker geworden. Dit heeft ook invloed op testautomatisering. De kosten-batenberekening verandert, omdat AI steeds meer kan helpen bij het schrijven van testcode. Daardoor wordt het eenvoudiger om testautomatisering te ontwikkelen en kunnen teams sneller meer tests maken.

Test automatisering

Tegelijkertijd neemt door AI ook de hoeveelheid software die wordt ontwikkeld toe. Er worden sneller en meer features opgeleverd en die moeten natuurlijk ook worden getest. Testautomatisering moet hierin meegroeien. Tools zoals Playwright spelen hier handig op in, met het aanbieden van Playwright MCP en Playwright CLI, en worden steeds vaker gebruikt in combinatie met AI voor het ontwikkelen en uitvoeren van tests.

De vraag is daarom niet óf je testautomatisering nodig hebt, maar vooral hoe je het gaat opzetten en hoe je blijft begrijpen wat het doet. Je wilt immers dat de tijd en investering die je erin steekt, leidt tot het gewenste resultaat. Of dit nu het verhogen van kwaliteit is, een betrouwbare applicatie of het sneller kunnen releasen door efficiënter te testen.

Daar zit ook een nieuw risico. Als AI het steeds eenvoudiger maakt om testcode te schrijven, kan de hoeveelheid testautomatisering snel groeien. De uitdaging verschuift dan van het schrijven van tests naar het beoordelen, begrijpen en onderhouden ervan. Een grote hoeveelheid automatisch gegenereerde tests betekent immers niet automatisch dat de juiste risico's worden afgedekt. Juist het kritisch beoordelen van wat AI maakt, wordt daarom belangrijker.

Het goed implementeren van testautomatisering in een complex landschap blijft daarmee een uitdaging. AI kan helpen om sneller te ontwikkelen, maar kennis van testen en de gebruikte tools blijft nodig om de juiste keuzes te maken.

Als je meer over de meest gebruikte (test)tools wilt lezen, dan kun je hierover informatie vinden in de TechRadar. Wil je onze TechRadar ontvangen, fysiek of digitaal? Ga naar: <https://testautomationtechradar.com/contact/>





Wim Selles

Hoe weet je of jouw team de juiste Test Automation tools en technieken kiest?

Nieuwe tools, frameworks en technieken volgen elkaar in hoog tempo op. Tegelijkertijd helpt AI ons steeds vaker bij het maken van technische keuzes. Maar hoe weet je of jouw team daadwerkelijk de juiste Test Automation tools en technieken kiest? Het antwoord begint niet bij de tool, maar bij de context.

Stel: je begint morgen met een compleet nieuw testautomatiseringsproject voor een webapplicatie. Welke tool kies je?

Playwright? Selenium? Cypress? WebdriverIO?

De kans is groot dat je direct een voorkeur hebt. Misschien omdat je een bepaalde tool al gebruikt. Omdat iemand in je team er veel ervaring mee heeft. Omdat een consultant hem adviseert. Of simpelweg omdat iedereen het erover heeft.

En tegenwoordig is er nog een adviseur bijgekomen: AI.

Vraag een AI coding agent om een nieuwe end-to-end testoplossing voor een moderne webapplicatie op te zetten en de kans is groot dat deze direct met een voorstel komt. Handig, maar er ontbreekt iets belangrijks: context. AI weet op dat moment misschien nog niets over je team, architectuur, requirements, infrastructuur of het kennisniveau binnen je organisatie. En toch ligt er al een tool op tafel.

Niet: welke tool is het beste?

Daar zit precies de valkuil. We stellen vaak de vraag: *welke tool is het beste?* Terwijl een veel relevantere vraag is: *welke oplossing past het beste bij onze situatie?* Want de beste Test Automation tool bestaat niet. De beste keuze voor jouw context wél.

Die context bestaat uit veel verschillende factoren. Welke technologie gebruikt je organisatie? Welke kennis heeft het team? Wie gaat de geautomatiseerde tests onderhouden? Wat voor product test je? Welke infrastructuur is al aanwezig? Welke browsers en devices moet je ondersteunen? En wat probeer je eigenlijk met Test Automation op te lossen?

Een team van developers dat volledig in TypeScript werkt, heeft bijvoorbeeld een ander vertrekpunt dan een testteam dat voornamelijk Java kent. Een organisatie met een bestaande Selenium Grid en honderden geautomatiseerde tests maakt een andere afweging dan een organisatie die vanaf nul begint. En zodra naast web ook native iOS- en Android-applicaties getest moeten worden, verandert de afweging opnieuw. Dezelfde tools kunnen dus in de ene situatie uitstekend passen en in de andere situatie minder logisch zijn. Maar als context zo belangrijk is, ontstaat een

volgende vraag: hoe weet je welke mogelijkheden er überhaupt zijn en welke het onderzoeken waard zijn?

Precies vanuit die vraag ontstond de Test Automation TechRadar. Binnen de Test Automation Group van Sofius doen meer dan 60 Test Automation Engineers bij uiteenlopende organisaties ervaring op met verschillende tools, technieken, frameworks en platforms. Die praktijk-kennis wordt verzameld en gestructureerd in de TechRadar.

Van populariteit naar onderbouwde keuzes

Dat overzicht wordt steeds belangrijker. Het Test Automation-landschap groeit en verandert snel. Naast bekende tools als Selenium, Cypress, Playwright en Appium ontstaan voortdurend nieuwe technieken, platforms en frameworks.

Daar komen ontwikkelingen rondom AI nog eens bovenop. Denk aan agents, context management en agentic test writing. AI kan niet alleen helpen bij het schrijven van testcode, maar krijgt steeds meer mogelijkheden om zelfstandig testwerkzaamheden voor te bereiden en uit te voeren.

Een paar jaar geleden ging de discussie vooral over Selenium versus Cypress en later over Cypress versus Playwright. Inmiddels ontstaan heel andere vragen. Hoe zetten we AI-agents in voor testing? Hoe zorgen we dat zo'n agent voldoende context heeft? Willen we AI zelfstandig tests laten schrijven? En wat betekent dat voor de rol van de Test Automation Engineer?

Juist doordat de mogelijkheden groeien, wordt het belangrijker om niet automatisch achter de nieuwste of populairste technologie aan te lopen. De TechRadar helpt om het speelveld te overzien en gericht te bepalen welke mogelijkheden het onderzoeken waard zijn.

Waarom 'Hold' niet betekent dat een tool slecht is

Selenium is een mooi voorbeeld van hoe zo'n radar gebruikt moet worden. Selenium staat binnen de TechRadar op Hold. Dat betekent niet dat Selenium ineens een slechte technologie is. Het is een bewezen technologie, wordt wereldwijd ondersteund, kent een enorm ecosysteem en wordt binnen veel organisaties nog dagelijks succesvol gebruikt.

Voor een bestaande omgeving kan Selenium dan ook nog steeds een prima keuze zijn. Maar begin je vandaag volledig opnieuw, dan zijn er moderne alternatieven die je op zijn minst serieus wilt onderzoeken. Dat is een belangrijk verschil.

Hetzelfde geldt andersom. Dat een tool als Playwright op Adopt staat, betekent niet automatisch dat Playwright voor ieder nieuw project de beste keuze is.

Voor een nieuw webproject met een TypeScript-team kan Playwright bijvoorbeeld een heel logische kandidaat zijn. Maar zodra dezelfde organisatie ook native mobiele apps wil testen en al beschikt over een bestaande automation-infrastructuur, kan een andere oplossing beter passen.

De tools zijn niet veranderd. De context wel. En daarmee verandert ook de afweging.

Drie vragen vóórdat je een tool kiest

Hoe weet je dan of jouw team de juiste keuze maakt?

Niet omdat AI een tool adviseert. Niet omdat iedereen hem gebruikt. Niet omdat een technologie op Adopt staat. En ook niet alleen omdat een consultant zegt dat het een goede oplossing is.

Een goede keuze kun je onderbouwen. Voordat je besluit welke technologie je inzet, zijn daarom drie vragen essentieel:

1. **Welk probleem proberen we op te lossen?**
2. **Welke context bepaalt onze keuze?**
3. **Welke alternatieven hebben we serieus onderzocht?**

De TechRadar kan daarbij waardevolle input geven. Niet door de beslissing voor je te nemen, maar door zichtbaar te maken welke mogelijkheden er zijn, welke ontwikkelingen het onderzoeken waard zijn en welke ervaringen andere Test Automation Engineers ermee hebben.

Juist in een wereld waarin tooling én AI zich steeds sneller ontwikkelen, wordt het belangrijker om niet zo snel mogelijk een antwoord te vinden, maar eerst de juiste vragen te stellen.

Understand. Assess. Experiment. Decide.

Gebruik technologie om je keuzes beter te onderzoeken, maar besteed de beslissing zelf niet uit. Ook niet aan AI.

Test Automation TechRadar: overzicht in een snel veranderend vakgebied

De wereld van Test Automation ontwikkelt zich razendsnel. Om organisaties te helpen overzicht en richting te houden, ontwikkelden deTesters, TestCoders en TechChamps, drie labels binnen Sofius, samen de Test Automation TechRadar.

De radar bundelt de praktijkervaring van meer dan 60 Test Automation Engineers. Tools, technieken, platforms en talen & frameworks worden beoordeeld en ondergebracht in vier ringen:

- **Adopt** – technologie waarin veel vertrouwen bestaat en die zich in de praktijk heeft bewezen.
- **Trial** – veelbelovend en interessant om in een echte situatie uit te proberen.
- **Assess** – de moeite waard om te onderzoeken en te bepalen of deze relevant kan zijn.
- **Hold** – technologie waarbij terughoudendheid bij nieuwe inzet verstandig is.

Belangrijk: de TechRadar is **geen ranglijst en geen kant-en-klaar keuzeadvies**.

Adopt betekent niet dat iedere organisatie een technologie moet gebruiken. En Hold betekent niet dat een tool slecht is. De radar laat zien hoe experts vanuit hun praktijkervaring naar technologieën kijken en helpt bepalen welke opties het onderzoeken waard zijn.

De TechRadar is bovendien een levend instrument. Nieuwe ontwikkelingen en praktijkervaringen worden continu onderzocht en verwerkt, zodat de radar meebeweegt met het vakgebied.

Wil je onze TechRadar ontvangen, fysiek of digitaal?

Ga naar: <https://testautomationtechradar.com/contact/>



Het testen van mobiele apps op mobiele devices met Appium

Het testen van mobiele apps op mobiele devices is een vak apart. Toen ik bij Medux begon aan een testproject voor de handscanners in het magazijn, viel me op hoe weinig er online te vinden is over deze specifieke tak van sport. Over het testen van webapplicaties is een overvloed aan informatie beschikbaar: van YouTube-tutorials tot diepgaande blogs. Maar zodra je je begeeft in de wereld van native mobiele apps en fysieke devices, wordt het ineens stil.

Kun je testautomatisering wel toepassen op een handscanner of andere device met een specifieke app? Jazeker! In dit blogartikel deel ik de inzichten die ik heb opgedaan. Wat heb je nodig om te beginnen met het testen op mobiele devices? Wat is het verschil tussen het testen van mobiele webapplicaties versus native mobiele apps, en waarom is dat verschil zo belangrijk? Ik bespreek de voor- en nadelen van testen op echte apparaten versus emulators. En ik laat zien hoe je mobiele apps kunt testen met tools zoals Robot Framework en Appium. Want ook als je werkt met gespecialiseerde mobiele devices, zijn er wel degelijk mogelijkheden om te automatiseren.

Verskil tussen tests op een browser en op een mobiele applicatie

Bij het testen van software op mobiele devices is het belangrijk om onderscheid te maken tussen het testen van mobiele webapplicaties en het testen van native of hybride mobiele applicaties. Mobiele webapplicaties worden via de browser op een mobiel apparaat geopend en delen doorgaans dezelfde codebasis als de desktop-versie van de website. Hierdoor kunnen automatische tests vaak worden hergebruikt; de applicatie is immers in essentie hetzelfde, alleen aangepast aan kleinere schermformaten en touch-bediening.

Bij het testen van mobiele applicaties ligt dat anders. Deze apps zijn meestal geschreven in andere programmeertalen (zoals Swift voor iOS

of Kotlin voor Android) en functioneren los van de browser. Het zijn zelfstandige programma's die je niet zomaar op een desktop kunt draaien. Dit vraagt om een compleet andere testaanpak, met specifieke tools en emulators of fysieke devices om gedrag, performance, installatieprocessen en interactie met hardware zoals camera of gps te kunnen testen.

Waar mobiele webtests vaak een uitbreiding zijn van je reguliere teststrategie, vereist het testen van apps op mobiele devices een gespecialiseerde en meer device-georiënteerde aanpak.

Automatisch testen van mobiele devices: wat heb je nodig om te kunnen beginnen?

Testautomatisering op mobiele devices lijkt op web automatisering, maar vraagt om een paar extra stappen. Waar je bij web testen vaak al uit de voeten kunt met een browser en Selenium, heb je voor het testen van mobiele devices een ander soort opstelling nodig. Dit heb je nodig:

- **De applicatie zelf.** Je hebt het installatiebestand van de app nodig (bijvoorbeeld een .apk voor Android of een .ipa voor iOS). Zorg ervoor dat je toegang hebt tot de juiste, meest recente versie van de app, vaak kan je dit opvragen bij de ontwikkelaar.
- **Een testdevice.** Dit kan een fysiek apparaat zijn, zoals een smartphone of – in mijn geval – een handscanner die via een USB-kabel met je computer wordt verbonden.
- **Een server.** De devices worden bediend via de Appium server. De server maakt het mogelijk om de testsoftware te verbinden met de devices waarop getest wordt.
- **Een driver.** Je hebt software nodig die automatische interactie met het apparaat mogelijk maakt. In het geval van Android gebruik je bijvoorbeeld UIAutomator als stuurprogramma om testacties op het device uit te voeren.

Testen op een fysiek device of emulator: wat kies je?

Emulators zijn softwarematige simulaties van mobiele apparaten op je computer. Ze bootsen het besturingssysteem en de hardware na, waardoor je verschillende devices en OS-versies kunt testen zonder fysieke apparaten. Ideaal voor snelle, schaalbare tests en integratie in CI/CD-pipelines.

“Emulators bieden veel flexibiliteit en schaalbaarheid, zeker in grotere of complexere testomgevingen.”

Voordelen van emulators:

- Geen fysieke apparaten nodig, dit bespaart ruimte en kosten.
- Snel schakelen tussen verschillende modellen en OS-versies.
- Simuleren van netwerkcondities (slechte wifi, geen verbinding, GPS-locaties, tijdzones).
- Automatisering in pipelines mogelijk zonder handmatige interventie.

Nadelen van emulators:

- Vereist veel rekenkracht van je computer.
- Vaak trager en minder betrouwbaar dan de echte devices.
- Hardware functies zoals camera, bluetooth of sensoren zijn moeilijk te simuleren.
- Bij specialistische devices (zoals handscanners) voldoen emulators vaak niet.

Fysieke apparaten zijn cruciaal als je wilt testen hoe de app écht functioneert in de praktijk, vooral bij hardware afhankelijke functies. Bij Medux testten we op exact dezelfde handscanners die ook in het magazijn gebruikt worden. Zo wisten we zeker dat onze tests realistisch waren en dat wat wij zien ook daadwerkelijk de gebruikerservaring weerspiegelt.



Wanneer kies je wat?

In de praktijk zie je vaak een mix: dagelijks testen op emulators en periodieke validatie op echte devices voor maximale betrouwbaarheid.

Gebruik een emulator als...	Gebruik een fysiek device als...
Je snel wilt schakelen tussen devices/versies	Je test op specifieke hardware zoals handscanners
Je tests wilt integreren in een CI/CD-pipeline	Je app sterk afhankelijk is van sensoren, bluetooth, camera
Je netwerkcondities wilt simuleren	Je exacte gebruikers-ervaring wilt nabootsen
Je geen grote verzameling fysieke apparaten hebt	Je met één specifiek device werkt

Mobiel testen met Robot Framework en Appium

Je hebt een applicatie nodig die jouw test-stappen uitvoert – net zoals Selenium dat doet voor browsers – en in de wereld van mobiele apps is dat Appium. Appium fungeert als een brug tussen je testcode en het mobiele apparaat (fysiek of virtueel). Het ondersteunt bijna elke programmeertaal en werkt goed samen met Robot Framework, dat een uitgebreide AppiumLibrary biedt.

Via Appium – dat als server draait – geef je testinstructies door, bijvoorbeeld: “vind deze knop en klik erop”. Alleen: je kunt in een app niet zomaar rechtsklikken om te inspecteren wat je ziet, zoals in HTML. Daarom gebruik je de **Appium Inspector**: die laat je de structuur achter het scherm zien, opgebouwd in XML.

Die XML-structuur is je gids tijdens het testen. Net zoals je in een website via HTML kunt navigeren naar knoppen of tekstvelden, kun je dat in apps via XML. Maar mobiel testen is minder vergevingsgezind. Teruggaan? Dat moet je expliciet in je script zetten. Direct naar een andere ‘pagina’? Vergeet het maar.

Met eenvoudige keywords zoals Open Application, Click Element en Wait Until Page Contains bouw je intuïtieve tests. Ook typische mobiele acties als swipe, scroll en long press zijn goed te automatiseren. Soms loop je tegen beperkingen aan, bijvoorbeeld bij het bedienen van externe apps (zoals de camera). Dan kun je aanvullende tools als Android Debug Bridge (ADB) gebruiken, al vraagt dat wel extra technische kennis.

Het belangrijkste is echter de samenwerking met je developers en inzicht in het type app en device waarop je test. Met die basis en Appium als tussenlaag kun je met Robot Framework effectief, betrouwbaar en schaalbaar mobiele apps automatiseren.

Conclusie

Goed mobiel testen vraagt meer dan alleen de juiste tools. Zorg voor inzicht in het type app dat je test (native of web), de devices waarop je test (fysiek of virtueel), en werk nauw samen met developers om de testbaarheid van de app te vergroten. Met die basis – en Appium als tussenlaag – kun je met Robot Framework effectief en schaalbaar mobiele apps testen.



Wat is de staat van Robot Framework?

Robot Framework staat er in 2026 goed voor. Dat zien we zowel aan het aantal downloads als aan de inzet bij klanten. Maandelijks wordt Robot Framework miljoenen keren gedownload en ook binnen onze projecten komen we het framework regelmatig tegen. Die positie zien we ook terug in de Test Automation TechRadar, ontwikkeld door drie Sofius-labels. Robot Framework valt daarin bij veel toepassingen in de categorie Adopt: bewezen technologie die direct kan worden ingezet wanneer deze aansluit bij de eisen van het project.

Tegelijkertijd heeft Robot Framework zich de afgelopen jaren flink ontwikkeld. Niet alleen het framework zelf veranderde, ook de tooling en libraries eromheen zijn volwassener geworden.



David Italiander



Robot Framework blijft zich ontwikkelen

In 2026 zitten we inmiddels op Robot Framework 7.4.2. De afgelopen versies brachten verschillende verbeteringen die het framework krachtiger en prettiger in gebruik maken. Denk bijvoorbeeld aan:

- Nieuwe control flow syntax, zoals TRY/EXCEPT, inline IF, RETURN, BREAK en CONTINUE;
- Native VAR syntax voor het toewijzen van variabelen;
- Een verbeterde Listener Interface;
- Dark mode voor Robot Framework-rapportages.

Daarmee is Robot Framework niet alleen technisch verder ontwikkeld, maar sluit het ook steeds beter aan op moderne softwareontwikkeling en testautomatisering.

Van RIDE naar Visual Studio Code

Ook de ontwikkelomgeving rondom Robot Framework is veranderd. Visual Studio Code is inmiddels voor veel developers en testers de standaardomgeving geworden.

Lange tijd werd daarin veel gebruikgemaakt van de Robot Framework Language Server van Robocorp. Nadat Robocorp stopte met de verdere ontwikkeling daarvan, ontstond behoefte aan een alternatief dat nieuwe Robot Framework-functionaliteiten bleef ondersteunen.

Dat alternatief is Robotcode, ontwikkeld door imbus. Een Visual Studio Code extensie die dezelfde features bood als de Robot Framework Language Server maar ook de nieuwe features ondersteunde en actief wordt onderhouden. Enkele features waarmee Robotcode het leven van de Robot Framework-ontwikkelaar makkelijk maakt zijn:

- **Syntax Highlighting:** Kleurt de code (keywords, variabelen, instellingen) zodat het leesbaar is en je sneller fouten spot. Basic code completion.
- **Ingebouwde debugger:** Helpt je te begrijpen waarom een test faalt door stap voor stap door de code te lopen.
- **Directe uitvoering:** Je kunt tests starten met één druk op de knop, zonder dat je handmatig commando's in de terminal hoeft te typen.

“Robot Framework ontwikkelt zich volop: niet alleen het framework zelf, maar ook de tooling en libraries eromheen worden steeds volwassener.”

SeleniumLibrary of BrowserLibrary?

Ook binnen het library-landschap is er meer te kiezen. Waar SeleniumLibrary lange tijd vrijwel vanzelfsprekend was voor het automatiseren van browserinteracties, heeft BrowserLibrary inmiddels een stevige positie opgebouwd.

SeleniumLibrary is volwassen, betrouwbaar en breed ondersteund. Doordat Selenium via de WebDriver API met browsers communiceert, kunnen echter timingproblemen ontstaan die soms leiden tot instabiele of ‘flaky’ tests.

BrowserLibrary is gebaseerd op Microsoft Playwright en kiest voor een modernere aanpak. Een belangrijk voordeel is auto-waiting: de library wacht automatisch totdat een element klaar is voor interactie. Ook kunnen eenvoudig meerdere browsercontexten binnen één testrun worden gebruikt, bijvoorbeeld om verschillende gebruikersrollen tegelijkertijd te testen.

Dat maakt BrowserLibrary voor veel nieuwe projecten een interessante keuze. Tegelijkertijd blijft SeleniumLibrary relevant, zeker in bestaande omgevingen waarin al veel kennis en automatisering aanwezig is. De juiste keuze blijft dus afhankelijk van de context.

Een eigen Robot Framework-certificering

Een andere belangrijke ontwikkeling is de komst van Robot Framework Certified Professional (RFCP). Dit certificeringsprogramma werd tijdens RoboCon 2025 in Helsinki aangekondigd en richt zich volledig op Robot Framework zelf.

Kandidaten worden onder andere getoetst op kernconcepten, syntax en structuur. Het examen bestaat uit multiplechoicevragen, duurt 60 minuten en vereist een score van minimaal 65%.

De certificering is interessant voor zowel beginnende als ervaren Robot Framework-gebruikers. Het certificaat laat zien dat iemand de basis van het framework goed beheerst. In



de praktijk blijft ervaring met libraries zoals BrowserLibrary of SeleniumLibrary daarnaast belangrijk: Robot Framework wordt vrijwel altijd gecombineerd met externe libraries.

Binnen Sofius hebben inmiddels meerdere consultants het RFCP-certificaat behaald.

En richting 2027?

Robot Framework laat eind 2026 zien dat een volwassen open-source framework niet stil hoeft te staan. Het framework zelf ontwikkelt door, de tooling rondom Visual Studio Code is vernieuwd en moderne libraries zoals BrowserLibrary maken het mogelijk om aan te sluiten bij ontwikkelingen als Playwright.

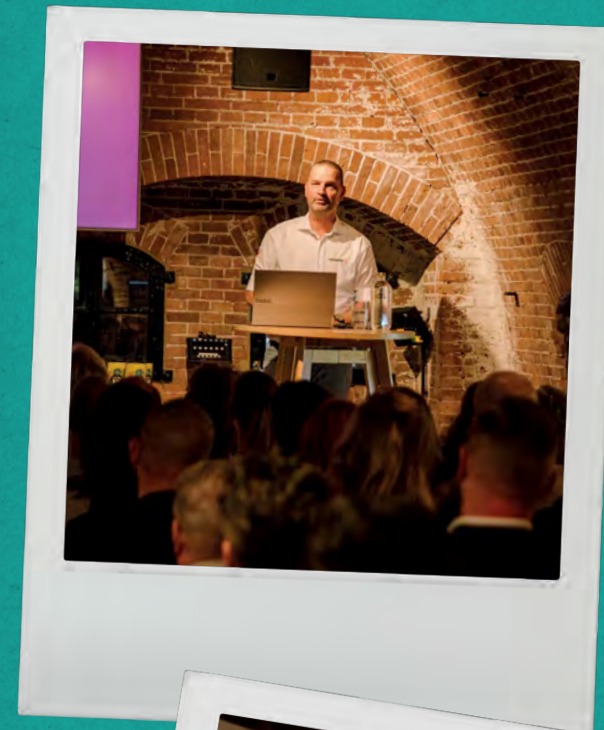
Richting 2027 verwachten we dat vooral die combinatie belangrijk blijft: een stabiele basis, een actief ecosysteem en de mogelijkheid om nieuwe technologieën relatief eenvoudig te integreren. Juist daardoor blijft Robot Framework een relevante keuze voor organisaties die testautomatisering schaalbaar en toekomstbestendig willen inrichten.

Benieuwd of Robot Framework past binnen jouw testautomatiseringsstrategie? Onze consultants denken graag mee over de juiste aanpak, tooling en inrichting voor jouw project.

Collab.
Test.
Accelerate.



Trends in Testing 2025



Het beheersen van soft skills is cruciaal om resultaten te behalen, mensen mee te krijgen met goede ideeën en effectief samen te werken. Goede consultants onderscheiden zich niet alleen door hun inhoudelijke kennis en kunde, maar juist ook door hun bovengemiddelde soft skills.

Soft skills

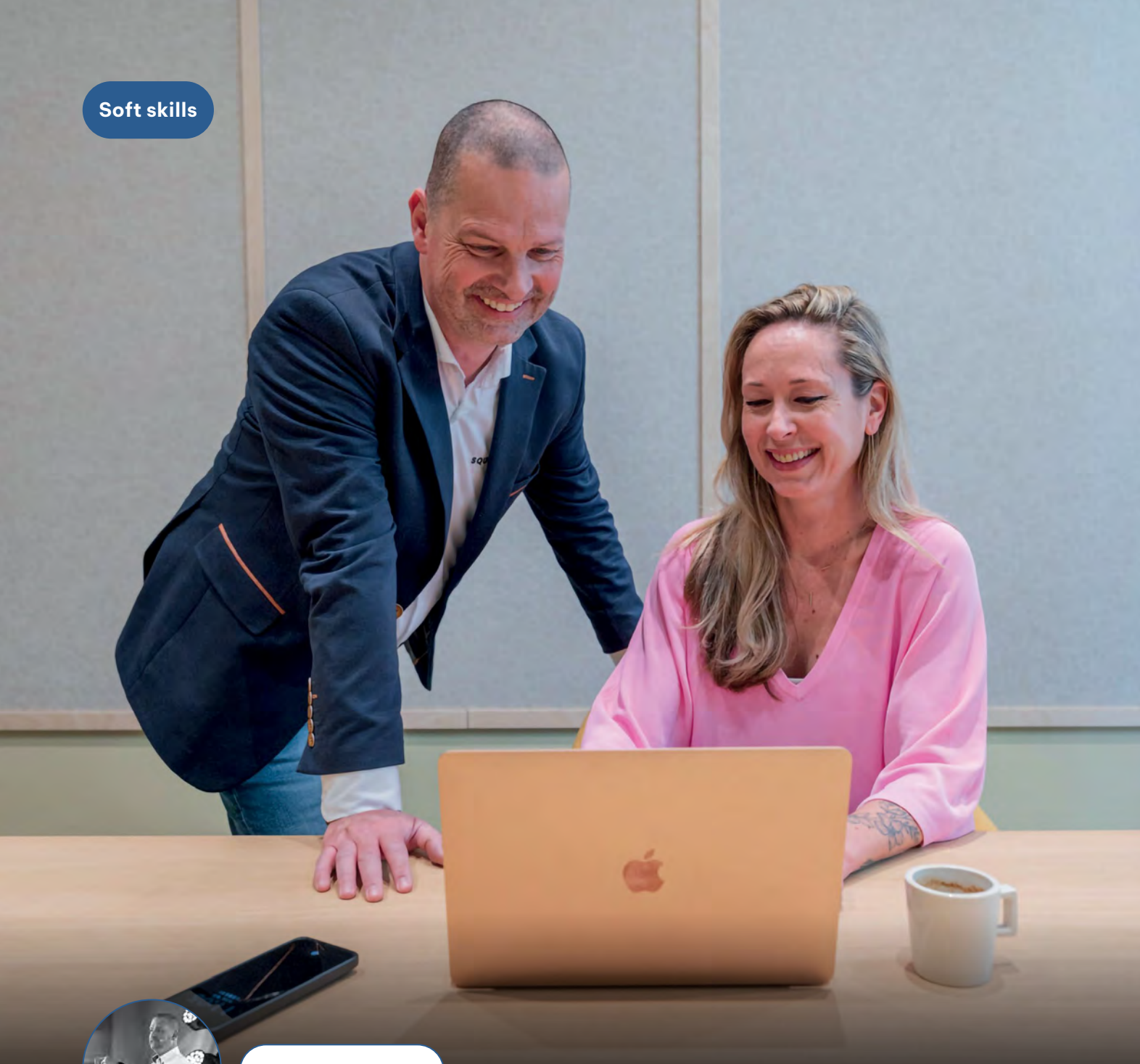
De trend soft skills is een samenvoeging van de onderwerpen: Leiderschap en Communicatieve vaardigheden. Deze onderwerpen hebben veel overlap, en zo hebben we ervoor gekozen om ze samen te voegen.

We zien een toenemende aandacht voor soft skills in de rol van testers in een wereld waar AI bestaat. Mede doordat AI steeds vaker taken uitvoert, verschuift de focus van onze testers van het uitvoeren van testen naar het kwaliteitsdenken versterken in het hele softwareontwikkelproces. Hoe breder kwaliteitsdenken binnen een organisatie wordt gedragen, hoe beter de kwaliteit en hoe korter de doorlooptijd doorgaans is. Het sneller begrijpen van elkaar, het voorkomen van langs elkaar heen praten, en het kritisch benaderen van specificaties en klantwensen leiden uiteindelijk tot betere producten én sterkere organisaties.

Om dit toe te kunnen passen, hebben al onze testers sterke soft skills. Hier zien we onder andere de volgende onderwerpen langskomen: leiderschap (ook zonder formele titel), coaching, verandering creëren, en vertrouwen wekken.

We zien ook steeds meer collega's op de werkvloer, die over het algemeen veel waarde hechten aan een gezonde balans tussen werk en privé. Daarnaast willen ze hun werk ervaren als zinvol, gaan ze anders om met feedback, hebben ze soms een iets kortere aandachtspanne, en hebben een andere communicatiestijl. Goede soft skills geven je de flexibiliteit om effectief samen te kunnen werken..





René van Veldhuijzen

Soft skills: de strategische spiergroep van de Quality Engineer

In een vakgebied dat zich razendsnel ontwikkelt, van AI gedreven test-automatisering tot complexe ketens in realtime delivery, is het verleidelijk om te denken dat techniek de hoofdrol speelt. Maar wie echt goed kijkt, ziet een ander patroon. De software test specialist (tegenwoordig beter bekend als de Quality Engineer), die het verschil maakt is niet per se degene met de meeste tool ervaring, maar wel degene die beschikt over een set vaardigheden die vaak onzichtbaar en ondergewaardeerd blijven: soft skills.

Bij Sofius geloven we dat kwaliteit begint bij mensen. Vanuit onze ervaring weten we dat energie, empathie en samenwerking net zo bepalend zijn voor succesvolle releases als frameworks en CI/CD-pijplijnen. Soft skills zijn geen bijzaak meer, ze zijn de strategische spiergroep van ons vak.

Communicatie: de zuurstof van kwaliteit

Een quality engineer communiceert continu: met ontwikkelaars, product owners, architecten, gebruikers, management en onze nieuwste gesprekspartner: Artificiële Intelligentie (AI). Heldere communicatie versnelt besluitvorming, voorkomt misverstanden en maakt risico's bespreekbaar. Het vermogen om testplannen, testscripts en complexe bevindingen te vertalen naar begrijpelijke taal is absoluut geen luxe, maar een kerncompetentie. In hybride teams is geschreven communicatie zelfs een kritische succesfactor geworden.

Kritisch denken: constructieve twijfel als superkracht

Kritisch denken is de kunst van het doorvragen, het herkennen van patronen, het toetsen van aannames, het onderscheiden van ruis & signalen en het zien van risico's die niet expliciet in requirements staan. En het is precies die mindset die wij binnen Sofius koesteren: niet om negatief te zijn, maar om constructief te twijfelen. Deze

vaardigheid tilt quality engineers van uitvoerders naar strategische kwaliteitsadvocaten die de juiste vragen stellen op het juiste moment.

Empathie: testen door de ogen van de gebruiker

Empathie lijkt misschien een zachte vaardigheid, maar in de praktijk is het een harde impactmaker. Quality engineers die de context, beperkingen en emoties van gebruikers begrijpen, anticiperen eerder op gebruiksproblemen, identificeren waarde beïnvloedende bevindingen effectiever en helpen bij het vormgeven van oplossingen die de gebruikerservaring daadwerkelijk verbeteren. Empathie maakt van quality engineers geen gatekeepers, maar gebruikers kampioenen; een rol die technologie niet kan overnemen.

Samenwerking: de stille motor achter succesvolle releases

Kwaliteit ontstaat nooit in isolatie. Het is het resultaat van samenwerking tussen disciplines, belangen, perspectieven en dus kortgezegd: mensen. Soft skills zoals actief luisteren, afstemmen, feedback geven en verwachtingen managen bepalen hoe soepel een team beweegt. Teams die dit bewust ontwikkelen, leveren aantoonbaar betere resultaten en ervaren veel minder frictie.

Aanpassingsvermogen: de Sofius vibe

De quality engineer (lees: moderne tester) moet wendbaar zijn. Nieuwe tools, nieuwe deliverymodellen, nieuwe risico's, nieuwere AI-versies en diverse generaties op de werkvloer: het speelveld verandert continu. Aanpassingsvermogen is de soft skill die quality engineers toekomstbestendig maakt. Met een open blik, nieuwsgierigheid, vakkennis en een gezonde dosis positiviteit wordt elke verandering een kans om kwaliteit te verbeteren.

Slotgedachte

Soft skills zijn geen randvoorwaarde, ze zijn de strategische kern van moderne softwarekwaliteit.

*Techniek maakt testen mogelijk.
Soft skills maken testen waardevol.*

Have you lost your mind(set)?

Stel je voor dat we een paar duizend jaar terug in de tijd zijn. Je bent op jacht naar je volgende maaltijd, je hebt honger, het is koud en je bent al uren aan het jagen. Plotseling hoor je geritsel in de struik naast je. Het klinkt groot. Er gaat een rilling over je rug, je adrenaline schiet omhoog. Je moet beslissen: gaat dit mij doden, of heb ik straks avondeten? De verkeerde beslissing kan je het leven kosten.

De evolutie gaat langzamer dan onze technologische ontwikkeling. In onze huidige samenleving functioneren we nog steeds met dit mechanisme, maar het helpt ons niet meer zoals het dat deed toen we jager-verzamelaars waren.



Linda van de Vooren

Er zijn vier ingrediënten die ons sturen in onze interne besluitvorming en redenering.

Bewustzijn – Als je je niet bewust bent van wat je denkt of waarop je reageert, dan kun je je redeneervermogen niet gebruiken om er iets aan te doen.

Vrije wil – Zonder de vrije wil om een andere keuze te maken, kun je dat letterlijk niet doen.

Creatieve verbeelding – We gebruiken onze creatieve verbeelding om andere opties en andere perspectieven te bedenken. Deze leiden tot de mogelijkheid om een andere keuze te maken die je in je situatie of reactie zou kunnen helpen.

Geweten – Zonder ons geweten zouden we ongeremde keuzes maken die onszelf en de mensen om ons heen mogelijk zouden schaden.

Met deze vier ingrediënten kunnen we een win-winsituatie creëren. Je kunt je eindeloze mogelijkheden voorstellen waarbij je acties positieve effecten hebben, en hoewel kiezen voor een win-win een logische zaak lijkt, doen mensen dat in de praktijk vaak niet.

Soms neem je een beslissing die niet goed voor jezelf is, maar wel gunstig voor de ander (verlies-win). Of je kies iets wat goed is voor jezelf, maar niet voor de ander (win-verlies). Misschien maak je een keuze die voor niemand gunstig is (verlies-verlies). De vijfde optie: niet kiezen.

Win-Win	Win-Lose
Lose-Win	Lose-Lose
Don't choose	

We hebben de verantwoordelijkheid om onze reacties te kiezen. Of, wanneer onze reactie of mindset niet goed zit, daar een verandering in aan te brengen. Je (huidige) mindset hangt sterk samen met je huidige reacties.

Hoe haal je iemand dus uit een overdreven reactie? Eén kleine truc is om een simpele vraag te stellen die een beroep doet op je geheugen, zoals 'Wat heb je drie avonden geleden gegeten?'. Het beantwoorden van deze vraag maakt gebruik

van een ander deel van je brein dan reageren, en dit zal de emotionele reactie sterk verminderen of laten verdwijnen.

Wanneer je denkt dat er geen bugs in het systeem zitten, zul je er waarschijnlijk ook geen vinden. Wanneer je denkt dat alle bugs gewoon aan jouw fouten liggen, worden bugs niet onderzocht of opgelost. In beide gevallen zal de kwaliteit van je applicatie afnemen. Daag jezelf uit om een andere keuze te maken. Pas je gedrag aan om terug te keren naar de juiste mindset voor het vinden van bugs en het waarborgen van kwaliteit.

Wat gebeurt er wanneer anderen jouw mindset veranderen? Externe invloeden zullen altijd invloed hebben op jouw mindset; de truc is om je daarvan bewust te zijn. Dit wordt soms 'beïnvloeding' of 'marketing' genoemd. Het is onvermijdelijk. Als je je bewust bent van deze invloed, kun je ervoor kiezen anders te handelen wanneer je verstrikt raakt in een negatieve spiraal of reacties die niet goed voelen. Je koopt toch ook niet letterlijk alles waar je een advertentie voor ziet? Je maakt de keuze om dat niet te doen, ondanks de manipulaties.

Aangezien je altijd wordt beïnvloed, beïnvloedt jouw gedrag ook altijd anderen. Dit is niet iets wat je kunt voorkomen. Wat je wél kunt doen, is ervoor zorgen dat mensen worden beïnvloed op een manier die zowel voor jou als voor hen gunstig is. Dit is niet eenvoudig, maar met enige kennis over hoe je brein jou probeert te helpen, kun je ermee werken en een andere keuze maken.

Ik heb een mindsetmanifest gemaakt! De manier waarop het Agile manifest is geschreven, is een prachtige manier om te benadrukken wat belangrijker is dan iets anders dat misschien minder behulpzaam is, maar dat zal gebeuren. In onze interacties met anderen en de keuzes die we vrijelijk kunnen maken,

We waarden

Emotionele controle boven emotionele zwakte

Positief denken boven negatief denken

Een groei mindset boven een fixed mindset

Waardering boven verwachting

Kleine aanpassingen boven grote verwachtingen

De mogelijkheid om iemands mindset te veranderen brengt de verantwoordelijkheid met zich mee om een bewuste keus te maken.



Het kan goed komen. Maar ook zomaar niet.

Niet alles wat kan, moet je beloven!

Iedere keer als we aan een project beginnen, is één ding meestal al duidelijk: de afgesproken einddatum. Hoe we daar als team precies gaan komen, is op dat moment vaak nog minder duidelijk. Dat is op zichzelf niet vreemd. In het begin van een traject zijn er altijd onzekerheden.

Toch ontstaan er in die fase al snel verwachtingen. Dat het haalbaar is. Dat het wel goed komt. Dat we de planning gaan redden. Vaak zijn dat goedbedoelde uitspraken, bedoeld om richting en vertrouwen te geven. Maar zodra anderen hun planning gaan baseren op wat wij zeggen, worden het meer dan alleen woorden.

In de sport hoor je iets vergelijkbaars. Wanneer een seizoen nog niet verloren is, verschijnt al snel de rekensom: als we vanaf nu alles winnen, kunnen we nog kampioen worden. Rekenkundig klopt dat. Maar het roept ook een andere vraag op: als het tot nu toe niet gelukt is om alles te winnen, wat maakt dat het vanaf nu wel gaat gebeuren? Wat is er veranderd?

“Als we vanaf nu alles winnen, kunnen we nog kampioen worden.”

Daarbij wordt het verschil tussen wat mogelijk is en wat waarschijnlijk is vaak kleiner gemaakt dan het in werkelijkheid is.

Dat zie je niet alleen in sport, maar ook in projecten. Nog voordat een project begint, worden verwachtingen vaak al vastgelegd in bijvoorbeeld een offerte. Daarin staat wat er geleverd gaat worden, hoeveel tijd dat kost en wat het ongeveer zal kosten. In theorie is zo'n offerte een eerlijke inschatting van wat nodig is

om het werk goed te doen. In de praktijk worden offertes echter vaak vooral op prijs beoordeeld. Wie het laagste bedrag noemt, heeft een grotere kans om de opdracht te krijgen.

De verleiding ontstaat dan om uit te gaan van het meest optimistische scenario. Als alles meezit, als er geen onverwachte problemen zijn en als alles in één keer goed gaat, dan zou het misschien inderdaad in die tijd en voor dat bedrag kunnen. Net als bij de rekensom in de sport klopt dat op papier. Maar in de praktijk is er bijna altijd iets dat anders loopt dan verwacht.

Ik heb ook meegemaakt dat verwachtingen blijven bestaan terwijl de werkelijkheid al veranderd is. In een project waarbij ik betrokken was, was afgesproken dat er op een bepaalde datum 110 services zouden worden opgeleverd. Dat was de afspraak waar iedereen zijn planning op baseerde. Twee weken voor de oplevering kwam het bericht: 110 gaan we niet redden. De logische vraag was: hoeveel dan wel? Het antwoord was: “geen één”.

Het probleem was niet dat het doel niet gehaald werd. Dat kan gebeuren. Het probleem was dat de werkelijkheid blijkbaar al langer bekend was, maar niet eerder gedeeld werd. In de tussentijd hadden anderen hun planning gebaseerd op iets wat al niet meer realistisch was. De reden daarvoor was niet zozeer optimisme, maar eerder politieke voorzichtigheid. Zolang er geen slecht nieuws werd gedeeld, konden verwachtingen blijven bestaan. Ik heb daar destijds discussie over gehad met een accountmanager. Mijn overtuiging was – en is nog steeds – dat openheid uiteindelijk altijd beter werkt dan het vasthouden aan een optimistisch verhaal dat al niet meer klopt.

Die overtuiging komt ook uit een eerdere ervaring. Ooit werkte ik bij een organisatie waar een eenvoudige gedachte centraal stond. Een fout maken was vervelend, maar niet het ergste wat kon gebeuren. Wat echt problematisch werd gevonden, was wanneer iemand probeerde een fout eerst zelf op te lossen zonder hulp te vragen. Niet omdat die persoon iets verkeerd wilde doen, maar omdat elke dag dat een probleem onzichtbaar blijft de kans verkleint dat de schade nog beperkt kan worden. Dat principe is me altijd bijgebleven. Niet omdat fouten vermeden moeten worden – die horen bij elk project – maar omdat

transparantie vaak het verschil maakt tussen een probleem dat beheersbaar blijft en een probleem dat uiteindelijk niet meer op te lossen is.

Misschien is het ook een menselijke neiging om het risico dat het dichtstbij is als het grootste te zien. Het ongemak van vandaag – een lastig gesprek, een tegenvaller moeten melden – voelt vaak zwaarder dan de mogelijke gevolgen later. Terwijl in projecten het omgekeerde vaak waar is: hoe eerder een probleem gedeeld wordt, hoe groter de kans dat er samen nog een oplossing gevonden kan worden.

Als er iets niet goed gaat, is het daarom vaak het verstandigst om dat zo snel mogelijk te delen. Niet omdat anderen onderdeel zijn van het probleem, maar omdat ze juist onderdeel van de oplossing kunnen worden.

In veel projectteams zit een sterke drive om iets neer te zetten waar klanten echt mee verder kunnen. Dat is vaak precies wat een team goed maakt. De wil om te leveren, om problemen op te lossen, om afspraken na te komen. Maar juist omdat teams zo graag willen dat het lukt, kan de wens om te leveren soms groter worden dan de realistische verwachting dat het ook echt gaat lukken.

Juist wanneer de uitkomst voor een ander belangrijk is, hoort bij een afspraak niet alleen wat mogelijk is, maar ook wat er mis kan gaan en hoe waarschijnlijk verschillende uitkomsten zijn. Niet omdat ambitie verkeerd is, maar omdat anderen hun keuzes baseren op wat wij zeggen. Uiteindelijk draait het in projecten niet alleen om plannings of beloftes, maar om vertrouwen. Zodra anderen hun planning, hun budget of hun verwachtingen baseren op onze woorden, ontstaat er afhankelijkheid.

Als er iets niet goed gaat, is het daarom vaak het verstandigst om dat zo snel mogelijk te delen. Niet omdat anderen onderdeel zijn van het probleem, maar omdat ze juist onderdeel van de oplossing kunnen worden.

Het kan goed komen. Maar ook zomaar niet.

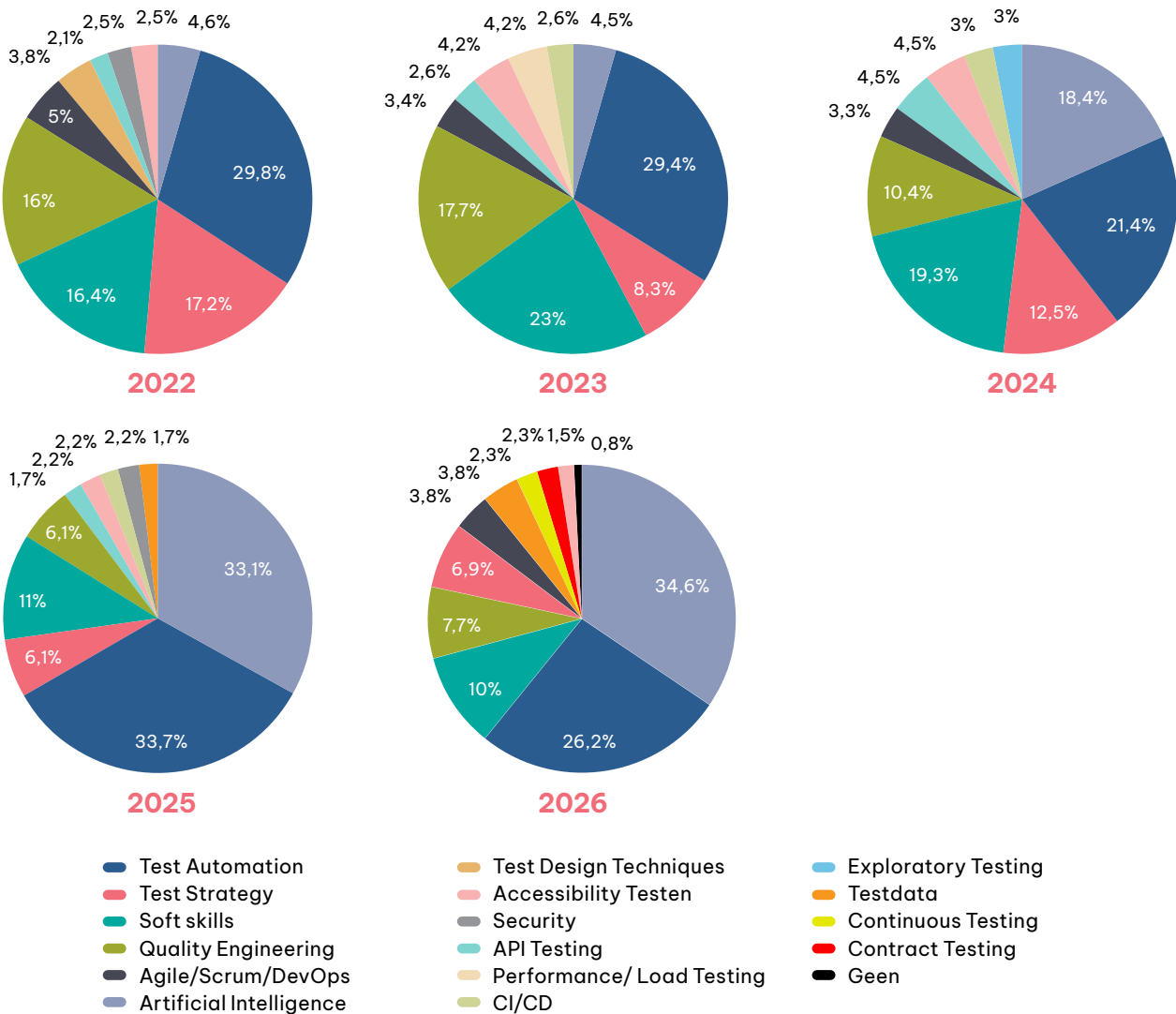
Onze onderzoeksoopzet

Voor de totstandkoming van dit trendrapport hebben we uitgebreid onderzoek gedaan naar de belangrijkste trends en ontwikkelingen in het testvak. Dit hebben we gedaan door systematisch te kijken naar waar de discussie in ons vakgebied écht plaatsvindt.

Concreet hebben we:

- De thema's en onderwerpen geanalyseerd die centraal stonden op toonaangevende test gerelateerde conferenties in Europa van 2023 tot en met 2026, zoals o.a. Agile Testing days, AutomationSTAR, EuroSTAR, ExpoQA, HUSTEF, Nordic Testing Days, Peerscon, Robocon, Test Automation Days, Testbash, Testing United, Testnet en WeTestAthens.
- Inzichten opgehaald uit trainingen, kennissessies en gesprekken met klanten en partners.
- De kennis en ervaringen van onze eigen consultants meegenomen, die als deelnemer én als spreker aanwezig waren op de internationale events.
- Onze collega's gevraagd om input. Al deze informatie hebben we samengebracht en geclusterd in duidelijke categorieën. Daarbij hebben we steeds gekeken naar de inhoudelijke samenhang en relevantie voor de praktijk van vandaag én morgen.

Onderwerpen test gerelateerde conferenties 2022-2026



Hoe worden de onderwerpen op conferenties bepaald?

Programma's van conferenties worden doorgaans bepaald door een comité van verschillende professionals, vaak op basis van een (actueel) thema. Een organisatie van een conferentie kiest de sprekers onder andere op basis van kwaliteit, onderwerp en inhoud van de paper waarin staat omschreven wat de presentatie of workshop inhoudt. Sprekers die een presentatie willen geven, kiezen hun onderwerp op basis van hun interesse, actuele ontwikkelingen, onderzoek of praktijkervaringen.

Human Intelligence als basis voor ons trendrapport

Programma's van conferenties worden doorgaans bepaald door een comité van verschillende professionals, vaak op basis van een (actueel) thema. Een organisatie van een conferentie kiest de sprekers onder andere op basis van kwaliteit, onderwerp en inhoud van de paper waarin staat omschreven wat de presentatie of workshop inhoudt. Sprekers die een presentatie willen geven, kiezen hun onderwerp op basis van hun interesse, actuele ontwikkelingen, onderzoek of praktijkervaringen.

Onze expertise

Sofius biedt het volledige spectrum aan diensten voor het verbeteren van softwarekwaliteit. Hierbij combineren we de inzet van gedreven consultants met state-of-the-art kennis van software, systemen, domeinen en markten.

Artificial Intelligence

Bij Sofius staat AI centraal in onze innovatie. Generatieve AI verandert softwareontwikkeling en kan code genereren die overtuigend oogt, maar verborgen risico's bevat. Daardoor is testen belangrijker dan ooit. Tegelijk zien wij AI als een versneller. Onze testconsultants passen AI kritisch, bewust en veilig toe, zodat klanten profiteren van efficiëntere, betrouwbare en toekomstbestendige softwareoplossingen.

Testautomatisering

Testautomatisering draagt bij aan het versneld ontwikkelen en opleveren van waardevolle software. Sofius heeft een krachtige visie, een bewezen aanpak en specialisten die test-automatisering succesvol kunnen uitrollen in Agile/DevOps omgevingen. Waarbij we tooling gebruiken die het best aansluit bij de klantbehoefte.

Testanalyse en -uitvoer

Onze testanalisten hebben een afgeronde hbo- of wo opleiding en zijn uitgebreid getraind en gecertificeerd in het testvak. Het 'echte' testwerk is geen handwerk, maar denkwerk. Daarbij zetten we onze kennis, creativiteit en ervaring met tools en technieken in.

Testmanagement

Onze testmanagers hebben uitgebreide kennis van diverse testmethodes, tooling en domeinen. Zij zijn uitstekend op de hoogte van de laatste stand van zaken, zorgen voor een stevige kennisimpuls en zijn thuis in meerder ontwikkelomgevingen. Zowel in (scaling) Agile/DevOps omgeving als in de meer traditionele waternal omgeving.

Coaching en advies

Onze coaches en adviseurs zijn nauw betrokken bij de operatie en beschikken over ruime ervaring binnen hun expertisegebied. Ze werken intensief samen binnen het team en begeleiden het volledige proces, zodat de organisatie zelfstandig verder kan.

Enterprise Software Testen

Het testen van (standaard) enterprise softwareoplossingen vraagt meer dan alleen testexpertise. Het neerzetten van een risk-based aanpak, het mobiliseren van business gebruikers en het valideren van inrichtingskeuzes vereist pakketkennis én pragmatiek. Sofius heeft veel ervaring als testpartner bij ERP implementaties en upgrades.

Low Code Testen

Low code is erop gericht om met minder programmeerwerk en met minder specialistische programmeerkennis applicaties te bouwen. Nadruk bij deze methode ligt op het configureren en samenvoegen van bouwstenen om de gewenste functionaliteit te verkrijgen. Sofius biedt gespecialiseerde low-code testdiensten om de kwaliteit van applicaties te garanderen. Onze experts gebruiken toonaangevende tools en methodes voor efficiënte en betrouwbare tests en snelle feedback loops.

Quality Improvement Services

Een digitale transitie betreft het optimaliseren van de manier waarop een organisatie technologieën en digitale middelen gebruikt. Zodat mensen, processen en producten, zich kunnen blijven aanpassen aan veranderende behoeften van de klant, de markt en de organisatie zelf. Onze mensen hebben ruime ervaring met deze ingrijpende trajecten. Samen brengen we jouw organisatie verder.

Wij leveren experts in diverse rollen.

Van testautomatiseerder tot Agile tester en van testengineer tot testmanager en testcoach.

Met welke trend is jouw interesse gewekt?

Wil je weten hoe je deze trends kunt inzetten om de kwaliteit van IT en software binnen jouw organisatie te bewaken? Heeft dit rapport vragen bij je opgeroepen over de eerste stappen richting implementatie, of ben je benieuwd hoe wij jouw organisatie verder kunnen helpen?

Neem dan contact op met Sofius.

www.sofius.com
Info@sofius.com

Powerhouse in software testen en kwaliteitsverbetering

Sofius is dé partner voor het verbeteren van softwarekwaliteit. Met onze 6 labels en meer dan 300 professionals werken wij elke dag aan het succes van onze klanten. Samen brengen we mensen, bedrijven en softwaresystemen verder.

We bieden het volledige spectrum aan diensten voor het verbeteren van softwarekwaliteit. Van testanalyse en -uitvoer tot testautomatisering en testmanagement. En van coaching & advies tot aan Enterprise software testen en low code testen.

Hierbij combineren we de inzet van gedreven consultants met state-of-the-art kennis van markten, domeinen, software en systemen.

sofius

bartosz

TESTERS

PTWEE®

SQUERIST »



TESTCODERS



TechChamps