servicetrace
SOFTWARE ROBOTICS

DER SOFTWARE QUALITY GUIDE.

Software testen, Tests automatisieren.

WARUM SIE SOFTWARE TESTEN SOLLTEN

„Wir beschäftigen genauso viele Tester wie Entwickler. Wenn wir ein neues Windows-Release vorbereiten, geht über die Hälfte des Budgets allein in die Qualitätskontrolle“, erklärte Bill Gates der InformationWeek in einem Interview von 2002. Wenn CEOs, Projektmanager und IT-Verantwortliche mit knappen Budgets haushalten müssen, rutscht das Thema Softwaretesten genau aus diesem Grund auf der Agenda weit nach unten und wird mit Einwänden wie „Stehen die Testkosten in einer vernünftigen Relation zu den möglichen Auswirkungen möglicher Softwarefehler? Lohnt sich der Aufwand?“ schnell ad acta gelegt. Das gesamte Budget für Softwareentwicklung wird lieber in die Programmierung gesteckt, da Qualitätssicherung keine neue Software an die Kunden bringt.

Risiko Softwarefehler

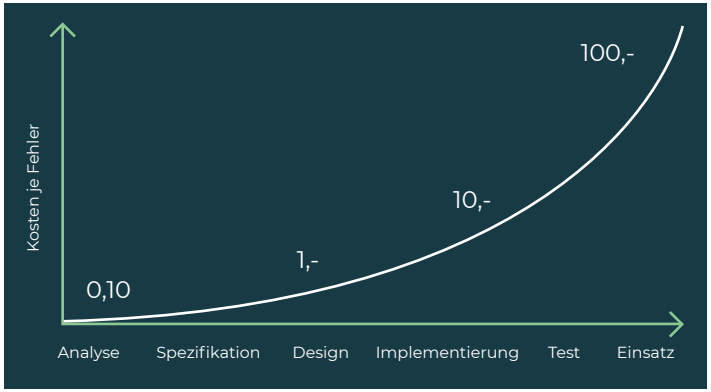
Der Berliner Hochschulprofessor Edzard Höfig unterhält einen Youtube-Kanal mit empfehlenswerten Online-Vorlesungen rund um den Themenkomplex Softwareentwicklung. Im Webcast Einführung in das Softwaretesten erklärt er eindrucksvoll, warum Softwaretests so wichtig sind: Er erzählt die Geschichte vom Jungfernflug der Trägerrakete Ariane 5, einem teuren Prestigeprojekt der ESA, das die technologische Führungsposition der europäischen Raumfahrtorganisation sichern und zahlreiche Investoren anlocken sollte.

Ariane 5 war technologisch ein kompletter Neubau und beschleunigte deutlich schneller als Ariane 4. Die Software der Ariane 5 war aber keine Neuentwicklung, sondern baute auf dem kompletten Code der Vorgängerversion für Ariane 4 auf. Ein Codeschnipsel im Startsystem, der bei Ariane 4 ein Problem beim Start behoben hatte und für die Startfähigkeit der Ariane 5 gar keine Rolle mehr spielte, konnte die hohen Beschleunigungsmesswerte nicht korrekt interpretieren und sendete falsche Informationen an die Steuerungslogik. Die Rakete kam deutlich vom geplanten Kurs ab, was den Selbstzerstörungsmechanismus auslöste.

Mit der (unbemannten) Ariane verbrannte am 4. Juni 1996 Material im Wert von rund 370.000.000 US-Dollar. Der Schnipsel Programmcode, der die Kettenreaktion auslöste, die zur Explosion der Rakete führte, ist als einer der teuersten Softwarefehler in die Geschichte eingegangen.

Was hier in großem Maßstab skizziert wird, gilt auch für kleiner dimensionierte Softwareprojekte. Die Botschaft ist klar: Bevor neue oder veränderte Systeme in Betrieb genommen werden, müssen sie eine ausreichende Testphase erfolgreich durchlaufen haben. Unternehmen, die auf Softwaretests verzichten, sparen nur kurzfristig und riskieren viel.

Die Fehlerkosten schlechter Software übersteigen regelmäßig die Fehlerverhütungskosten, die für die Qualitätssicherung entstanden wären. Für die Kostenentwicklung im Software Lifecycle gilt das „Rule of 10“ aus der industriellen Fertigung, nach dem sich die Fehlerkosten eines Produkts mit jeder Projektphase verzehnfachen.



Die 10er-Regel der Fehlerkosten in der Softwareentwicklung

Was ist Softwarequalität?

Die Kundensicht liefert eine pragmatische Definition:

- Die Software soll die Kundenanforderungen erfüllen und Arbeitsabläufe optimal unterstützen
- Die Software soll fehlerfrei laufen und keine Störungen im Betrieb verursachen.

Daraus entstehen diese Prioritäten für die Entwicklung hochwertiger Software:

- Fokus auf klar und vollständig definierten Requirements
- Fokus auf Fehlersuche vor Inbetriebnahme

Daraus leiten sich diese Maßnahmen ab:

- Enge und wiederholte Abstimmung zwischen Kunden, Product Owner und Entwicklern
- Testen in den Entwicklungsprozess integrieren

Fehlerwirkung und Fehlerbehebung

Die Kosten für Softwarefehler wirken direkt und indirekt. Direkt entstehen Kosten für das Finden der Fehlerursache (Debugging) und die Fehlerbehebung (Bugfixing). Gut bezahlte Softwareentwickler verbringen durchschnittlich die Hälfte ihrer Arbeitszeit mit ungeliebter und unproduktiver Fehlerbehebung, Zeit, die für die kreative Weiterentwicklung der Software fehlt.

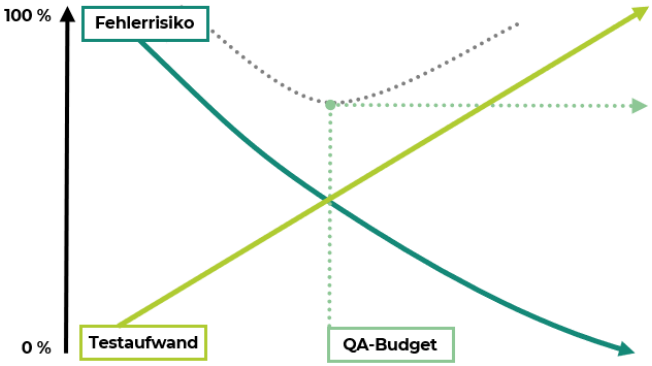
Indirekte und unkalkulierbar hohe Kosten entstehen durch die möglichen Fehlerwirkungen, wenn Softwarefehler in die Produktion gelangt sind. Sie erstrecken sich über Ausfälle produktiver Abläufe, Kundenverlust, sinkende Reputation und Vertragsstrafen.

Aber nicht nur der wirtschaftliche Schaden durch die enormen Fehlerkosten hängt Unternehmen so mittelfristig vom Wettbewerb ab, sondern besonders die verpasste Gelegenheit, hohe Softwarequalität als differenzierendes Anbietermerkmal zu nutzen.

Testaufwand vs. Fehlerkosten

Eine wirtschaftlich vernünftige Investition in Qualitätssicherung berücksichtigt das Verhältnis von Testaufwand und Fehlerkosten. Die Abbildung verdeutlicht, dass mit steigendem Testaufwand das Risiko für potenzielle Fehlerkosten sinkt. Ab einem bestimmten Punkt übersteigen die Testkosten die möglichen Fehlerkosten. Am Tiefpunkt der Parabel über diesem Schnittpunkt siedeln die Werte für Ihr optimales Qualitätsbudget.

Optimaler Aufwand für Software-QA



Testkosten	Fehlerkosten
€ 300.000	€ 0
€ 270.000	€ 100.000
€ 240.000	€ 200.000
€ 210.000	€ 300.000
€ 180.000	€ 400.000
€ 150.000	€ 500.000
€ 120.000	€ 600.000
€ 90.000	€ 700.000
€ 60.000	€ 800.000
€ 30.000	€ 900.000
€ 0	€ 1000.000

Das Beispiel ermittelt bei maximalen Testkosten von 300.000 Euro und maximalen angenommenen Fehlerkosten von 1000.000 Euro einen Wert zwischen 210.000 und 240.000 Euro, der in die Qualitätssicherung einfließen sollte.

Bauen Sie das Thema Softwarequalität fest in Ihre Digitalstrategie ein und investieren Sie Ihr Entwicklungsbudget vermehrt in Softwaretests. Qualität setzt sich durch und zahlt sich aus:

- Sie positionieren sich unter zahlreichen konkurrierenden Anbietern als herausragender Qualitätshersteller
- Sie etablieren sich mit hochwertigen, langlebigen Softwareprodukten als zuverlässige Digital-Manufaktur langfristig am Markt

WARUM SIE SOFTWARETESTS AUTOMATISIEREN SOLLTEN

Die beschleunigte Dynamik des globalen Marktgeschehens und die wachsenden Ansprüche der Verbraucher erzeugen besonders im schnelllebigen Digitalsektor einen hohen Innovationsdruck. Um im Wettbewerb mitzuhalten, müssen Unternehmen neue Softwareprodukte häufiger und schneller als die Konkurrenz in Betrieb nehmen. 2, 3 Releases im Jahr sind nicht mehr genug. Besonders wenn die Software selbst zum Geschäftsmodell wird, sind Updates in kürzeren Abständen von Quartalen oder Wochen erforderlich. Digitale Big Player wie z.B. bei Google, Facebook oder Tictoc releasen mehrmals täglich.

Agile Softwareentwicklung

Der Trend zu immer schnelleren Release-Zyklen mit immer häufigeren Updates passt sich den Anforderungen des globalisierten und digitalisierten 21. Jahrhunderts an, in dem Konsumenten, Anbieter und Produkte spontan, flüchtig und unberechenbar interagieren (vgl. Infobox VUCA). Langfristige Planung funktioniert in dieser kurzlebigen Welt nicht mehr: Das hat zuerst in der Softwareentwicklung seit den frühen 2000ern agilen Frameworks und Methoden wie Scrum, Kanban und Lean Management Tür und Tor geöffnet.

Agile Entwicklung (lat. agilis: „flink, beweglich“) verabschiedet sich von einem linearen Entwicklungsprozess, bei dem zu Projektbeginn gemeinsam mit dem Auftraggeber alle Anforderungen an die Software spezifiziert werden, die Software dann in einer mehrmonatigen Entwicklungsphase mit allen neuen Funktionen und Eigenschaften implementiert und anschließend in einem „Big Bang“ released wird. Dieser traditionelle Weg des Software Engineerings führt oft zu enttäuschenden Fehlentwicklungen.

Traditionelle Softwareentwicklung: planmäßig scheitern?

Softwaresysteme müssen heute derart komplexe und dynamische Businessumgebungen unterstützen, dass die Anforderungen an das Produkt eingangs noch gar nicht vollständig, sicher und eindeutig definiert werden *können*. Das heißt: Im Projektverlauf werden sich Anforderungen durchaus ändern oder neue hinzukommen. Wird die Software jetzt weiter planmäßig „im Silo“ entwickelt, entsteht am Ende: teure Software für die Tonne.

VUCA beschreibt die veränderten Rahmenbedingungen und Herausforderungen für Unternehmen im digitalen Zeitalter. Konsumenten binden sich nicht länger dauerhaft an Marken oder Unternehmen, sondern wechseln spontan zu Anbietern mit Produkten, die ihren **veränderlichen** (volatile) Bedarfen entsprechen. Künftige Marktentwicklungen und Kundenanforderungen sind jederzeit **unsicher** (uncertain): Unsere **komplexe** (complex) und **ambivalente** (ambigue) Welt mit global und digital vernetzten Wirtschaftskreisläufen und allgegenwärtiger Multioptionalität erweist sich als unberechenbar dynamisch und bietet keinen Raum mehr für Planungssicherheit.

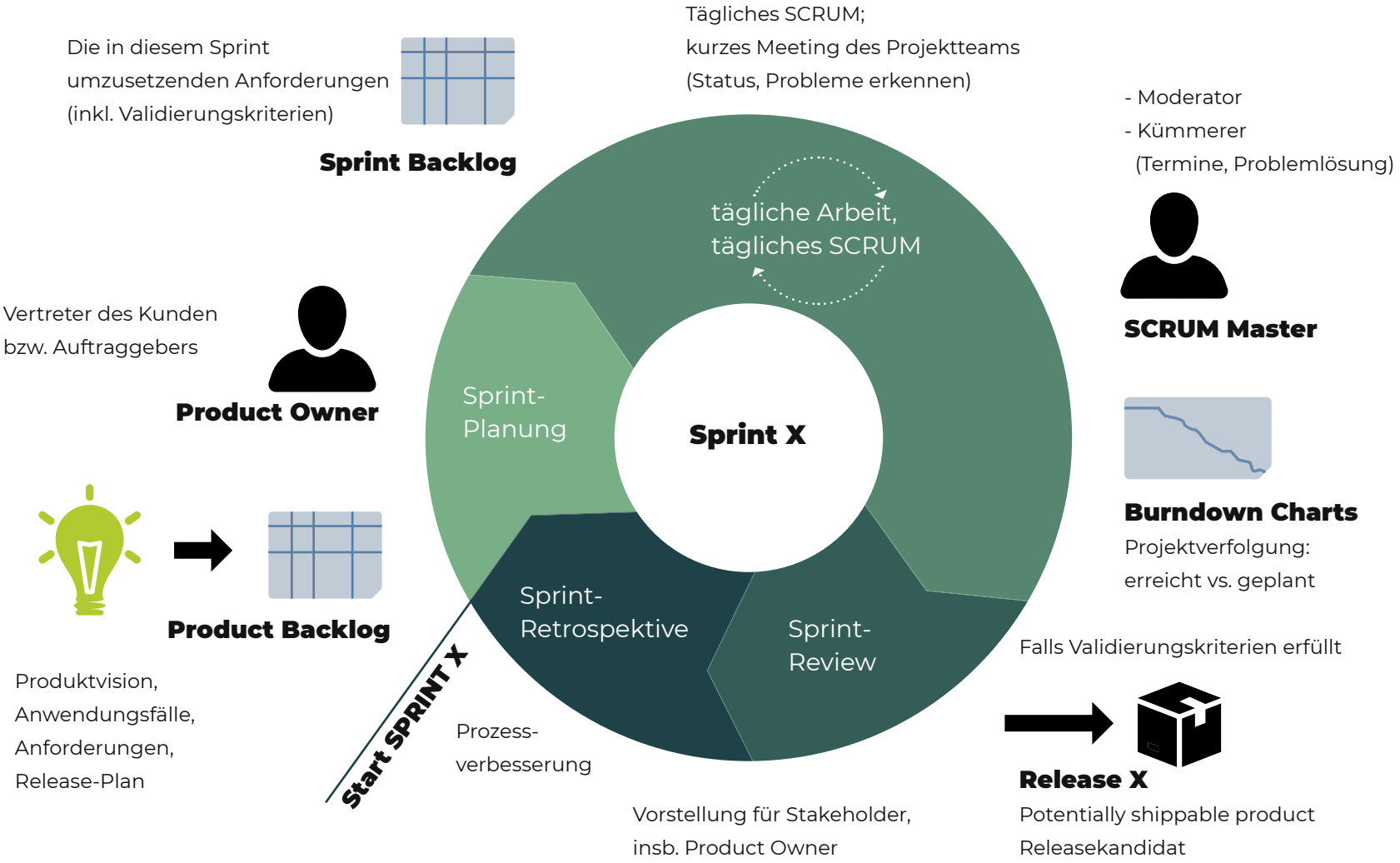
Das bedeutet das Ende linearen Denkens und linearer Strategien – und ebnet den Weg für agile Methoden.

Um die Software zu erzeugen, die der Kunde wirklich brauchen wird, geht agile Softwareentwicklung nach dem Scrum-Framework nicht mehr linear, sondern iterativ (= sich schrittweise der Lösung annähernd) und inkrementell (= schrittweise vorgehend, aufeinander aufbauend) vor.

Scrum: in Teiletappen zur brauchbaren Software

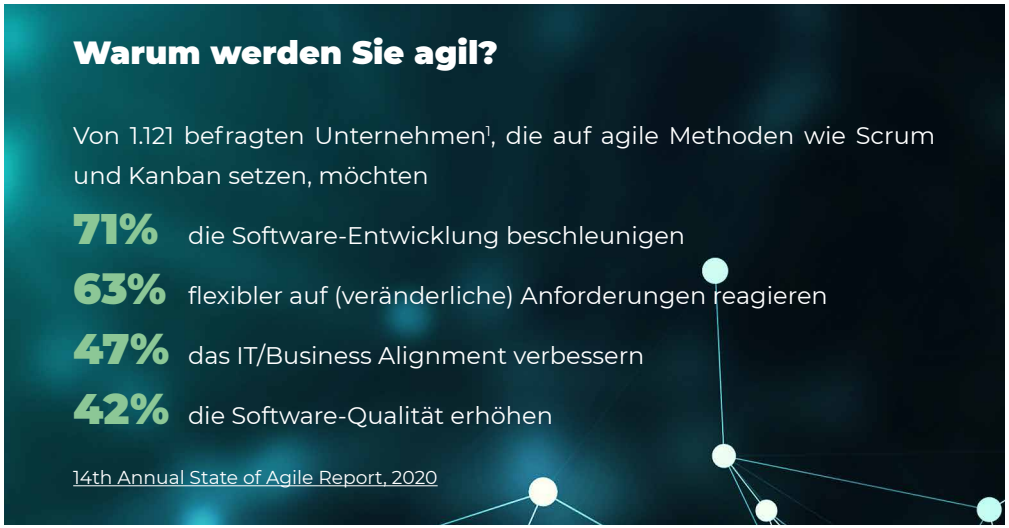
Zu Projektstart formuliert der Auftraggeber die gewünschten Produkteigenschaften als „User Stories“, die der Product Owner im Backlog listet und priorisiert. Das Entwicklerteam implementiert die neuen Funktionalitäten nach und nach in ein- bis vierwöchigen „Sprints“. Am Ende jedes Sprints steht ein gebrauchsfertiges Teilprodukt, das der Kunde erneut validiert. Nach den „Sprint Reviews“ können in weiteren Absprachen zwischen Product Owner und Auftraggeber die Anforderungen an die Software nachjustiert werden.

Agile Softwareentwicklung kann durch den modularisierten Entwicklungsprozess, die rapiden Releasezyklen und die kontinuierlichen Feedbackschleifen mit dem Auftraggeber flexibel auf sich schnell verändernde Kundenwünsche reagieren und so das Risiko von Fehlentwicklungen vorbei an den Kundenanforderungen minimieren.



Agil trendet

Funktionierende Software ist wichtiger als umfassende Dokumentation, kontinuierliche Kommunikation mit dem Kunden rangiert vor abstrakten Vertragsverhandlungen, und Reagieren auf Veränderungen zählt mehr als das starre Befolgen eines Plans – die neuen Prinzipien für Softwareentwicklung, die das Agile Manifest 2001 in einem revolutionären Gestus formulierte, sind heute Goldstandard für Organisationen, die ihr Innovationstempo beschleunigen und dabei nah am Kunden agieren möchten.



Deutlich wird bei den Zahlen in der Infobox zweierlei:

- Die meisten Unternehmen priorisieren bei der Einführung agiler Methoden den Faktor Geschwindigkeit, dicht gefolgt vom Faktor Flexibilität
- Immerhin knapp 60% schenken dem Aspekt Software-Qualität keine weitgehende Beachtung

Softwaretest: ein teures Bottleneck?

Ein Grund für die Zurückhaltung sind die schon beschriebenen Kosten, die für die Qualitätssicherung anfallen. Zum Gesamtbudget eines Softwareprojekts tragen die Aufwände für einen reifen Testprozess mit rund 30-50% durchschnittlich bei.

Der zweite Grund ist: Software testen kostet Zeit. Der wichtige End-to-End-Streckentest etwa, der prüfen soll, ob der Kunde mit dem Softwaresystem geschäftskritische Abläufe wie z.B. das Abschließen eines Vertrags oder eine Produktbestellung erfolgreich durchführen kann, kann bei einer komplexen Anwendung in einem großen Systemverbund schnell hunderte von Testfällen mit tausenden Teilschritten enthalten. Der damit verbundene enorme Zeitaufwand für das Design, die Durchführung und die Auswertung der Tests bremst agile Teams und Prozesse aus.

In Gartner's 2018 Agile Survey, 44% of respondents identified working with teams using traditional methods as one of their top-three challenges for adopting agile development. Testing is a common constraint in this situation due to its … reliance on manual efforts; as a result, quality is slow to achieve and ineffective.[1]

Bei ohnehin knappen Ressourcen und unter Innovationsdruck wird das Budget häufig fast vollständig von der Risikoprävention auf die Weiterentwicklung verlagert.

As the pace of delivery increases … manual testing can't keep up and quickly becomes a bottleneck. As a result, testing often gets sacrificed and quality deteriorates … .[2]

Was passiert, wenn Unternehmen auf Softwaretests verzichten, weil diese das digitale Innovationstempo zu drosseln drohen, wurde im Abschnitt *Warum sollte ich Software testen?* beschrieben. Nur gute Softwarequalität setzt sich nachhaltig durch und steigert die Wettbewerbsfähigkeit – ist aber ohne Tests nicht zu haben.

[1] Christopher Little, Jim Scheibmeir: When Dev goes agile, optimize operations' testing capabilities with automation, Gartner Research 2018

[2] Joachim Herschmann, Thomas Murphy, Jim Scheibmeir: 4 Essential Steps to implement Test Automation, Gartner Research 2019

Mit Automatisierung schneller, günstiger und besser testen

Eher behäbig Qualität liefern – oder schnell Mangelware in Umlauf bringen? Wenn manuelle Abläufe im Testprozess automatisiert werden, müssen sich Softwareprojektveranwortliche nicht mehr so rigoros zwischen Softwarequalität und Innovationstempo entscheiden. Mit einer durchdachten Testautomatisierungsstrategie kann die Qualitätssicherung nicht nur das agile Tempo der Softwareentwicklung aufnehmen; die automatisierten Softwaretests erhöhen zudem die Produktqualität und senken die Testkosten.

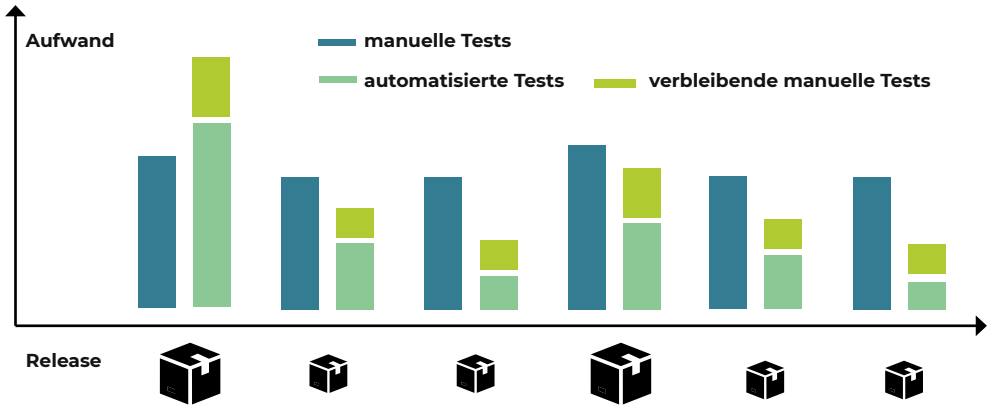
Mehr Agilität	
• beschleunigte Testdurchführung	• häufiges und schnelles Feedback der Testergebnisse an die Entwicklung
• häufigere Testdurchführung	• schnellere Fehlerbehebung und Weiterentwicklung
Verbesserte Qualität	
• erhöhte Testabdeckung	• geringeres Risiko des Nicht-Entdeckens von Fehlern
• standardisierte Testausführung	• durch Menschen verursachte Fehler sind ausgeschlossen
Sinkende Kosten	
• weniger Softwarefehler	• weniger Fehlerkosten
• geringere Kosten für Testdurchführung	• Laufende Automatisierung kostet nach Return on Investment den Bruchteil eines durchschnittlichen Software-Tester-Gehalts
Effizienter Ressourceneinsatz	
• Befreit qualifizierte Fachtester von zeitraubenden, langweiligen und fehlernfälligen Routinetests	• Setzt Ressourcen frei für kreative und komplexe Aufgaben (Usability Tests, explorative Tests) oder Neuqualifikationen (> Test Automation Engineer)

Testautomatisierung: Invest und RoI

Naiv wäre allerdings die Vorstellung von Testautomatisierung als sofort rentabler magischer Praktik, die sich wie von Zauberhand nebenbei erledigt. Die Umstellung von manuell auf automatisch erfordert einen initialen Aufwand, der den der manuellen Tests anfangs übersteigt. Dazu gehört:

- Überzeugung und Commitment von Test-Team, Unternehmensleitung, IT-Leitung, Einkauf
- Auswahl zu automatisierender Teststufen und -fälle
- Vergleich und Auswahl von Automatisierungstools
- Schulung der Mitarbeiter
- Definition (neuer) Prozesse, Rollen und Aufgaben im Test-Team
- Erstellen und Testen der automatisierten Testfälle

Der initiale Aufwand rentiert sich dann aber umso schneller und besser, je häufiger Sie Software-Updates releasen und je länger Sie die Software weiterentwickeln.



Mit jedem Release sinkt der Aufwand für die Testautomatisierung. da die einmal automatisierten Tests beim nächsten Minor Release einfach wiederverwendet werden und beim nächsten Major Release nur um die neuen Testfälle erweitert werden müssen. Der Aufwand für manuelle Tests wiederholt sich dagegen mit jedem Release und erhöht sich sogar bei zunehmender Komplexität der Anwendung.

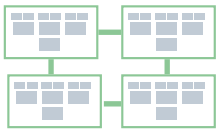
ÜBERSICHT TESTSTUFEN

Aufwand hoch

Aufwand gering

Business-Sicht

System-Sicht

Teststufe	Schicht	Ziel	Durchführung	Tools (z.B.)
Abnahmetest Abnahme durch den Kunden / Auftraggeber 	GUI	Validierung und Abnahme des fertigen Softwaresystems durch den Auftraggeber	Kunde, Fachanwender, Product Owner	Manuelle Tests
Systemintegrationstest Testet die Interoperabilität mit weiteren Systemen 	GUI	Systemverbund auf fehlerfreie Durchführbarkeit von Businessprozessen mit systemübergreifenden E2E-Klickstrecken testen	Tester	Ranorex Servicetrace Tricentis
Systemtest Testet das voll integrierte System aus Anwendersicht 	GUI	System / Software gegen Vollständigkeit und Funktionalität definierter Systemanforderungen testen	Tester	Ranorex Servicetrace Tricentis
Integrationstest Testet das Zusammenspiel von mindestens zwei Komponenten 	Schnittstellen / APIs	Funktionalität integrierter Komponenten testen. Inkrementelles Vorgehen, bis alle Komponenten als komplettes System integriert sind	Entwickler, Tester	Citrus
Komponententest / Unittest Testet einzelne Softwarebausteine sofort nach Implementierung 	Code	Stabilität und Funktionalität atomarer und isolierter Softwareeinheiten testen. Fehler können einfach und schnell entsprechenden Zeilen im Code zugeordnet und behoben werden	Entwickler	JUnit MSUnit Pytest

Diese Übersicht bündelt die für eine durchgehende Qualitätssicherung in der Softwareentwicklung essenziellen Teststufen. Erst wenn ein Built eine Teststufe auf einer niedrigeren Hierarchie erfolgreich durchlaufen hat, wird er in die nächste Teststufe aufgenommen. Von den niedrigen bis zu den höheren Teststufen steigern sich Kosten- und Zeitaufwand.

Automatisierte Testläufe sind bis auf den Abnahmetest durch den Kunden auf allen Teststufen möglich. Ein hoher Automatisierungsgrad im gesamten Testprozess erzielt a) Zeitersparnis durch beschleunigte Testdurchlaufzeiten und schnelles Feedback an die Entwicklung, b) höhere Softwarequalität durch erhöhte Testabdeckung.

IN 8 SCHRITTEN ZUM TESTAUTOMATISIERUNGS-FRAMEWORK

Der letzte Abschnitt behandelt Best Practices für eine optimal effiziente und rentable Einführung einer Testautomatisierung.

1) Basics: Hier lohnt sich Automatisierung

Prüfen Sie zuerst anhand dieser 4 Faktoren, welche Tests Sie automatisieren sollten:

- Application Lifecycle: Automatisieren Sie Tests für langlebige Anwendungen, die noch viele Releases erwarten lassen.
- Testhäufigkeit: Automatisieren Sie Tests, die wiederholt bei jedem Release anfallen.
- Stabilität: Solange ein definiertes Feature noch entwickelt wird und häufigen Modifikationen unterliegt, sollte manuell getestet werden, da die Automatisierung hier mit einem zu hohen Wartungsaufwand verbunden ist. Automatisieren Sie dann, wenn das Feature stabil implementiert ist.
- Geschäftsrisiko: Priorisieren Sie die Test Cases anhand einer Risikoanalyse und automatisieren Sie Tests für die Features, deren Fehlfunktion die höchsten negativen Auswirkungen für Businessprozesse erzeugen würde.

2) Quick-win: Automatisieren Sie Regressionstests

Regressionstests werden auf jeder Teststufe durchgeführt. Sie prüfen, ob ein Test nach einer Änderung an der Software bzw. im Softwaresystem wiederholt erfolgreich durchlaufen werden kann, oder ob sich durch den Change ungewünschte Seiteneffekte ergeben haben.

Die Häufigkeit der Ausführung, der repetitive und monotone Charakter der Tests und die einfache Wiederverwendung der Testfälle machen Regressionstests zum Top-Automatisierungskandidaten mit hohem Einsparpotenzial.

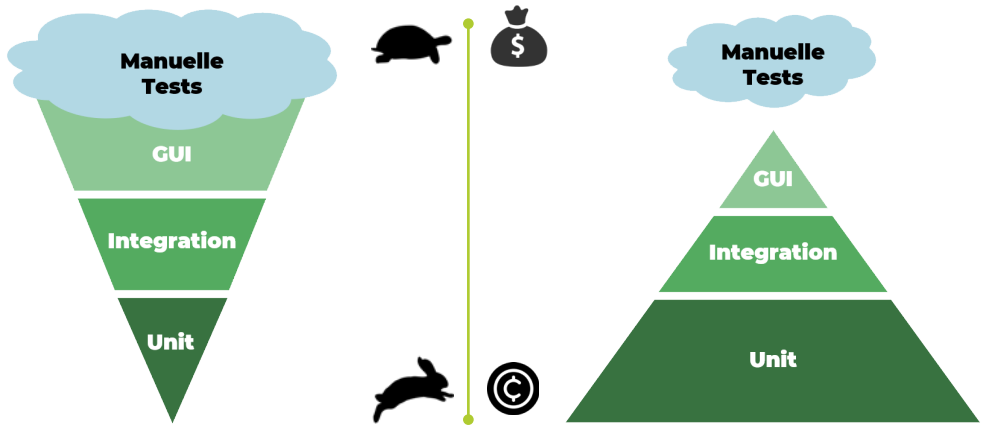
3) Ausbauen: Identifizieren Sie weitere Testfälle für die Automatisierung

Folgende Testtypen eignen sich aufgrund ihrer Häufigkeit und ihres Wiederholungscharakters außerdem für die Automatisierung:

- Smoke Tests stellen grundsätzliche Lauffähigkeit der Anwendung sicher (z.B. Anwendung starten, Login, ausgewählte Schlüsselfunktionalitäten) und sollten vor allen weiteren Testaktivitäten durchgeführt werden
- Datengetriebene Tests durchlaufen den gleichen Testfall mit verschiedenen Datensätzen (z.B. Username, Kundennummer, E-Mail, Zahlungsart etc.)
- Cross-Browser-Tests durchlaufen den gleichen Testfall in verschiedenen gängigen Browsern (Chrome, Firefox, Edge, Safari)
- Nicht-funktionale Performance- und Lasttests messen die Reaktionszeit der Software bei einfachem und parallelem Zugriff

4) Shift-left: Testen Sie früh, testen Sie viel

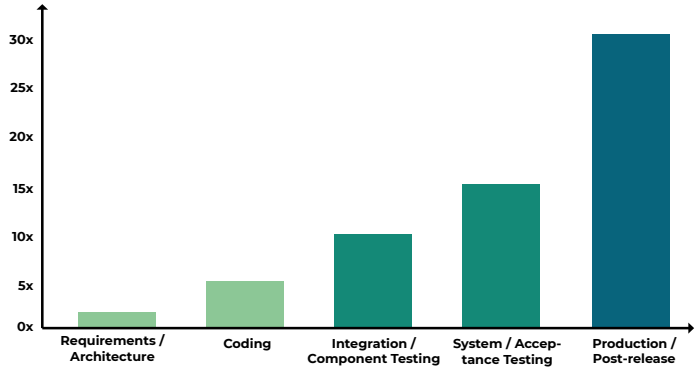
Einschlägige Orientierung für einen empfehlenswerten Automatisierungsaufwand nach Teststufen bietet die Test Automation Pyramid nach Mike Cohn.



Teststrategien: Ice Cream Cone Pattern vs. Test Automation Pyramide

Die üblicherweise anzutreffende Automatisierungs-Praxis in Softwareprojekten ist der Top-Down-Ansatz des „Ice Cream Cone"-Patterns: Der höchste Automatisierungs-Aufwand wird in GUI-Tests gesteckt mit dem Ziel, Softwarefehler am Ende der Entwicklungsphase aufzuspüren. Diese Methode ist wenig erfolgsversprechend und rentabel:

- Erstellen und Pflege der automatisierten GUI-Tests kostet vergleichsweise viel Zeit und Aufwand
- Ein Fehler auf Systemebene liefert keinen Nachweis darüber, welche Stellen im Code betroffen sind. Finden und Beheben der Ursachen ist mühsam und langwierig
- Je später im Entwicklungsprozess ein Bug gefunden wird, desto teurer wird seine Beseitigung



Die Fehlerbehebungskosten steigen mit jeder Stufe im Entwicklungsprozess. Häufiges und frühes Testen senkt diese Kosten deutlich.

Erfolgversprechender ist ein Bottom-up-Ansatz bzw. eine pyramidenförmige Gewichtung des Automatisierungsaufwands entlang der Teststufen. Dem liegt die Prämisse zugrunde, dass Softwarequalität nicht erst am Ende des Entwicklungsprozesses auf Systemebene „ertestet" werden kann, sondern von Anfang an in das Produkt eingebaut werden muss. Dieses Paradigma führt zum „Shift left" in der Softwarequalitätssicherung: Die Tests beginnen nicht erst am Ende der Entwicklungsphase, sondern begleiten parallel den gesamten Entwicklungsprozess. Beim Test Driven Development (TDD) werden die Tests sogar schon vor dem zu testenden Softwarebaustein geschrieben. Das zwingt Programmierer dazu, konsequent und mit möglichst schlankem (cleanen) Code auf die geforderten Funktionalitäten bzw. auf die Testbarkeit des Prüflings hin zu entwickeln.

Die breite Basis der Testautomatisierungs-Pyramide bilden also möglichst viele Unit - oder Modultests. Die Vorteile:

- niedriger Aufwand für Erstellung und Pflege der Tests, schnelle Laufzeit
- Modultests erfolgen direkt in der Entwicklungsumgebung statt in einer aufwändigen Testumgebung, da keine Abhängigkeiten von anderen Systemkomponenten getestet werden
- die Prüflinge sind feingranular modularisiert und beziehen sich auf einen dedizierten Abschnitt im Code, so lassen die Testergebnisse direkt auf die Fehlerursache schließen
- Modultests validieren zuverlässig Funktionalität und Robustheit des Codes für eine Abnahme in die nächste Teststufe

Die nächstgrößere Menge an Tests bzw. Automatisierungsaufwand bilden die Integrationstests, die die Interoperabilität der einzelnen Module validieren. Nur die Prüflinge, die die automatisierten Integrationstests bestehen, werden in das Softwaresystem implementiert.

Anschließend können wenige ausgesuchte automatisierte GUI-Tests die wichtigsten businesskritischen E2E-Klickpfade auf Durchführbarkeit bzw. Regressionen testen, während manuell nur noch explorative, komplexe GUI-Tests am neuen System durchgeführt werden.

5) Shift-right: Testen Sie die Kundensicht vor und nach dem Go Live

E2E-Tests aus Anwendersicht bilden nach der Testpyramide die kleinste Einheit der zu automatisierenden Tests. Für die Fehlersuche und -behebung sind sie weniger geeignet, weil eine fehlschlagende Funktion sich auf potenziell tausende Codeabschnitte beziehen kann. Vielmehr dienen Sie als Akzeptanztests dem Zweck, die Durchführbarkeit kritischer Geschäftsprozesse aus Kundensicht zu validieren, also die praktische Tauglichkeit der Software nachzuweisen.

Automatisierte GUI-Tests sind verglichen mit Unit- oder Integrationstests aufwändig in der Erstellung und Wartung. Hier machen 3 Stellschrauben die Automatisierung effizienter und rentabler:

1. Sie wählen ein Tool, das GUI-Tests weitgehend vereinfacht (mehr dazu im Abschnitt GUI-Tests: Kriterien für die Toolwahl)
2. Sie nutzen die automatisierten E2E-Testfälle für funktionale Tests, Performance-Tests und Lasttests
3. Sie verwenden die automatisierten E2E-Testfälle nach dem Go Live des Release weiter und nutzen Sie für ein kontinuierliches E2E-Monitoring der wichtigsten Geschäftsprozesse aus Perspektive der Anwender

„Shift-right" bedeutet hier, die automatisierte Qualitätssicherung auch über die Entwicklungsphase hinaus im weiteren Lebenszyklus der Applikation anzuwenden.

6) GUI-Tests: Kriterien für die Toolwahl

GUI-Tests gelten als besonders aufwändig zu erstellen und schwierig zu warten. Hier listen wir einige typische Probleme, die bei der GUI-Testautomatisierung entstehen können – und wie Sie diese mit der Wahl eines geeigneten Tools minimieren.

Einfache und schnelle Testerstellung und -Anpassung

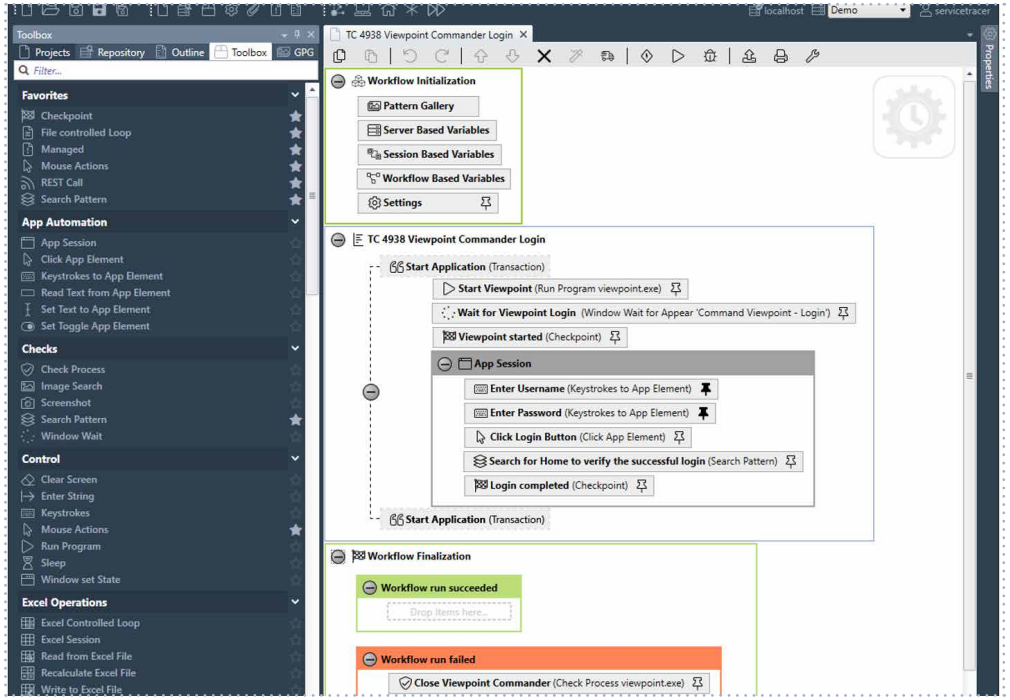
Weil GUI-Tests hunderte bis tausende von Testschritten enthalten und nach Implementieren neuer Funktionalitäten angepasst werden müssen, ergeben sich hier zwei wichtige Kriterien für die Toolwahl:

a) die Testerstellung muss möglichst einfach sein,

b) die Tests müssen schnell und punktuell anpassbar sein.

Für die einfache Testerstellung empfehlen sich No-Code-Lösungen, mit denen Testfälle aus vordefinierten Bausteinen in einem grafischen Editor zu einem strukturierten E2E-Workflow per Drag & Drop zusammengefügt werden. Das geht nicht nur schneller als Skripten, sondern ermöglicht auch programmierfernen Testern oder Fachanwendern aus dem Business, automatisierte E2E-Testfälle eigenhändig zu erstellen. Business User verstehen am besten die zu testenden Anforderungen bzw. Geschäftsabläufe in ihrem Fachbereich und bringen damit alle fachlichen Voraussetzungen für das Erstellen angemessener und hinreichender Testfälle für E2E-Akzeptanztests mit.

Capture & Replay-Ansätze sind zwar vordergründig bequemer und schneller als die Drag & Drop-Testerstellung in einem grafischen Editor. Der Nachteil ist allerdings der monolithische und unflexible Charakter aufgezeichneter Testfälle. Bei jeder Änderung der GUI müssen die Testfälle komplett neu aufgezeichnet, also noch einmal komplett manuell durchlaufen werden.



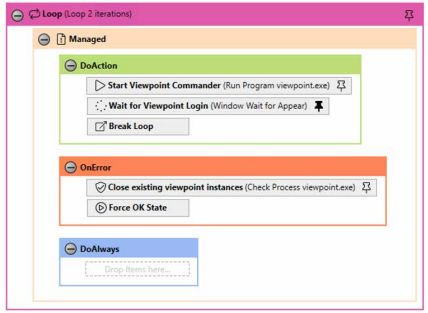
Mit No-Code-Lösungen GUI-Tests ohne Programmieren erstellen, z.B. mit dem Servicetrace Workflow Studio.

Mit Drag & Drop erstellte, modular und granular strukturierte grafische E2E-Workflows können dagegen gezielt nur an der von der Änderung der GUI betroffenen Sequenz angepasst werden. Ändert sich zum Beispiel das Icon zum Starten der Anwendung, muss nur die Bilderkennung für dieses Icon entsprechend neu konfiguriert werden.

Eine weitere Vereinfachung und Beschleunigung bietet die Option, Elemente, die in mehreren Workflows vorkommen, in einem zentralen Repository zu managen. Erforderliche Anpassungen eines Elements brauchen dann nur einmal zentral durchgeführt und auf sämtliche Automatisierungen ausgerollt werden.

Fehlerbehandlung

Lange E2E-Workflows bestehen in der Regel aus einer Sequenz aneinandergereihter Testfälle. Schlägt bei der Ausführung ein Testfall im Workflow fehl, können alle folgenden Schritte nicht ausgeführt werden und die Testautomatisierung bleibt bei der letzten fehlgeschlagenen Transaktion hängen. Damit E2E-Workflows auch im punktuellen Fehlerfall vollständig ausgeführt werden, sollte das Testautomatisierungs-Tool ein stabiles Error-Handling integrieren.



Fehlertolerant: integriertes Error-Handling erhöht die Stabilität langer E2E-Tests.

Servicetrace Test Automation löst die Fehlerbehandlung über so genannte Managed Blocks im Automatisierungs-Workflow: Hier wird das weitere Verhalten der Automatisierung im Falle einer fehlgeschlagenen Sequenz definiert. Tritt ein Fehler auf, wird dieser mit Screenshot und Analysepaket dokumentiert – und die nächste Sequenz im Workflow wird abgearbeitet.

Benachrichtigungen / Popups

Auch spontan auftretende Popups z.B. bei Updates weiterer in den Workflow integrierter Anwendungen können die Stabilität von GUI-Testautomatisierungen beeinträchtigen. Bei der Toolwahl sollten Sie also darauf achten, dass die Automatisierungslösung solche Ereignisse automatisiert abfangen und behandeln kann.

Umgang mit Wartezeiten

Manchmal reagiert das Testsystem langsamer als üblich – und schon läuft die Automatisierung in einen Fehler. Ihre Testautomatisierungs-Lösung sollte also konfigurierbare Wartezeiten für Ereignisse im Workflow ermöglichen, z.B. bis sich ein Browser-Fenster öffnet und einen bestimmten Inhalt anzeigt.

Dynamische Web-GUIs

Gerade bei Webanwendungen ändern sich GUI-Elemente oft oder werden je nach Nutzereinstellung anders angezeigt. Verfahren, die direkt auf der Nutzeroberfläche arbeiten, wie z.B. Bild- oder Mustersuche, sind in diesem Fall nicht geeignet.

Ihre Automatisierungslösung sollte also speziell für die Automatisierung dynamischer Web-Anwendungen weitere Verfahren wie die Steuerung von HTML-Objekten bereitstellen.

Abweichende Systemeinstellungen

Je nach Systemeinstellungen auf dem Endgerät können Auflösung, Farbtiefe oder Position der auf dem Desktop angezeigten Elemente variieren.

Für eine hohe Stabilität Ihrer GUI-Tests muss sich das Automatisierungs-Tool robust gegenüber abweichenden Desktop-Einstellungen verhalten.

Hardware- und Software-Ressourcen

Das Einrichten der Test-Umgebung erfordert in der Regel zusätzliche Hardware bzw. Software-Lizenzen. Sie können hier enorm Kosten sparen, wenn Ihr Automatisierungs-Tool parallele Tests auf einem Endgerät ermöglicht.

Toolkompatibilität

Vorteilhaft ist eine einfache Integrierbarkeit in gängige Test Management Tools wie z.B. Jira oder Jenkins.

7) Testautomatisierung und Teamwork: neue Aufgaben und Rollen

Wenn Sie planen, eine Testautomatisierung einzuführen, sollten Sie Ihren Mitarbeitern eventuell auftretende Ängste nehmen, durch Automatisierung ersetzt zu werden. Testautomatisierung kann und wird manuelles Testen nicht vollständig substituieren können, sie ist nicht dazu da, um Tester zu entlassen, sondern um sie zu entlasten.

Sie sollten also von Anfang an klarstellen, dass Testautomatisierung genau die lästigen, wiederkehrenden und langweiligen Aufgaben erledigt, die bei jedem Software-Release anfallen und jedes Test-Team bis über die Belastungsgrenze hinaus beanspruchen. Den Testern entsteht durch Automatisierung viel mehr Zeit für intellektuell anspruchsvolle, explorative Tests, die ihrer Qualifikation besser entsprechen, ihrer Arbeit mehr Sinn verleihen und so die Zufriedenheit im Job erhöhen.

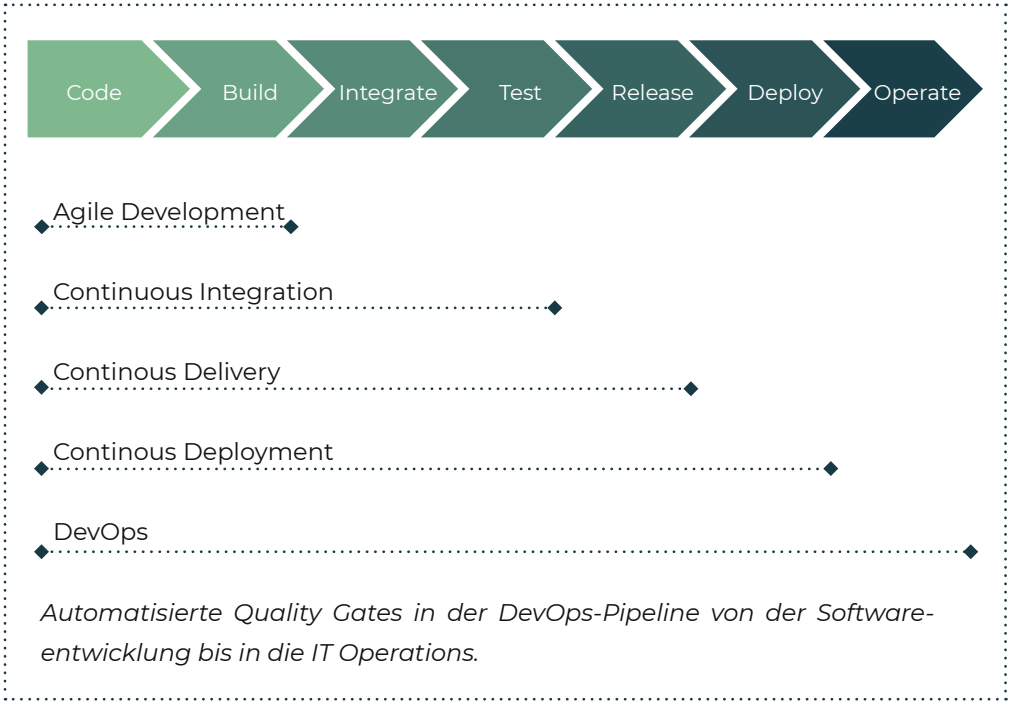
Zudem entstehen durch den neuen Arbeitsbereich Automatisierung neue Aufgabenfelder: etwa das des Test Automation Engineer, der sich vollumfänglich und in Vollzeit dem Thema widmet. Für bisher manuelle Tester ist hier auch eine Weiterqualifikation denkbar – entweder direkt beim Tool-Anbieter oder über eine Zertifizierung beim ISTQB als „Certified Tester Expert Level – Testautomatisierung".

8) DevOps: Continuous Delivery, Deployment, Operation

Die Testautomatisierung ist ein wichtiger Baustein einer umfassenden DevOps-Pipeline, in der ein neues Software Built automatisiert verschiedene Quality Gates passiert. Die Abnahme des Codes erfolgt über automatisierte Unit Tests z.B. mit JUnit, die Abnahme integrierter Module über Continuous Integration Tools wie z.B. Jenkins, die Abnahme komplett implementierter Systeme über GUI-Test-Tools wie z.B. Servicetrace Test Automation.

Der nächste Schritt in einer effizient automatisierten DevOps-Pipeline ist das automatisierte Deployment der Releases, die die vorigen automatisierten Tests erfolgreich bestanden haben. Mit der Hilfe von Tools wie Maven oder Docker kann die neue Softwareversion dann z.B. zu einem definierten Zeitpunkt automatisiert in die Produktivsysteme eingespielt werden.

Vervollständigt wird die automatisierte DevOps-Pipeline mit kontinuierlichen E2E-Tests (End-to-End-Monitoring) geschäftskritischer Klickstricken in der produktiven Umgebung.



Automatisierte Quality Gates in der DevOps-Pipeline von der Softwareentwicklung bis in die IT Operations.

Aus der Praxis:
Test Automation Kundenstories

British Heart Foundation setzt auf Performance-Tests

„Die Test-Automation-Lösung von Servicetrace erfüllte unsere anspruchsvollen Anforderungen an Volumen- und Transaktionstests am besten."
Sarah Yates | Project Manager

DSL- und Mobilfunk-Anbieter 1&1 skaliert mit Software Robotern

„Ohne Automatisierung wären solche regelmäßigen, umfassenden Regressionstests gar nicht möglich gewesen."
Andreas Förch | Leiter Testing Unit

Change? Aber sicher. Test Automation bei Dekra SE

„Das ist wirklich das einzige Tool, mit dem wir komplexe Tests in allen Applikationen durchführen können."
Katharina Hauch |
Leiterin Test Factory

servicetrace
SOFTWARE ROBOTICS

Servicetrace:
No-Code-Lösung für automatisierte GUI-Tests

Digitale Innovationen erfolgen in immer kürzeren Abständen – häufig im Intervall agiler Sprints. Das enorme Testvolumen, das dabei anfällt, können Sie zuverlässig und kostenschonend stemmen – mit Testautomatisierung von Servicetrace.

Qualität braucht Zeit – aber wenn die Zeit knapp wird, darf die Qualität nicht leiden. Deshalb setzen Tester-Teams auf Servicetrace Test Automation, wenn QA-Prozesse in der Software-Entwicklung beschleunigt werden müssen. Die Lösung eignet sich für ausnahmslos alle Anwendungen (web / non-web).

Kunden schätzen den No-Code-Ansatz der Lösung. Die Test Cases werden im Workflow Studio mit einem intuitiven grafischen Drag & Drop-Verfahren schnell und einfach erstellt und angepasst. So können sogar User aus den Fachbereichen Softwaretests eigenhändig automatisieren.

Servicetrace Test Automation bietet eine vollumfängliche Test Suite, in der Sie Epics, Stories und Test Cases managen. Die Test Cases werden in einem revisionssicheren Repository archiviert – auditsicher. Nach der Testausführung liefern die Testergebnisse mit detailliertem Analysepaket ein komfortables Feedback an die Entwicklung.

Sie managen Ihre Arbeit in Jira? Perfekt – Servicetrace Test Automation lässt sich voll integrieren.

Robotic News abonnieren & up to date bleiben
www.servicetrace.com/de/robotic-news