Testautomatisierung

Eine kurze Einführung

Ziele

Warum sollte man Tests automatisieren

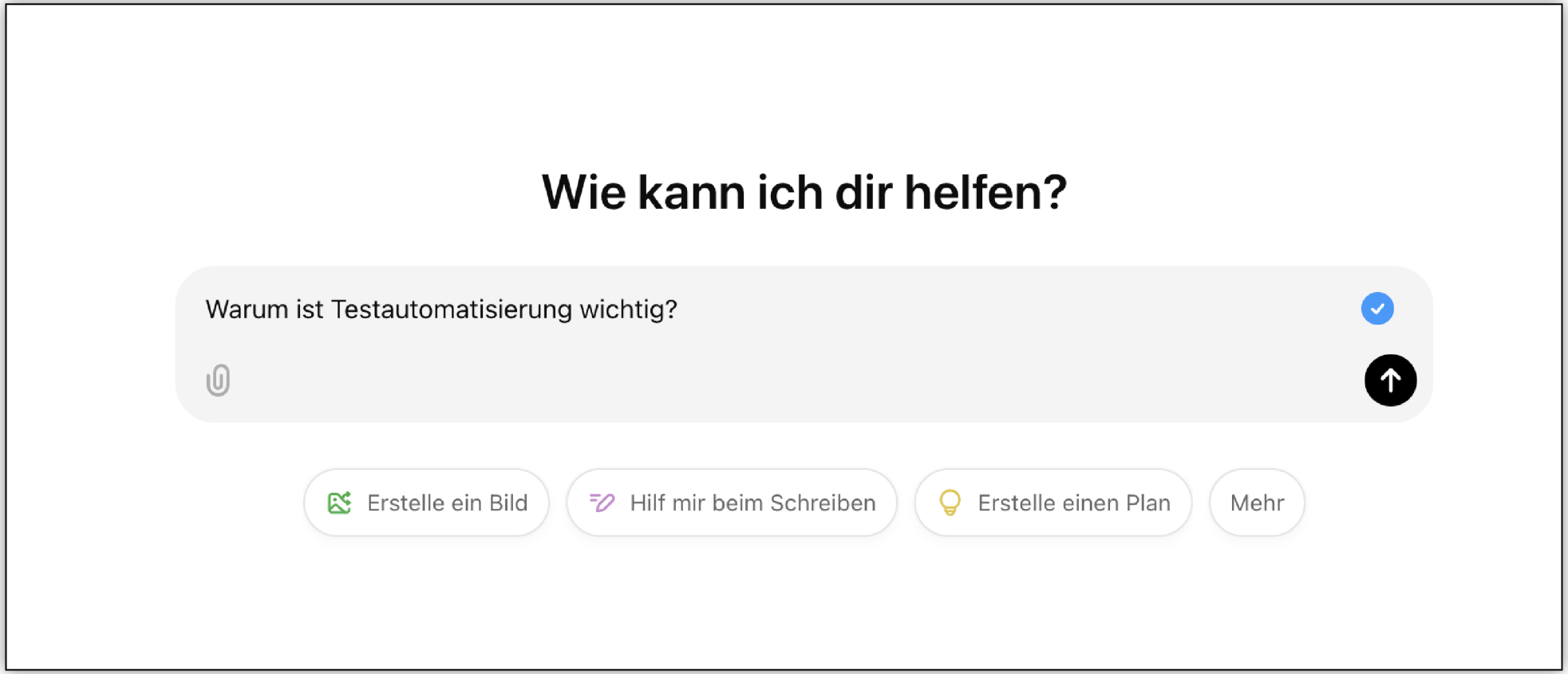
Testautomatisierung

Frage in die Runde (bezüglich Software, Webseiten):

Warum testen?

Warum Tests automatisieren?

Testautomatisierung



Testautomatisierung (ChatGPT)

Testautomatisierung ist ein wesentlicher Bestandteil moderner Softwareentwicklung, da sie zahlreiche Vorteile bietet, die zur Verbesserung der Qualität, Effizienz und Kostenkontrolle beitragen. Hier sind die wichtigsten Gründe, warum Testautomatisierung wichtig ist:

- Erhöhte Testabdeckung
- Schnelligkeit und Effizienz
- Fehlererkennung in frühen Phasen
- Wiederholbarkeit und Konsistenz
- Kostenersparnis auf lange Sicht
- Schnellere Markteinführung
- Verlässliche Regressionstests
- Bessere Ressourcennutzung
- Skalierbarkeit
- Einfache Integration in CI/CD-Pipelines

Ein guter (Software) Test ist?

- Zielgerichtet
- Vollständig
- Wiederholbar
- Effizient
- Automatisierbar
- Fehlerorientiert
- Verstehbar
- Kosteneffizient
- Unabhängig
- Ergebnisorientiert
- Realitätsnah
- Anpassungsfähig
- Dokumentiert

Ziele

Schlussendlich möchte man sich vor unerwarteten (unschönen) Ereignissen schützen.

Das funktioniert am besten, mit einem Test-Set welches übliche Prozesse (Workflows) der Benutzer abbildet.

Hierbei ist nicht jede "Komponente" gleich wichtig.

Ziele

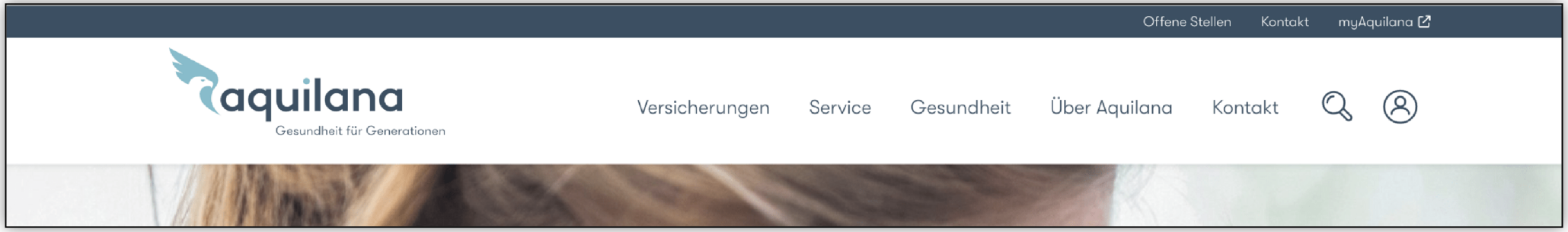
Schlussendlich möchte man sich vor unerwarteten (unschönen) Ereignissen schützen.

Das funktioniert am besten, mit einem Test-Set welches übliche Prozesse (Workflows) der Benutzer abbildet.

Hierbei ist nicht jede "Komponente" gleich wichtig.

Ein Beispiel zu Prioritäten

Header einer Krankenversicherung (Schweiz)

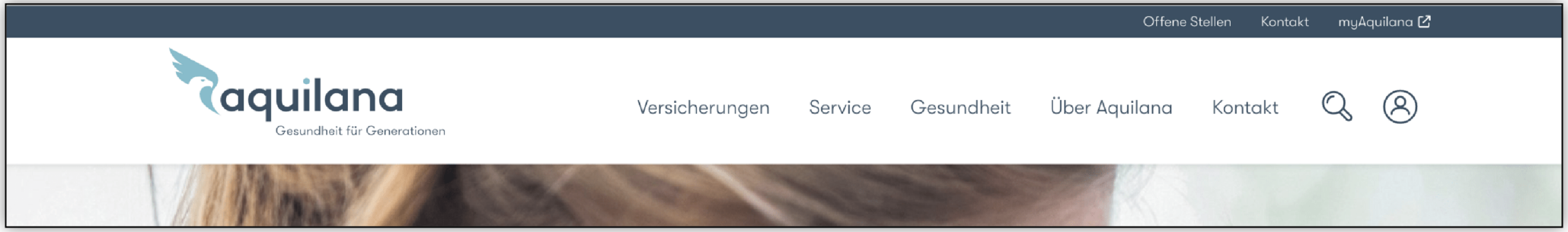


Wie wichtig sind die Links?

Welcher am wichtigsten / unwichtigsten?

Ein Beispiel zu Prioritäten

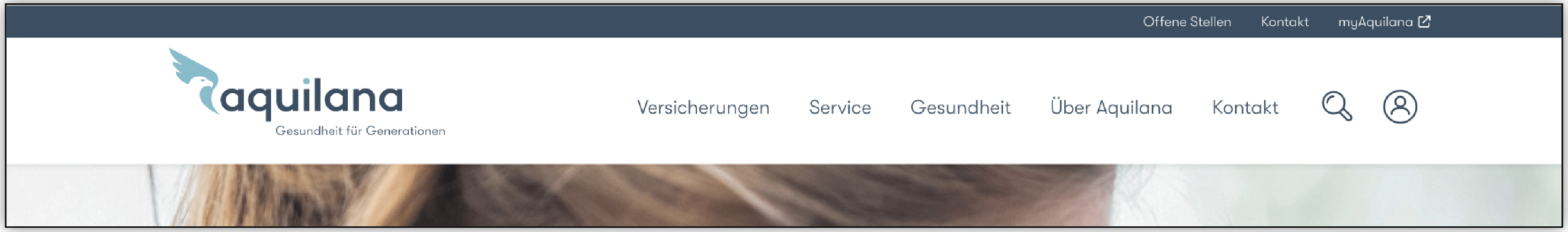
Header einer Krankenversicherung (Schweiz)



Es mag banal klingen, aber der Link zu den "Service", "Kontakt" (sogar 2x), und "myAquilana (Kundenportal)" sind für den **`Bestandskunden`** sehr wichtig.

Ein Beispiel zu Prioritäten

Header einer Krankenversicherung (Schweiz)



Jedoch sind diese Links für **`Interessierte`** sehr uninteressant. Hier sind eher "Versicherungen" und "Service" und "über Aquilana" wichtig.

Trotzdem sollte man, sicherstellen, dass diese Links **`nach jedem Update`** weiterhin funktionieren.

Ein Beispiel zu Prioritäten

Footer einer Krankenversicherung (Schweiz)

Frage: Was denkt ihr, ist hier der «wertvollste Link» ?

Fazit

Wir merken schnell, es gibt für Software und Webseiten mehrere Nutzerprofile.

Oft werden hierfür Personas verwendet:

Personas Definition

Personas **veranschaulichen typische Vertreter einer Zielgruppe**. Sie haben Erwartungen, Werte, Wünsche und Ziele und zeigen menschliche Verhaltensweisen. Eine Persona ist die Personifizierung bzw. der Prototyp einer Zielgruppe und hilft dabei, Annahmen über Kunden zu treffen.

Fazit

Kennt man seine Nutzerprofile, kann man besser ableiten, **`welche Funktionalitäten eine hohe Priorität geniessen`**.

Nun könnte man hingehen, und jemanden einstellen, der bei jedem Update alle diese Funktionalitäten stets nach dem gleichen Muster testet.

Doch die Zeit (und Konzentration) **`ist limitiert`** und viele Webseiten fügen immer **`mehr Features`** hinzu und nicht weniger.

Fazit

Priorisierung ist wichtig, sowohl bei manuellen Tests als auch bei Automatisierungen.

Wenn man aber zusätzlich merkt, dass ein Test immer gleich ist und sehr oft ausgeführt werden muss (aufgrund hoher Kundeninteraktion) ist dieser Test optimal für Automatisierung.

Frage

Nochmals zurück zur Krankenversicherung:

Was würden Sie zuerst automatisieren:

1) Prämienrechner (um die Monatsprämie zu berechnen)

2) Kontaktformular (um mit dem Krankenversicherer zu kommunizieren)

Frage

Offensichtlich sind beides sehr wichtige Funktionen, aber welche ist komplexer zum Testen (und damit teurer)?

1) Prämienrechner (um die Monatsprämie zu berechnen)

2) Kontaktformular (um mit dem Krankenversicherer zu kommunizieren)

Frage

Offensichtlich sind beides sehr wichtige Funktionen, aber welche ist komplexer zum Testen (und damit teurer)?

1) Prämienrechner (um die Monatsprämie zu berechnen)

viel komplizierter!

2) Kontaktformular (um mit dem Krankenversicherer zu kommunizieren)

Mission?

Was ist also unsere **"Mission"** in der Testautomatisierung?

Wir müssen die Testressourcen (Menschen) schonen.

+

Wir müssen die wichtigsten (Geschäfts-) Prozesse sicherstellen.

Wie

Die richtigen Testfälle und Teststrategien

Testfälle und Teststrategien

Leider gibt es hierauf keine Antwort, die allgemeingültig ist.

Es erfordert Wissen über die Benutzer (Personas) der Webseite oder Software und über die eigenen Möglichkeiten.

Ein geschickter, gut überlegter Test kann viele unnötige Tests ersetzen.

Testfälle und Teststrategien

Hierzu eine Frage:

Sollte man Konstruktoren in Java Unit-Testen?

Testfälle und Teststrategien

Hierzu eine Frage:

Sollte man Konstruktoren in Java Unit-Testen?

Die Antwort kann durchaus Ja sein, aber in den meisten Fällen eher nein. Zumindest nicht isoliert, meint auch ChatGPT.

Testfälle und Teststrategien

Hierzu eine Frage:

Sollte man Konstruktoren in Java Unit-Testen?

Die Antwort kann durchaus Ja sein, aber in den meisten Fällen eher nein. Zumindest nicht isoliert, meint auch ChatGPT.

Fazit

In der Regel **sollte der Konstruktor selbst nicht direkt getestet werden**, es sei denn, er enthält komplexe Logik. Stattdessen sollten die Tests auf den **Methoden des Objekts** fokussiert werden, die nach der Konstruktion aufgerufen werden. Testen Sie das Verhalten des Objekts insgesamt, anstatt sich nur auf die Instanziierung zu konzentrieren.

Testfälle

Die Testfälle haben wir nun einigermassen angeschaut, wie sieht es aber mit Teststrategien aus?

Teststrategie

Genauso wichtig wie Testfälle (abhängig von Personas). Sollte man sich auch die richtige Teststrategie überlegen.

Hierzu 9 wichtige Punkte, die man beachten sollte:

Teststrategie (1)

Umfang und Ziele: Die Teststrategie legt den Umfang fest, indem sie die zu testenden Softwarebereiche und Funktionalitäten spezifiziert. Ausserdem werden die Ziele des Testens beschrieben, z. B. die Identifizierung von Fehlern, die Validierung von Anforderungen, die Gewährleistung der Konformität und die Messung der Produktqualität.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie (2)

Teststufen und -arten: Die Strategie skizziert verschiedene Testebenen, einschließlich Unit-Tests, Integrationstests, Systemtests und Abnahmetests. Sie definiert auch die Testarten wie funktionale, Leistungs-, Sicherheits- und Anwendertests.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie (3)

Testumgebungen: In der Teststrategie werden die erforderlichen Testumgebungen für jede Testebene beschrieben, einschließlich Hardware, Software, Netzwerkkonfigurationen und Datensetups, um realitätsnahe Szenarien zu simulieren.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie (4)

Eingangs- und Ausgangskriterien: Eingangskriterien definieren die Bedingungen, die Ihre Software erfüllen muss, bevor das Testen auf einer bestimmten Ebene beginnen kann. Im Gegensatz dazu legen die Ausstiegskriterien die Bedingungen fest, unter denen das Testen für diese Stufe abgeschlossen ist.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie (5)

Ergebnisse des Testens: Die Teststrategie umreisst die zu erbringenden Leistungen für das Testen, wie Testpläne, Testfälle, Skripte, Daten und Berichte.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie (6)

Defekt-Management: Die Strategie für das Testen umfasst Richtlinien für die Berichterstattung, Verfolgung und Behebung von Fehlern, die Definition der Schwere und Priorität von Fehlern und die Art und Weise, wie Sie diese den Beteiligten mitteilen.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie (7)

Ressourcenzuweisung: Die Strategie adressiert die Zuweisung von Ressourcen, einschließlich Mitarbeitern, Testwerkzeugen und Infrastruktur, um die Testen-Aktivitäten effektiv durchzuführen.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie (8)

Risiken und Ausweichmöglichkeiten: In der Strategie werden potenzielle Risiken beim Testen identifiziert und Eventualitäten zur Adressierung dieser Risiken aufgestellt. Sie gewährleistet, dass das Risikomanagement ein integraler Bestandteil des Testens ist.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie (9)

Verbesserung des Testprozesses: Die Strategie enthält auch Bestimmungen zur kontinuierlichen Verbesserung des Testens auf der Grundlage der Erkenntnisse aus früheren Projekten oder des während des Testens erhaltenen Feedbacks.

https://aqua-cloud.io/de/testing-strategy/

Teststrategie

Es ist absolut lohnenswert, sich diese 9 Fragestellungen beim nächsten Projekt vor Augen zu führen. Da damit überflüssige Tests reduziert werden können.

Aus meiner eigenen Erfahrung wird oft einfach "getestet" ohne Planung, Strategie und Gedanken zum eigentlichen Benutzer.

Wie sind eure Erfahrungen?

Automatisierung

Muss sich auf jeden Fall lohnen

Automatisierung

Sobald man eine Strategie definiert hat, sich Gedanken zu "Flows" der Benutzer gemacht hat und ggf. auch analytische Werte zu Rate gezogen ist man vorbereitet.

Nun ist es an der Zeit "**`High Value`**" oder "**`High Complex`**" Flows zu bestimmen.

High Value

Bei High Value Flows geht es darum, sich in den Benutzer zu denken und sich vorzustellen, welcher Fehler veranlasst:

- Frustration / Ärger
- Umständliche Workarounds
- Zeit von Mitarbeitern (Kundendienst / Support)

Die richtige Fragestellung ist hier "wie viel kostet mich ein Fehler"?

High Complexity

Bei High Complexity Flows geht es darum, kritische Funktionalitäten zu garantieren, die bei Fehlern zu hohem (Entwicklungs-) Aufwand führen.

Hier betrachten wir die Unternehmens und nicht die Kundensicht.

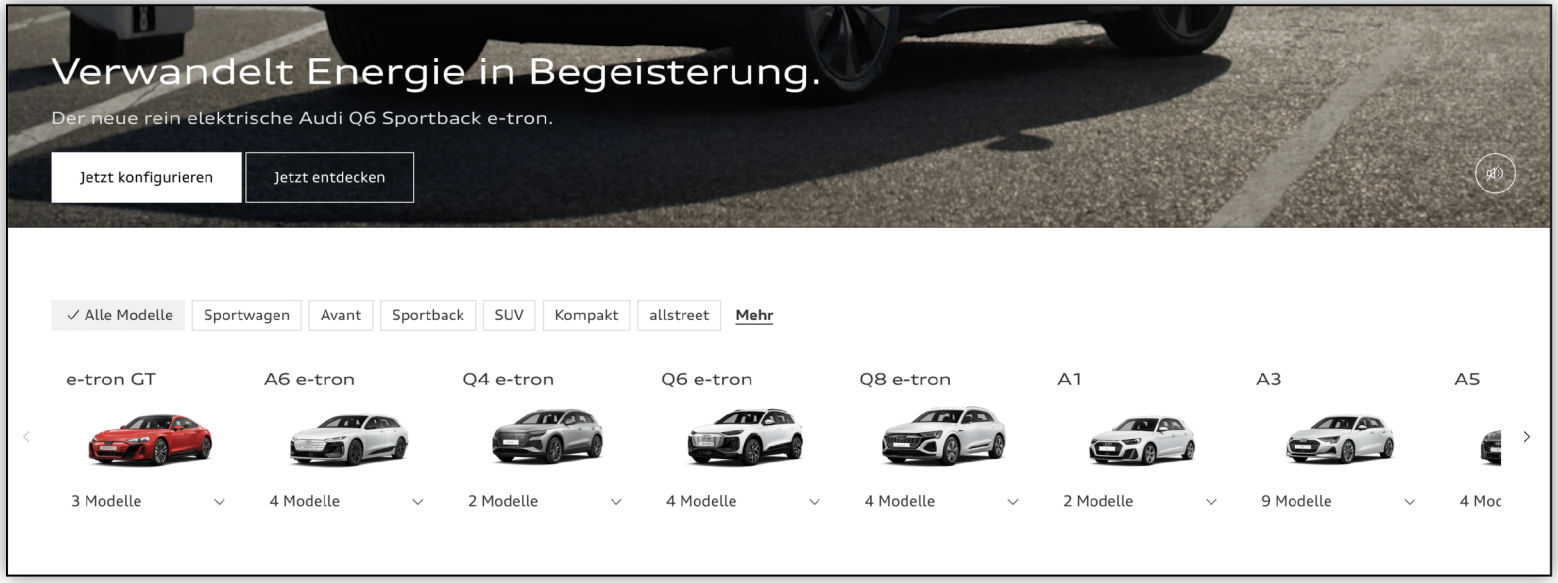
Die richtige Fragestellung ist hier "wie viel kostet es mich, einen Fehler zu beheben"?

Gedankenspiel

Wir kümmern uns um die Webseite von Audi.

Was könnte hier **`High Value`** oder **`High Complexity`** sein?

Nehmt euch 2-3 Min, um die Webseite zu besuchen.



Gedankenspiel Version 2

Bei Audi kennen wir die "interna" nicht, daher ist High Complexity sehr schwer für uns Aussenstehende zu bestimmen.

Wie steht es mit eurer Firmenwebseite?

Was wäre hier High Value und High Complexity?

Nutzwertmatrix

Wichtig, hier ist die Häufigkeit / Anzahl Kunden vernachlässigt - wir rechnen vereinfacht einfach mit 1 Tag Ausfall mehr, weil das Problem nicht bemerkt wurde beim Release.

Wenn ich nun die "wertvollsten" Flows kenne, kann man eine kleine **`vereinfachte`** Nutzwertmatrix anstellen:

Flow	Wert (Kosten) pro Fehler / Ausfall	Aufwand zur Automatisierung	Wahrscheinlichkeit
Flow 1	10'000 (pro Tag)	20h	Hoch
Flow 2	15'000 (pro Tag)	100h	Tief
Flow 3	100 (pro Tag)	1h	Sehr Hoch

Welchen Flow würden sie angehen?

Wichtig, hier ist die Häufigkeit / Anzahl Kunden vernachlässigt - wir rechnen vereinfacht einfach mit 1 Tag Ausfall mehr, weil das Problem nicht bemerkt wurde beim Release.

Nutzwertmatrix

Wenn ich nun die "wertvollsten" Flows kenne, kann man eine kleine **`vereinfachte`** Nutzwertmatrix anstellen:

Flow	**Wert (Kosten) pro Fehler / Ausfall**	**Aufwand zur Automatisierung**	**Wahrscheinlichkeit**
Flow 1	10'000 (pro Tag)	20h	Hoch
Flow 2	15'000 (pro Tag)	100h	Tief
Flow 3	100 (pro Tag)	1h	Sehr Hoch

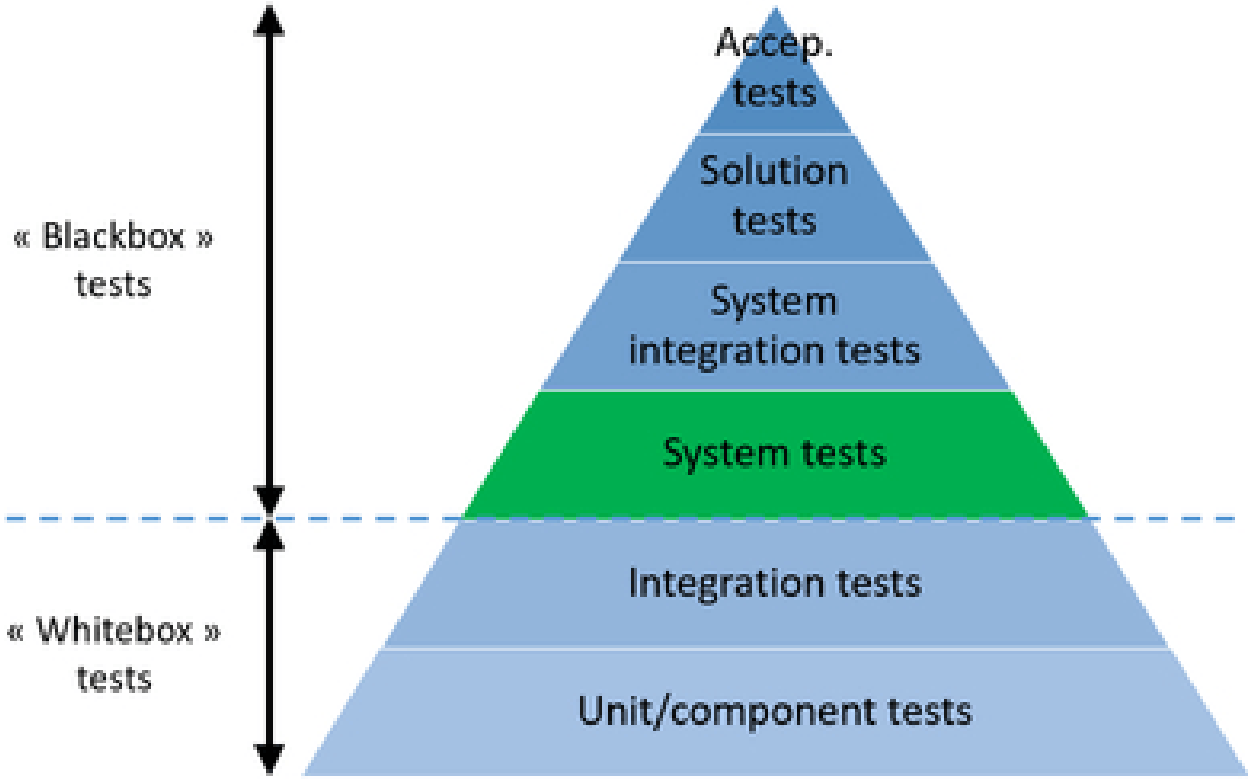
Welchen Flow würden sie angehen?

Variante 10'000/20 = 500 | 15'000/100 = 150 | 100/1 = 100

Testarten

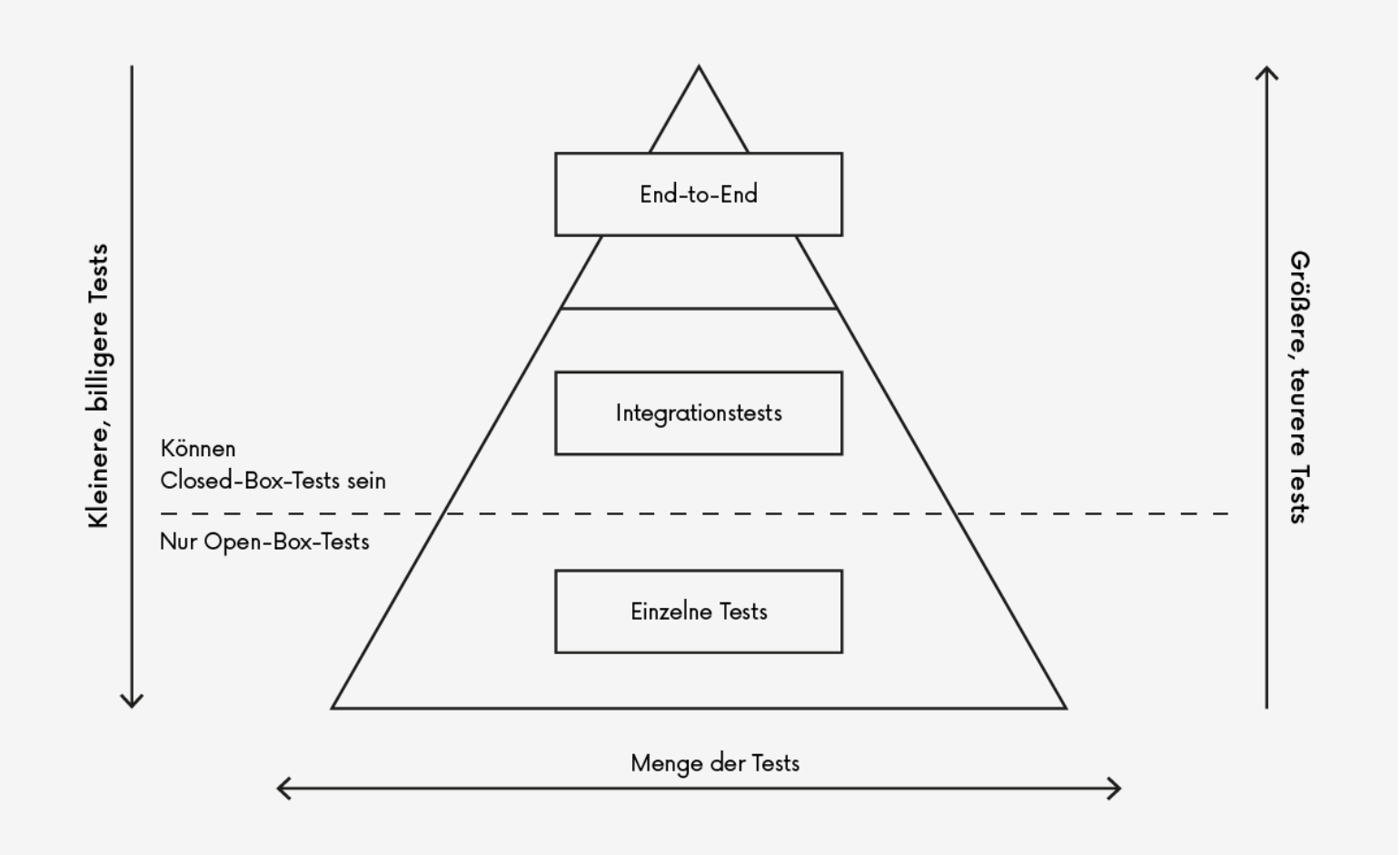
Welche Möglichkeiten habe ich?

Testpyramide



https://blog.atinternet.com/de/systemtests-ein-objektiver-blick-auf-das-produkt/

Testpyramide



https://vmsoftwarehouse.de/9-schritte-fur-die-testautomatisierung

Wahl des Toolings

Die Wahl des Toolings sollte in erster Linie die Testpyramide und die Flows im Auge behalten. **Was möchte ich wie testen?**

Danach ein kurzer Blick **nach vorn**, wie könnte "Testing" i**n 6-12 Monaten aussehen**, was für Ziele (Einsparungen) kann ich erreichen?

Schlussendlich ein letzter Blick ins Team, was ist der **Tech-Stack** und wie sind **die Skills im Team**?

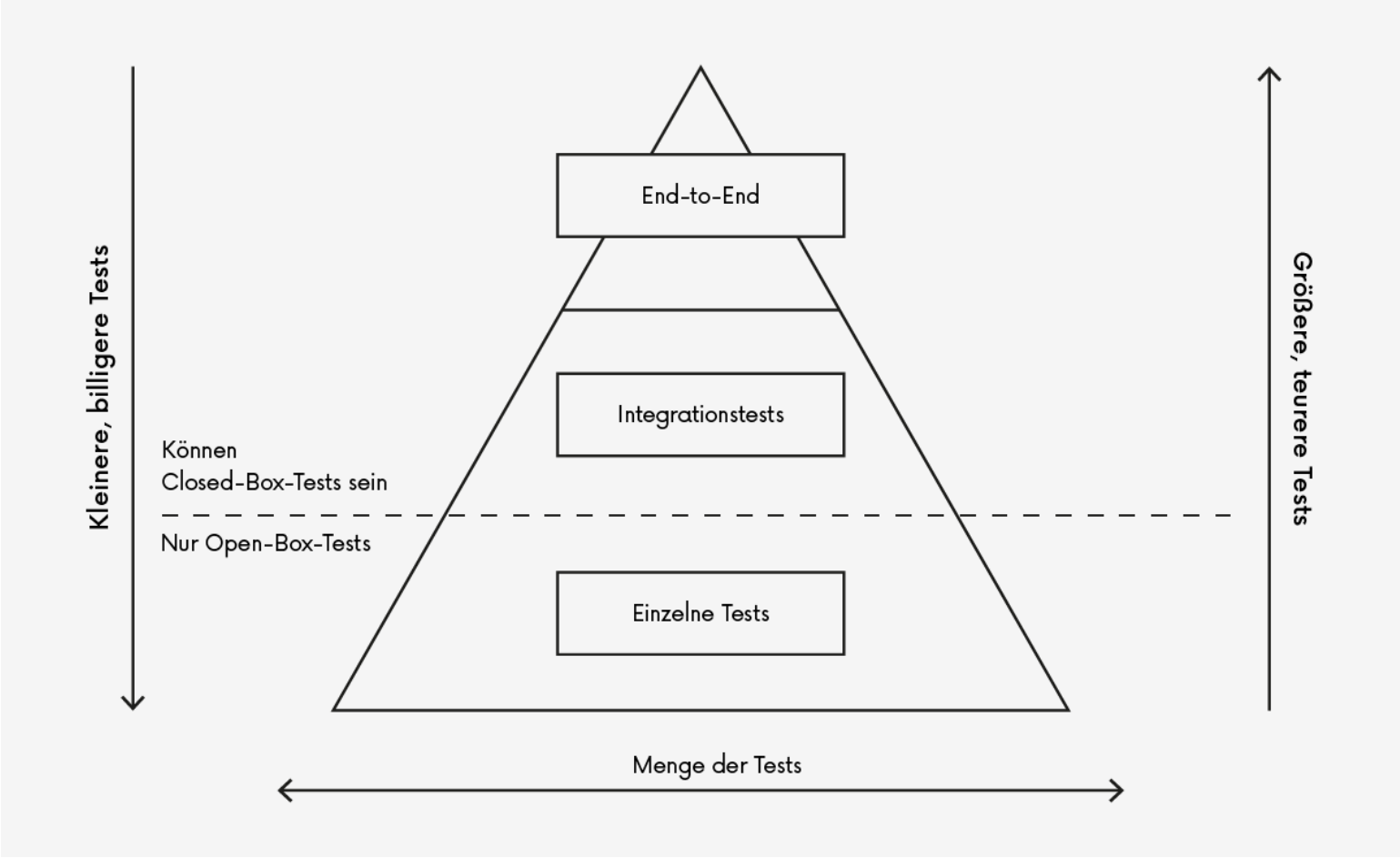
Wahl des Toolings

Web Testing Frameworks	**Selenium** • UEs unterstützt mehrere Browser und Sprachen. • Es lässt sich mit anderen Tools wie TestNG, JUnit und Maven integrieren.	**Cypress** • Ein JavaScript-basiertes Framework für End-to-End-Tests. • Ermöglicht das Nachladen in Echtzeit und das automatische Starten.	**Playwright** • Entwickelt von Microsoft. • Es unterstützt viele Browser, wie Chrome, Firefox und WebKit. • Die Möglichkeit, den Browser direkt zu automatisieren.
Mobile Testing Frameworks	**Appium** • Unterstützt plattformübergreifende Tests für iOS und Android • Es unterstützt viele Sprachen, wie Java, Ruby, Python usw.	**Espresso** • Googles Framework für Android. • Ermöglicht schnelle und zuverlässige Tests der Benutzeroberfläche. • Es lässt sich gut mit Android Studio integrieren.	**XCUITest** • Apples Framework für iOS. • Integriert in Xcode. • Ermöglicht robuste und effiziente Tests der Benutzeroberfläche.
API Testing Frameworks	**Postman** • Ein beliebtes Tool zum Erstellen und Testen von APIs. • Unterstützt automatisierte Tests mit Newman (command-line collection runner).	**RestAssured** • Java-Bibliothek zum Testen von RESTful-Webdiensten. • Es ist mit JUnit und TestNG integriert. • Es bietet eine einfach zu verwendende DSL für das Schreiben von Tests.	**SoapUI** • Ein Werkzeug zum Testen von SOAP- und REST-Webdiensten. • Es unterstützt sowohl Funktions- als auch Belastungstests.
Unit Testing Frameworks	**JUnit** • Weit verbreitet im Java-Ökosystem. • Enthält Anmerkungen zur Identifizierung von Prüfmethoden. • Es unterstützt die Integration mit IDEs und Kompilierwerkzeugen.	**TestNG** • Inspiriert von JUnit, aber effizienter. • Unterstützt die parallele Ausführung von Tests. • Es bietet eine flexible Testkonfiguration.	**pytest** • Beliebt in der Python-Gemeinschaft. • Einfache Syntax und erweiterte Funktionen. • Unterstützt Patching und parametrisierte Tests.

https://vmsoftwarehouse.de/9-schritte-fur-die-testautomatisierung

Wahl des Toolings

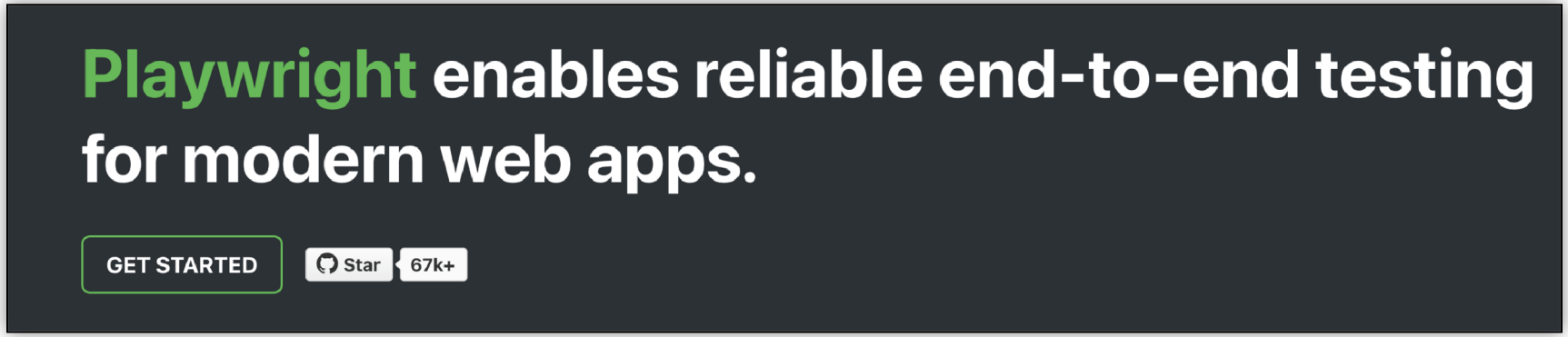
Web Testing Frameworks	Selenium	Cypress	Playwright
	• UEs unterstützt mehrere Browser und Sprachen. • Es lässt sich mit anderen Tools wie TestNG, JUnit und Maven integrieren.	• Ein JavaScript-basiertes Framework für End-to-End-Tests. • Ermöglicht das Nachladen in Echtzeit und das automatische Starten.	• Entwickelt von Microsoft. • Es unterstützt viele Browser, wie Chrome, Firefox und WebKit. • Die Möglichkeit, den Browser direkt zu automatisieren.



Wo würdet ihr Playwright ansiedeln?

https://vmsoftwarehouse.de/9-schritte-fur-die-testautomatisierung

Playwright


Playwright enables reliable end-to-end testing for modern web apps.
GET STARTED
Star
67k+

End to End vs. Integrationstest

End to End: End-to-End-Testing (E2E) ist eine Methode, bei der der gesamte Ablauf eines Systems von Anfang bis Ende getestet wird, **genau so, wie ein Nutzer es verwenden würde**. Ziel ist es sicherzustellen, dass alle Komponenten nahtlos zusammenarbeiten, einschliesslich: Frontend. Backend.

Integrationstest: Der Begriff Integrationstest bezeichnet in der Softwareentwicklung eine **aufeinander abgestimmte Reihe von Einzeltests**, die dazu dienen, verschiedene voneinander abhängige Komponenten eines komplexen Systems im Zusammenspiel miteinander zu testen.

Fazit

Wir werden uns in diesem Kurs vor allem auf End-to-End Aspekte des Testings konzentrieren.

Diese können beliebig gross oder klein ausfallen, sprich einzelne oder viele Aspekte behandeln.

Dennoch sollten sie nicht mit Integrationstests oder Einzeltests (Unit-Tests) verwechselt werden.

Fragen

Ende

Das war alles für dieses Kapitel