# Certified Tester
# Advanced Level
# Test Automation Engineering (CTAL-TAE)
# Syllabus

*v2.0*

---

International Software Testing Qualifications Board

---


---

Zur Verfügung gestellt von

Austrian Testing Board, German Testing Board und Swiss Testing Board



# Urheberschutzvermerk

# Änderungsübersicht

| Version | Datum | Änderung |
|---|---|---|
| Syllabus 2016 | 2016/10/21 | CTAL-TAE GA Release |
| Syllabus v2.0 (english) | 2024/05/03 | CTAL-TAE v2.0 GA Release |
| Syllabus v2.0 (deutsch) | 2024/11/27 | CTAL-TAE v2.0 deutsche Übersetzung |

# Inhaltsverzeichnis

# Danksagungen

Dieses Dokument wurde von der Generalversammlung des ISTQB® am 3. Mai 2024 formell freigegeben.

Es wurde von der Test Automation Task Force der Specialist Working Group des International Software Testing Qualifications Board erstellt: Graham Bath (Vorsitzender der Specialist Working Group), Andrew Pollner (stellvertretender Vorsitzender der Specialist Working Group und Vorsitzender der Test Automation Task Force), Péter Földházi, Patrick Quilter, Gergely Ágnecz, László Szikszai. Zu den Gutachtern der Test Automation Task Force gehörten: Armin Beer, Armin Born, Geza Bujdoso, Renzo Cerquozzi, Jan Giesen, Arnika Hryszko, Kari Kakkonen, Gary Mogyorodi, Chris van Bael, Carsten Weise, Marc-Florian Wendland.

Technischer Reviewer: Gary Mogyorodi

Die folgenden Personen haben an dem Review, der Kommentierung und der Abstimmung über diesen Lehrplan teilgenommen:

Horváth Ágota, Laura Albert, Remigiusz Bednarczyk, Jürgen Beniermann, Armin Born, Alessandro Collino, Nicola De Rosa, Wim Decoutere, Ding Guofu, Istvan Forgacs, Elizabeta Fourneret, Sudhish Garg, Jan Giesen, Matthew Gregg, Tobias Horn, Mattijs Kemmink, Hardik Kori, Jayakrishnan Krishnankutty, Ashish Kulkarni, Vincenzo Marrazzo, Marton Matyas, Patricia McQuaid, Rajeev Menon, Ingvar Nordström, Arnd Pehl, Michaël Pilaeten, Daniel Polan, Nishan Portoyan, Meile Posthuma, Adam Roman, Pavel Sharikov, Péter Sótér, Lucjan Stapp, Richard Taylor, Giancarlo Tomasig, Chris Van Bael, Koen Van Belle, Johan Van Berkel, Carsten Weise, Marc-Florian Wendland, Ester Zabar.

ISTQB Working Group Advanced Level Test Automation Engineer (Ausgabe 2016): Andrew Pollner (Vorsitz), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

Die folgenden Personen waren an der deutschen Übersetzung des Lehrplans beteiligt: Manfred Baumgartner, Jürgen Beniermann, Armin Born, Jan Giesen (Leiter der Arbeitsgruppe), Richard Seidl, Carsten Weise, Marc-Florian Wendland.

Am Beta-Review der deutschen Übersetzung waren die folgenden Personen beteiligt: Raphael Dumhart, Jessica Heymann, Uwe Martena und Arndt Pehl.

Unser herzlicher Dank geht an Ursula Zimpfer für ihre wertvolle Unterstützung bei der Bearbeitung der deutschsprachigen Fassung des vorliegenden Lehrplans.

# 0 Einleitung

## 0.1 Zweck dieses Lehrplans

Dieser Lehrplan bildet die Grundlage für die internationale Qualifikation Test Automation Engineering (CTAL-TAE) in der Aufbaustufe Advanced Level des Softwaretest-Qualifizierungsprogramms des International Software Testing Qualifications Board (im Folgenden ISTQB® genannt). Das German Testing Board e. V. (im Folgenden GTB® genannt) hat diesen Lehrplan in Zusammenarbeit mit dem Austrian Testing Board (ATB) und dem Swiss Testing Board (STB) in die deutsche Sprache übersetzt. Das GTB® und das ISTQB® stellen den Lehrplan folgenden Adressaten zur Verfügung:

1. Nationalen Mitgliedboards, die den Lehrplan in ihre Sprache(n) übersetzen und Schulungsanbieter akkreditieren dürfen. Die nationalen Mitgliedboards dürfen den Lehrplan an die Anforderungen ihrer nationalen Sprache anpassen und Referenzen hinsichtlich lokaler Veröffentlichungen berücksichtigen.
2. Zertifizierungsstellen zur Ableitung von Prüfungsfragen in ihrer nationalen Sprache, die an die Lernziele dieses Lehrplans angepasst sind.
3. Schulungsanbieter zur Erstellung von Lehrmaterialien und zur Bestimmung angemessener Lehrmethoden.
4. Zertifizierungskandidaten zur Vorbereitung auf die Zertifizierungsprüfung (entweder als Teil einer Schulung oder unabhängig davon).
5. Der internationalen Software- und Systementwicklungs-Community zur Förderung des Berufsbildes des Software- und Systemtesters und als Grundlage für Bücher und Fachartikel.

## 0.2 Testautomatisierungsentwickler im Softwaretest

Die Qualifizierung Test Automation Engineering richtet sich an alle, die sich mit Softwaretests und Testautomatisierung beschäftigen. Dazu gehören beispielsweise Tester, Testanalysten, Testautomatisierungsentwickler, Testberater, Testarchitekten, Testmanager und Softwareentwickler. Diese Qualifikation eignet sich auch für alle, die ein grundlegendes Verständnis für die Testautomatisierung erlangen wollen, wie z. B. Projektmanager, Qualitätsmanager, Softwareentwicklungsmanager, Business-Analysten, IT-Direktoren und Unternehmensberater.

Der Lehrplan CTAL Test Automation Engineering richtet sich an Testende, die Testautomatisierung implementieren oder verbessern wollen. Er definiert Methoden und Praktiken, die eine nachhaltige Lösung unterstützen können.

Weitere Richtlinien und Referenzmodelle, die sich auf Testautomatisierungslösungen beziehen, sind Softwareentwicklungsstandards für die ausgewählten Softwareentwicklungslebenszyklen, Programmiertechnologien und Formatierungsstandards. In diesem Lehrplan wird keine Softwareentwicklung gelehrt. Von Testautomatisierungsentwicklern wird jedoch erwartet, dass sie über Fähigkeiten, Erfahrungen und Fachwissen im Bereich der Softwaretechnik verfügen.

Darüber hinaus sollten Testautomatisierungsentwickler die branchenüblichen Programmier- und Dokumentationsstandards sowie Best Practices kennen, um sie bei der Entwicklung einer Testautomatisierungslösung anwenden zu können. Diese Praktiken können die Wartbarkeit, Zuverlässigkeit und IT-Sicherheit der Testautomatisierungslösung erhöhen. Solche Standards beruhen in der Regel auf Qualitätsmerkmalen.

## 0.3 Karriereweg für Tester und Testautomatisierungsentwickler

Das ISTQB®-Programm unterstützt Testexperten in allen Phasen ihrer Laufbahn und bietet ihnen sowohl eine breite als auch eine tiefe Wissensbasis. Personen, die die ISTQB®-Zertifizierung CTAL Test Automation Engineering erlangen, sind möglicherweise auch an der Qualifikation Testautomatisierungsstrategie (CT-TAS) interessiert.

Personen, die die ISTQB®-Zertifizierung CTAL Test Automation Engineering erlangen, können auch an den Core Advanced Levels (Test Analyst, Technical Test Analyst und Testmanager) und danach an den Expert Levels (Testmanagement oder Improving the Test Process) interessiert sein. Für alle, die ihre Fähigkeiten im Bereich des Testens in einer agilen Testumgebung ausbauen möchten, kommen die Zertifizierungen Agile Technical Tester oder Agile Test Leadership at Scale in Frage. Die Spezialisten-Lehrpläne bieten eine Vertiefung in Bereichen, die spezifische Testansätze und Testaktivitäten beinhalten, z. B. Testautomatisierungsstrategie, Performanztest, Sicherheitstest, KI-Test und Mobile Application Testing. Zudem gibt es Lehrpläne, die domänenspezifisches Wissen adressieren, wie z. B. Automotive Software Testing oder Game Testing. Aktuelle Informationen zum ISTQB® Certified Tester Scheme finden Sie unter *www.istqb.org*.

## 0.4 Geschäftlicher Nutzen

In diesem Abschnitt werden die geschäftlichen Nutzen (Business Outcomes, BO) aufgeführt, die von einem Kandidaten erwartet werden, der die Zertifizierung zum Testautomatisierungsentwickler erlangt hat.

Ein Kandidat, der die Zertifizierung CTAL Test Automation Engineering erworben hat, kann Folgendes:

| Kode | Beschreibung |
|---|---|
| TAE-BO1 | Zweck der Testautomatisierung beschreiben |
| TAE-BO2 | Testautomatisierung im Softwareentwicklungslebenszyklus verstehen |
| TAE-BO3 | Konfiguration einer Infrastruktur zur Ermöglichung der Testautomatisierung verstehen |
| TAE-BO4 | Den Evaluierungsprozess für die Auswahl der richtigen Werkzeuge und Strategien kennen |
| TAE-BO5 | Designkonzepte für den Aufbau modularer und skalierbarer Testautomatisierungslösungen verstehen |
| TAE-BO6 | Einen Ansatz, einschließlich eines Piloten, zur Planung der Testautomatisierung innerhalb des Softwareentwicklungslebenszyklus auswählen |
| TAE-BO7 | (Neue oder modifizierte) Testautomatisierungslösungen, die den technischen Anforderungen entsprechen, entwerfen und entwickeln |
| TAE-BO8 | Umfang und Ansatz der Testautomatisierung und Wartung von Testmitteln berücksichtigen |
| TAE-BO9 | Die Integration automatisierter Tests in CI/CD-Pipelines verstehen |
| TAE-BO10 | Verstehen, wie man Testautomatisierungsdaten sammelt, analysiert und darüber berichtet, um Stakeholder zu informieren |
| TAE-BO11 | Testautomatisierungs-Infrastruktur verifizieren |
| TAE-BO12 | Möglichkeiten zur kontinuierlichen Verbesserung der Testautomatisierung definieren |

## 0.5 Überprüfbare Lernziele und kognitiver Wissensstand

Die Lernziele (Learning Objectives, LO) unterstützen den geschäftlichen Nutzen und dienen zur Ausarbeitung der Prüfungen zum Certified Tester Advanced Level Test Automation Engineering (CTAL-TAE).

Im Allgemeinen sind alle Inhalte dieses Lehrplans auf den Stufen K2, K3 und K4 prüfbar, mit Ausnahme der Einleitung und der Anhänge. Das heißt, vom Prüfling kann gefordert werden, einen Schlüsselbegriff oder ein Konzept aus einem der acht Kapitel wiederzuerkennen, sich daran zu erinnern oder wiedergeben zu können. Die spezifischen Lernziele sind zu Beginn eines jeden Kapitels angegeben und wie folgt klassifiziert:

- K2: Verstehen
- K3: Anwenden
- K4: Analysieren

Weitere Einzelheiten und Beispiele für Lernziele können in Anhang A gefunden werden.

Alle Begriffe, die als Schlüsselbegriffe direkt unter den Kapitelüberschriften aufgelistet sind, müssen bekannt sein, auch wenn sie nicht ausdrücklich in den Lernzielen erwähnt werden.

## 0.6 Die Zertifizierungsprüfung Test Automation Engineering

Die Prüfung für das Zertifikat CTAL Test Automation Engineering basiert auf diesem Lehrplan. Zur Beantwortung einer Prüfungsfrage kann Wissen aus mehreren Abschnitten dieses Lehrplans erforderlich sein. Alle Abschnitte dieses Lehrplans sind prüfungsrelevant, außer der Einführung und den Anhängen. Im Lehrplan sind Standards, Fachbücher und andere ISTQB®-Lehrpläne als Referenzen genannt; deren Inhalt ist jedoch nur insoweit prüfungsrelevant, als er im vorliegenden Lehrplan in zusammengefasster Form enthalten ist.

Weitere Einzelheiten zur Prüfung für das Zertifikat CTAL Test Automation Engineering können im Dokument "Exam Structures and Rules" v1.1 des ISTQB® gefunden werden.

Voraussetzung für die Teilnahme an der Zertifizierung Test Automation Engineering ist, dass die Kandidaten ein Interesse an Softwaretests und Testautomatisierung haben. Es wird jedoch dringend empfohlen, dass die Kandidaten auch

- ein Mindestmaß an Erfahrung in der Softwareentwicklung und im Testen von Software haben, z. B. sechs Monate Erfahrung als Software-Testingenieur oder als Softwareentwickler,
- einen Kurs absolvieren, der nach ISTQB®-Standards akkreditiert ist (von einem der vom ISTQB® anerkannten Mitgliedsgremien).

Anmerkung zu den Anforderungen: Das ISTQB®-Zertifikat Certified Tester Foundation Level muss vor der ISTQB®-Zertifizierungsprüfung CTAL Test Automation Engineering erworben werden.

## 0.7 Akkreditierung

Ein nationales ISTQB®-Mitgliedboard kann Schulungsanbieter akkreditieren, deren Lehrmaterial diesem Lehrplan entspricht. Die Akkreditierungsrichtlinien können bei diesem nationalen Board (in Deutschland: German Testing Board e.V.; in der Schweiz: Swiss Testing Board; in Österreich: Austrian Testing Board) oder bei einer der Organisationen bezogen werden, die die Akkreditierung im Auftrag des nationalen Boards durchführt. Eine akkreditierte Schulung ist als konform mit diesem Lehrplan anerkannt und darf eine ISTQB®-Prüfung als Teil der Schulung enthalten.

Die Akkreditierungsrichtlinien für diesen Lehrplan folgen den allgemeinen Akkreditierungsrichtlinien, die von der ISTQB®-Arbeitsgruppe "Processes Management and Compliance" veröffentlicht wurden.

## 0.8 Umgang mit Normen und Standards

Im Lehrplan Test Automation Engineering wird auf einige Normen verwiesen (z. B. IEEE- und ISO-Normen). Der Zweck dieser Verweise ist es, einen Rahmen zu schaffen (wie bei den Verweisen auf *ISO/IEC 25010 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE)* (2023) bezüglich der Qualitätsmerkmale) oder eine Quelle für zusätzliche Informationen zu bieten, falls der Leser dies wünscht. Bitte beachten Sie, dass der Lehrplan die Normdokumente als Referenz verwendet. Die Inhalte der Normen sind nicht prüfungsrelevant. Weitere Informationen zu Normen finden Sie in *Kapitel 13*.

## 0.9 Auf dem Laufenden bleiben

Die Softwarebranche verändert sich schnell. Um mit diesen Veränderungen umzugehen und den Beteiligten den Zugang zu relevanten und aktuellen Informationen zu ermöglichen, haben die ISTQB®-Arbeitsgruppen auf der Website *www.istqb.org* Links angelegt, die auf unterstützende Dokumente und Änderungen an Standards verweisen. Diese Informationen sind im Rahmen des Lehrplans für das Zertifikat CTAL Test Automation Engineering nicht prüfungsrelevant.

## 0.10 Detaillierungsgrad

Der Detaillierungsgrad dieses Lehrplans ermöglicht international einheitliche Kurse und Prüfungen. Um dieses Ziel zu erreichen, besteht der Lehrplan aus:

- Allgemeinen Lehrzielen, die die Absicht des Lehrplans für das Test Automation Engineering beschreiben.
- Einer Liste von Begriffen, die die Schulungsteilnehmer kennen müssen.
- Lernzielen der einzelnen Wissensgebiete, die die zu erreichenden kognitiven Lernergebnisse beschreiben.
- Einer Beschreibung der Schlüsselkonzepte, einschließlich Verweisen auf Quellen wie anerkannte Literatur oder Normen.

Der Inhalt des Lehrplans ist keine Beschreibung des gesamten Wissensgebiets des Softwaretestens. Er spiegelt den Detaillierungsgrad wider, der in den Schulungskursen für das Zertifikat Test Automation Engineering abgedeckt werden soll. Der Schwerpunkt liegt auf Testkonzepten und -verfahren, die für alle Softwareprojekte gelten können, auch für solche, die nach agilen Methoden durchgeführt werden. Dieser Lehrplan enthält keine spezifischen Lernziele, die sich auf das agile Testen beziehen, aber er diskutiert, wie diese Konzepte in agilen Projekten und anderen Projektarten angewendet werden können.

## 0.11 Aufbau des Lehrplans

Es gibt acht Kapitel mit prüfbarem Inhalt. Die Überschrift auf der obersten Ebene jedes Kapitels gibt die Zeit für das Kapitel an. Unterhalb der Kapitelebene werden keine Zeitangaben gemacht. Für akkreditierte Ausbildungskurse verlangt der Lehrplan mindestens 21 Unterrichtsstunden (Anmerkung des Übersetzerteams: Die ISTQB-Kurse finden an ganzen Tagen mit jeweils 7 Stunden pro Tag statt. Der Lehrplaninhalt muss in diese Parameter passen, daher wurde die Anzahl der Unterrichtsstunden im Originallehrplan von 19.5 auf 21 aufgerundet, um die Gesamtzahl der Tage zu erreichen), die sich wie folgt auf die acht Kapitel verteilen:

- Kapitel 1: 45 Minuten - Einführung und Ziele der Testautomatisierung
  - Der Tester lernt die Vorteile der Testautomatisierung und ihre Grenzen kennen.
  - Die Testautomatisierung innerhalb verschiedener Softwareentwicklungslebenszyklus-Modelle wird behandelt.
  - Der Tester lernt, wie sich die Architektur eines Systems unter Test (SUT) auf die Eignung von Testwerkzeugen auswirkt.

- Kapitel 2: 180 Minuten - Vorbereitung auf die Testautomatisierung
  - Das Design für die Testbarkeit des SUT durch Beobachtbarkeit, Steuerbarkeit und eine klar definierte Architektur wird behandelt.
  - Der Tester lernt die Testautomatisierung in verschiedenen Umgebungen kennen.
  - Faktoren, die zur Beurteilung einer geeigneten Testautomatisierungslösung erforderlich sind, werden behandelt.
  - Der Tester lernt die technischen Überlegungen kennen, die für die Erarbeitung von Empfehlungen zur Testautomatisierung erforderlich sind.
- Kapitel 3: 210 Minuten - Testautomatisierungsarchitektur
  - Die Testautomatisierungsarchitektur und ihre Komponenten, die zu einer Testautomatisierungslösung führen, werden behandelt.
  - Der Tester lernt die Schichten und ihre Anwendung in einem Testautomatisierungsframework kennen.
  - Mehrere Ansätze zur Verwendung von Werkzeugen zur Testautomatisierung werden behandelt.
  - Der Tester lernt, wie Entwurfsprinzipien und Entwurfsmuster auf die Testautomatisierung angewendet werden können.
- Kapitel 4: 150 Minuten - Implementierung der Testautomatisierung
  - Es wird behandelt, wie man ein Pilotprojekt zur Testautomatisierung effektiv plant und einsetzt.
  - Der Tester lernt die Risiken der Implementierung und Strategien zur Risikominderung kennen.
  - Faktoren, die die Wartbarkeit von Code für die Testautomatisierung verbessern, werden behandelt.
- Kapitel 5: 90 Minuten - Implementierungs- und Bereitstellungsstrategien für die Testautomatisierung
  - Der Tester lernt etwas über CI/CD-Pipelines und automatisierte Testdurchführung auf verschiedenen Teststufen.
  - Das Konfigurationsmanagement für Komponenten der Testautomatisierung wird behandelt.
  - Der Tester lernt etwas über Abhängigkeiten bei API- und Vertragstests.
- Kapitel 6: 150 Minuten - Testautomatisierung - Berichte und Metriken
  - Der Tester lernt, wo Daten aus einem SUT und der Testautomatisierung für die Analyse und Berichterstattung gesammelt werden können.
  - Die Datenanalyse aus SUT-Berichten und Testautomatisierung zur Aufdeckung von Ursachen für Fehlerwirkungen wird behandelt.
  - Die Verwendung von Testberichten und Dashboards zur Information der Beteiligten wird behandelt.
- Kapitel 7: 135 Minuten - Verifizierung der Testautomatisierungslösung
  - Der Tester lernt, wie er die korrekte Funktionsweise der Komponenten und der Umgebung für die Testautomatisierung untersuchen und verifizieren kann.

  - Es wird sichergestellt, dass Testskripte und Testsuiten korrekt ausgeführt werden.
  - Der Tester wird verstehen, wann er eine Grundursachenanalyse durchführen muss.
  - Techniken zur Analyse von Testautomatisierungscode auf Qualität werden behandelt.
- Kapitel 8: 210 Minuten - Kontinuierliche Verbesserung
  - Zusätzliche Bereiche der Datenanalyse zur Verbesserung von Testfällen werden behandelt.
  - Der Tester lernt, wie er eine Testautomatisierungslösung und ihre Komponenten verbessern und aktualisieren kann.
  - Die Identifizierung von Möglichkeiten zur Konsolidierung und Rationalisierung der Testautomatisierung wird behandelt.
  - Der Tester lernt, wie Werkzeuge zur Testautomatisierung bei der Unterstützung und Einrichtung von Tests helfen können.

# 1 Einführung und Ziele der Testautomatisierung - 45 Minuten (K2)

**Schlüsselbegriffe**

System unter Test, Testautomatisierung, Testautomatisierungsentwickler

**Lernziele für Kapitel 1: Die Lernenden können ...**

## 1.1 Zweck der Testautomatisierung

TAE-1.1.1 (K2) ... die Vor- und Nachteile der Testautomatisierung erklären

## 1.2 Testautomatisierung im Softwareentwicklungslebenszyklus

TAE-1.2.1 (K2) ... den Einsatz der Testautomatisierung in verschiedenen Softwareentwicklungslebenszyklus-Modellen erklären

TAE-1.2.2 (K2) ... geeignete Testautomatisierungswerkzeuge für ein bestimmtes System unter Test auswählen

# 1.1 Zweck der Testautomatisierung

## 1.1.1 Die Vor- und Nachteile der Testautomatisierung erklären

Unter Testautomatisierung, die eine automatisierte Testausführung und Testberichterstattung umfasst, versteht man eine oder mehrere der folgenden Aktivitäten:

- Die Verwendung von speziell entwickelten Softwarewerkzeugen zur Steuerung und Einrichtung von Testsuiten für die Testausführung.
- Die automatisierte Ausführung von Tests.
- Den Vergleich der tatsächlichen mit den erwarteten Ergebnissen.

Eine Testautomatisierung bietet signifikante Funktionen und Fähigkeiten, die mit einem System unter Test (SUT) interagieren können. Die Testautomatisierung kann einen großen Bereich von Software abdecken. Mögliche Lösungen decken hierbei viele Arten von Software ab (z. B. SUT mit einer Benutzungsschnittstelle (UI), SUT ohne UI, mobile Anwendungen, Netzwerkprotokolle oder Schnittstellen).

Die Testautomatisierung hat viele Vorteile, wie z. B.:

- Sie ermöglicht im Vergleich zu manuellen Tests die Durchführung von mehr Testfällen pro Build.
- Sie bietet die Möglichkeit, Tests zu erstellen und auszuführen, die nicht manuell ausgeführt werden können (z. B. Echtzeitreaktionen, Remote-Tests oder parallele Tests).
- Sie ermöglicht Tests, die komplexer sind als manuelle Tests.
- Sie führt Testfälle schneller aus als bei manueller Ausführung.
- Sie ist weniger anfällig für menschliche Fehlhandlungen.
- Sie ist effektiver und effizienter in der Nutzung von Testressourcen.
- Sie bietet schnelleres Feedback über die Qualität des SUT.
- Sie hilft bei der Verbesserung der Systemzuverlässigkeit (z. B. Verfügbarkeit und Wiederherstellbarkeit).
- Sie verbessert die Konsistenz der Testdurchführung über Testzyklen hinweg.

Die Testautomatisierung hat jedoch auch potenzielle Nachteile, darunter:

- Zusätzliche Kosten für das Projekt, da möglicherweise ein Testautomatisierungsentwickler (TAE) eingestellt, neue Hardware gekauft und Schulungen durchgeführt werden müssen.
- Erforderliche Anfangsinvestition für die Einrichtung einer Testautomatisierungslösung.
- Benötigte Zeit für die Entwicklung und Wartung einer Testautomatisierungslösung.
- Das Erfordernis, klare Ziele für die Testautomatisierung zu definieren, um den Erfolg zu gewährleisten.
- Mangelnde Flexibilität der Tests und geringere Anpassbarkeit an Änderungen im SUT.
- Einführung zusätzlicher Fehlerzustände durch Testautomatisierung.

Es gibt Beschränkungen bei der Testautomatisierung, die beachtet werden müssen:

- Nicht alle manuellen Tests können automatisiert werden.
- Es wird nur geprüft, was in den automatisierten Tests definiert wurde.
- Testautomatisierung kann nur maschineninterpretierbare Ergebnisse überprüfen. Das bedeutet, dass bestimmte Qualitätsmerkmale möglicherweise nicht durch Automatisierung getestet werden können.
- Testautomatisierung kann nur Ergebnisse prüfen, die mit einem automatisierten Testorakel verglichen werden können.

## 1.2 Testautomatisierung im Softwareentwicklungslebenszyklus

### 1.2.1 Den Einsatz der Testautomatisierung in verschiedenen Softwareentwicklungslebenszyklus-Modellen erklären

**Wasserfall**

Das Wasserfallmodell ist ein SDLC-Modell (Software Development Lifecycle, SDLC), das sowohl linear als auch sequenziell ist. Bei diesem Modell gibt es verschiedene Phasen (d. h. Anforderungen, Entwurf, Implementierung, Verifizierung und Wartung). Jede Phase schließt in der Regel mit einer Dokumentation ab, die genehmigt werden muss. Die Implementierung der Testautomatisierung erfolgt normalerweise parallel zur oder nach der Implementierungsphase. Testläufe finden in der Regel während der Verifizierungsphase statt, da die Softwarekomponenten erst dann zum Testen bereit sind.

**V-Modell**

Das V-Modell ist ein SDLC-Modell, bei dem Prozessschritte sequenziell durchgeführt werden. Da ein Projekt von Anforderungen auf höheren Ebenen bis hin zu Anforderungen auf niedrigen Ebenen definiert wird, werden entsprechende Test- und Integrationsaktivitäten definiert, um diese Anforderungen zu verifizieren. Hieraus leiten sich die traditionellen Teststufen ab: Komponententest, Komponentenintegrationstest, Systemtest, Systemintegrationstest und Abnahmetest, wie in ISTQB-CTFL (2023), Abschnitt 2.2 beschrieben. Die Bereitstellung eines Testautomatisierungsframeworks (TAF) für jede Teststufe ist möglich und wird sogar empfohlen.

**Agile Softwareentwicklung**

In der agilen Softwareentwicklung gibt es viele Einsatzmöglichkeiten für eine Testautomatisierung. Im Gegensatz zum Wasserfall- oder V-Modell können bei einer agilen Softwareentwicklung TAEs und Stakeholder über die Roadmap, den Zeitplan und die geplante Testdurchführung entscheiden. Hierfür gibt es Best Practices wie Code-Reviews, Pair-Programming oder regelmäßige automatisierte Testausführungen. Durch die Beseitigung von Informationsinseln (d. h. die Sicherstellung der Zusammenarbeit von Entwicklern, Testern und anderen Stakeholdern) können Teams alle Teststufen mit einer Testautomatisierung in angemessenem Umfang und Tiefe abdecken und so eine Sprint-begleitende Testautomatisierung erreichen. Weitere Einzelheiten sind in ISTQB-CT-TAS (2024), Abschnitt 3.2 zu finden.

### 1.2.2 Geeignete Testautomatisierungswerkzeuge für ein bestimmtes System unter Test auswählen

Um die am besten geeigneten Testwerkzeuge für ein bestimmtes Projekt zu ermitteln, muss zunächst das SUT analysiert werden. TAEs müssen die Anforderungen an das Projekt identifizieren, um so eine Grundlage für die Werkzeugauswahl zu schaffen.

Da für UI-Software und beispielsweise Webservices unterschiedliche Features von Testautomatisierungswerkzeugen verwendet werden können, ist es wichtig zu verstehen, was das Projekt im Laufe der Zeit erreichen möchte. Es gibt keine Begrenzung für die Anzahl der Testautomatisierungswerkzeuge und -funktionen, die verwendet oder ausgewählt werden können. Allerdings sollten immer die Kosten dafür berücksichtigt werden. Die Verwendung kommerzieller Standardsoftware oder auch die Implementierung einer benutzerdefinierten Lösung auf Basis von Open-Source-Technologie kann durchaus ein komplexer Prozess sein.

Als Nächstes sollte evaluiert werden, wie die Zusammensetzung und Erfahrung des Teams ist, das bei der Testautomatisierung mitwirken soll. Wenn Tester wenig oder gar keine Programmiererfahrung haben, kann der Einsatz einer Low-Code- oder No-Code-Lösung eine sinnvolle, praktikable Wahl sein.

Für Tester mit Programmierkenntnissen kann es hilfreich sein, Werkzeuge auszuwählen, deren Sprache mit der des SUT übereinstimmt. Dies bietet unter anderem den Vorteil, gemeinsam mit Entwicklern mögliche Fehlerzustände der Testautomatisierung effizienter zu beheben, und die Möglichkeit, dass die Mitglieder verschiedener Teams sich untereinander austauschen und Wissen aufbauen können.

# 2 Vorbereitung auf die Testautomatisierung - 180 Minuten (K4)

**Schlüsselbegriffe**

API-Test, GUI-Test, Testbarkeit

**Lernziele für Kapitel 2: Die Lernenden können ...**

## 2.1 Die Konfiguration einer Infrastruktur zur Ermöglichung einer Testautomatisierung verstehen

TAE-2.1.1 (K2) ... Anforderungen an die Konfiguration einer Infrastruktur beschreiben, um Testautomatisierung zu ermöglichen

TAE-2.1.2 (K2) ... den Einsatz einer Testautomatisierung in verschiedenen Umgebungen erläutern

## 2.2 Den Evaluierungsprozess für die Auswahl der richtigen Werkzeuge und Strategien kennen

TAE-2.2.1 (K4) ... ein System unter Test analysieren, um die geeignete Testautomatisierungslösung zu bestimmen

TAE-2.2.2 (K4) ... technische Befunde einer Werkzeugevaluation veranschaulichen

# 2.1 Die Konfiguration einer Infrastruktur zur Ermöglichung der Testautomatisierung verstehen

## 2.1.1 Anforderungen an die Konfiguration einer Infrastruktur beschreiben, um Testautomatisierung zu ermöglichen

Die Testbarkeit des SUT (d. h. die Verfügbarkeit von Softwareschnittstellen, die das Testen unterstützen, z. B. um die Steuerung und Beobachtbarkeit des SUT zu ermöglichen) sollte parallel zum Entwurf und zur Implementierung der anderen Funktionen des SUT entworfen und implementiert werden. Diese Arbeit wird im Allgemeinen von einem Softwarearchitekten durchgeführt, da Testbarkeit eine nicht-funktionale Anforderung an das System ist. Dies geschieht oft unter Einbeziehung eines TAE, um die spezifischen Bereiche zu identifizieren, in denen Verbesserungen vorgenommen werden können.

Um die Testbarkeit des SUT zu verbessern, können verschiedene Lösungen eingesetzt werden, die unterschiedliche Konfigurationsanforderungen haben, z. B.:

- Objekterkennung durch Bezeichner
  - Die verschiedenen Entwicklungsframeworks können diese Bezeichner automatisch generieren oder die Entwickler können sie manuell setzen.
- Systemumgebungsvariablen
  - Bestimmte Anwendungsparameter können geändert werden, um das Testen durch die Administration zu erleichtern.
- Deployment-Variablen
  - Diese sind ähnlich zu Systemumgebungsvariablen, können aber vor dem Start des Deployments gesetzt werden.

Das Design für die Testbarkeit eines SUT besteht aus den folgenden Aspekten:

- Beobachtbarkeit: Das SUT muss Schnittstellen bereitstellen, die einen Einblick in das SUT ermöglichen. Testfälle können dann diese Schnittstellen nutzen, um festzustellen, ob die tatsächlichen Ergebnisse mit den erwarteten Ergebnissen übereinstimmen.
- Steuerbarkeit: Das SUT muss Schnittstellen bereitstellen, über die Aktionen auf dem SUT durchgeführt werden können. Dies können UI-Elemente, Funktionsaufrufe, Kommunikationselemente (z. B. Transmission Control Protocol/Internet Protocol (TCP/IP) und Universal-Serial- Bus-(USB-)Protokolle) oder elektronische Signale für physikalische sowie logische Schalter mit Hilfe der verschiedenen Umgebungsvariablen sein.
- Transparenz der Architektur: Die Dokumentation einer Architektur muss klare, verständliche Komponenten und Schnittstellen bereitstellen, die eine Beobachtbarkeit und Steuerbarkeit auf allen Teststufen ermöglichen und die Qualität fördern.

## 2.1.2 Den Einsatz einer Testautomatisierung in verschiedenen Umgebungen erläutern

Verschiedene Arten von automatisierten Tests können in unterschiedlichen Umgebungen ausgeführt werden. Diese Umgebungen können sich je nach Projekt und Methodik unterscheiden, wobei die meisten

Projekte über eine oder mehrere Umgebungen verfügen, die für das Testen genutzt werden. Aus technischer Sicht können diese Umgebungen aus Containern, Virtualisierungssoftware und anderen Ansätzen erstellt werden.

Eine Reihe möglicher Umgebungen, die in Betracht gezogen werden können, sind die folgenden:

**Lokale Entwicklungsumgebung**

In einer lokalen Entwicklungsumgebung wird die Software zunächst erstellt, und die Komponenten werden mit Hilfe von Automatisierung getestet, um die funktionale Eignung zu überprüfen. In der lokalen Entwicklungsumgebung können verschiedene Testarten auftreten, darunter Komponententests, GUI-Tests und Tests des Application Programming Interface (API). Wichtig ist auch, dass durch die Verwendung einer integrierten Entwicklungsumgebung (IDE) auf dem jeweiligen Computer White-Box-Tests durchgeführt werden können, um schlechte Codierung und Qualitätsprobleme so früh wie möglich zu erkennen.

**Build-Umgebung**

Ihr Hauptzweck besteht darin, die Software zu erstellen und Tests auszuführen, die die Korrektheit des resultierenden Builds in einem DevOps-Ökosystem überprüfen. Bei dieser Umgebung kann es sich entweder um eine lokale Build-Umgebung oder um einen CI/CD-Agenten (Continuous Integration/Continuous Delivery) handeln, in dem Low-Level-Tests (d. h. Komponententests und Komponentenintegrationstests) und statische Analysen ohne tatsächliche Bereitstellung in anderen Umgebungen durchgeführt werden können.

**Integrationsumgebung**

Nach der Durchführung von Low-Level-Tests und statischer Analyse ist die nächste Stufe eine Systemintegrationsumgebung. Hier liegt ein Release-Kandidat des SUT vor, der vollständig mit anderen Systemen integriert ist, die getestet werden können. In dieser Umgebung kann eine vollständig automatisierte Testsuite, entweder UI-Tests oder API-Tests, ausgeführt werden. In dieser Umgebung gibt es keine White-Box-Tests, sondern nur Black-Box-Tests (d. h. Systemintegrations- und/oder Abnahmetests). Es ist wichtig zu beachten, dass dies die erste Umgebung ist, in der Monitoring vorhanden sein sollte, um für eine effiziente Untersuchung von Fehlerzuständen/Fehlern zu sehen, was im Hintergrund während der Nutzung des SUT passiert.

**Vorproduktionsumgebung**

Eine Vorproduktionsumgebung wird meist zur Bewertung von nicht-funktionalen Qualitätsmerkmalen (z. B. Performanz) verwendet. Obwohl nicht-funktionale Tests in jeder Umgebung durchgeführt werden können, liegt ein besonderer Schwerpunkt auf der Vorproduktionsumgebung, da sie der Produktion so nahe wie möglich kommt. Häufig können Benutzerabnahmetests von den Stakeholdern durchgeführt werden, um das Endprodukt zu verifizieren. Falls erforderlich ist es möglich, die vorhandene automatisierte Testsuite auch hier auszuführen. Diese Umgebung wird ebenfalls durch Monitoring überwacht.

**Produktions-/Operationsumgebung**

Eine Produktionsumgebung kann verwendet werden, um funktionale und nicht-funktionale Qualitätsmerkmale in Echtzeit zu bewerten, während die Benutzer mit einem bereitgestellten System interagieren, wobei das Monitoring und bestimmte Best Practices das Testen in Produktion ermöglichen (z. B. Canary Release, Blue/Green Deployment und A/B-Tests).

# 2.2 Den Evaluierungsprozess für die Auswahl der richtigen Werkzeuge und Strategien kennen

## 2.2.1 Ein System unter Test analysieren, um die geeignete Testautomatisierungslösung zu bestimmen

Jedes SUT kann sich von einem anderen unterscheiden, dennoch gibt es verschiedene Faktoren und Merkmale, die analysiert werden können, um eine erfolgreiche Testautomatisierungslösung (engl. Test Automation Solution, TAS) zu finden. Während der Untersuchung eines SUT müssen TAEs Anforderungen unter Berücksichtigung des Umfangs und der gegebenen Funktionalitäten sammeln. Verschiedene Arten von Anwendungen (z. B. Webservice, mobile und Webanwendungen) benötigen aus technischer Sicht unterschiedliche Arten der Testautomatisierung. Die Untersuchung kann - und sollte - in Zusammenarbeit mit anderen Stakeholdern (z. B. manuellen Testern, Stakeholdern der Geschäftsbereiche und Business-Analysten) durchgeführt werden, um so viele Risiken wie möglich und deren Minderung zu identifizieren, um eine vorteilhafte Testautomatisierungslösung für die Zukunft zu haben.

Die Anforderungen an einen Testautomatisierungsansatz und eine Testautomatisierungsarchitektur sollten folgende Punkte berücksichtigen:

- Welche Aktivitäten des Testprozesses sollten automatisiert werden (z. B. Testmanagement, Testentwurf, Testgenerierung und Testdurchführung)?
- Welche Teststufen sollten unterstützt werden?
- Welche Testarten sollten unterstützt werden?
- Welche Testrollen und Fähigkeiten sollten unterstützt werden?
- Welche Softwareprodukte, Produktlinien und -familien sollen unterstützt werden (z. B. um den Zeitraum und die Lebensdauer der implementierten TAS zu definieren)?
- Welche Art von SUTs müssen mit der TAS kompatibel sein?
- Wie ist die Verfügbarkeit von Testdaten und deren Qualität?
- Welche Methoden und Wege sind möglich, um nicht erreichbare Fälle zu emulieren (z. B. Anwendungen von Drittanbietern)?

## 2.2.2 Technische Befunde einer Werkzeugevaluation veranschaulichen

Nachdem das SUT analysiert und die Anforderungen von allen Stakeholdern gesammelt wurden, gibt es wahrscheinlich Testautomatisierungswerkzeuge, die diese Anforderungen erfüllen und in Betracht gezogen werden können. Möglicherweise gibt es kein einzelnes Werkzeug, das alle identifizierten Anforderungen erfüllt. Die Stakeholder sollten sich dieser Möglichkeit bewusst sein.

Es ist hilfreich, die Befunde zu den möglichen Werkzeugen zu erfassen und die verschiedenen direkten und indirekten Anforderungen in einer Vergleichstabelle zu reflektieren. Ziel der Vergleichstabelle ist es, den Stakeholdern die Unterschiede zwischen den Werkzeugen auf der Grundlage spezifischer Anforderungen zu verdeutlichen. In der Vergleichstabelle werden die Werkzeuge in den Spalten und die Anforderungen in den Zeilen aufgeführt. Die Zellen enthalten Informationen über die Eigenschaften der einzelnen Werkzeuge in Bezug auf die einzelnen Anforderungen und über die Prioritäten.

Im Allgemeinen sollten die Werkzeuge zur Testautomatisierung daraufhin bewertet werden, ob sie die im vorherigen Abschnitt (*Kapitel 2.2.1*) genannten Anforderungen erfüllen. Zu den Anforderungen, die bei der Bewertung und dem Vergleich der Werkzeuge zu berücksichtigen sind, gehören die folgenden:

- Die Sprache/Technologie des Werkzeugs und der IDE-Werkzeuge
- Die Fähigkeit, ein Werkzeug zu konfigurieren, so dass es verschiedene Testumgebungen unterstützt, Konfigurationen ausführt und dynamische oder statische Setup-Werte verwendet.
- Die Fähigkeit, Testdaten innerhalb des Werkzeugs zu verwalten. Das Testdatenmanagement kann mit einem zentralen Repository für die Versionssteuerung integriert werden.
- Für verschiedene Testarten müssen möglicherweise unterschiedliche Werkzeuge zur Testautomatisierung ausgewählt werden.
- Die Fähigkeit, Berichte zu erstellen. Dies ist wichtig, um die Anforderungen an die Testberichterstattung des Projekts zu erfüllen.
- Die Fähigkeit andere Werkzeuge zu integrieren, die in einem Projekt oder in der Organisation verwendet werden, wie CI/CD, Aufgabenverfolgung, Testmanagement, Testberichterstattung oder andere Werkzeuge.
- Die Fähigkeit, die gesamte Testarchitektur zu erweitern und die Skalierbarkeit, Wartbarkeit, Modifizierbarkeit, Kompatibilität und Zuverlässigkeit von Werkzeugen zu bewerten.

Diese Vergleichstabelle ist eine gute Quelle, um ein vorgeschlagenes Werkzeug oder Werkzeugset für die Testautomatisierung des SUT zu bestimmen.

Der Prozess, wie die Entscheidung getroffen wird, welche(s) Werkzeug(e) verwendet werden soll(en), kann variieren. Der Vorschlag für ein Werkzeug sollte jedoch den entsprechenden Stakeholdern zur Genehmigung vorgelegt werden.

# 3 Testautomatisierungsarchitektur - 210 Minuten (K3)

**Schlüsselbegriffe**

datengetriebener Test (Data Driven Test), generische Testautomatisierungsarchitektur, lineare Skripterstellung, Mitschnitt, modellbasierter Test (Model Based Test), schlüsselwortgetriebener Test (Keyword Driven Test), strukturierte Skripterstellung, Testadaptierungsschicht, Testautomatisierungsframework, Testautomatisierungslösung, testgetriebene Entwicklung (Test Driven Development), Testschritt, Testskript, Testware, verhaltensgetriebene Entwicklung (Behavior Driven Development)

**Lernziele für Kapitel 3: Die Lernenden können ...**

## 3.1 Entwurfskonzepte in der Testautomatisierung anwenden

TAE-3.1.1 (K2) ... die wichtigsten Funktionen einer Testautomatisierungsarchitektur erläutern

TAE-3.1.2 (K2) ... erläutern, wie man eine Testautomatisierungslösung entwirft

TAE-3.1.3 (K3) ... die Schichten von Testautomatisierungsframeworks anwenden

TAE-3.1.4 (K3) ... die verschiedenen Ansätze zur Automatisierung von Testfällen anwenden

TAE-3.1.5 (K3) ... die Entwurfsprinzipien und Entwurfsmuster in der Testautomatisierung anwenden

# 3.1 Entwurfskonzepte in der Testautomatisierung anwenden

## 3.1.1 Die wichtigsten Funktionen einer Testautomatisierungsarchitektur erläutern

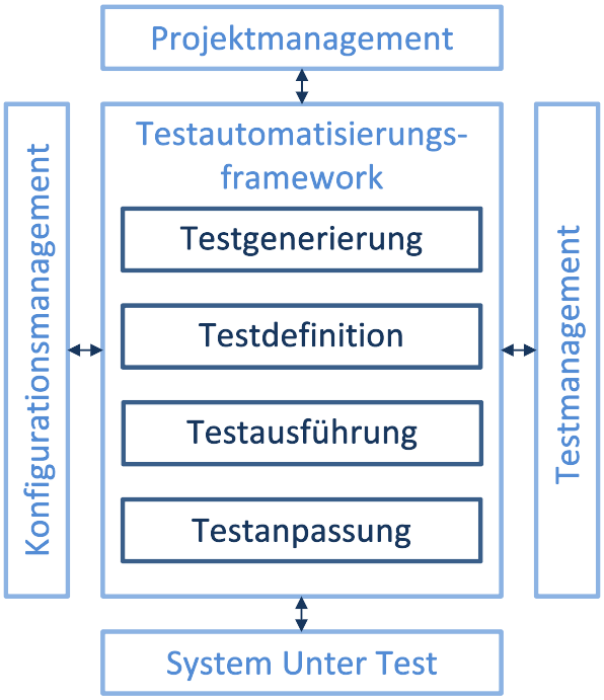


Abbildung 1: Generische Testautomatisierungsarchitektur (gTAA)

**Generische Testautomatisierungsarchitektur (gTAA)**

Die gTAA ist ein High-Level-Designkonzept, das eine abstrakte Sicht auf die Kommunikation zwischen der Testautomatisierung und den Systemen bietet, mit denen die Testautomatisierung verbunden ist, d. h. dem SUT, dem Projektmanagement, dem Testmanagement und dem Konfigurationsmanagement (siehe Abbildung 1). Sie stellt auch die Fähigkeiten zur Verfügung, die beim Entwurf einer Testautomatisierungsarchitektur (TAA) abgedeckt werden müssen.

Die Schnittstellen der gTAA beschreiben Folgendes:

- Die SUT-Schnittstelle beschreibt die Verbindung zwischen dem SUT und dem TAF (siehe *Kapitel 3.1.3* bezüglich des Testautomatisierungsframeworks).
- Die Projektmanagement-Schnittstelle beschreibt den Entwicklungsfortschritt der Testautomatisierung.
- Die Testmanagement-Schnittstelle beschreibt das Mapping von Testfalldefinitionen und automatisierten Testfällen.
- Die Schnittstelle für das Konfigurationsmanagement beschreibt die CI/CD-Pipelines, Umgebungen und Testmittel.

**Fähigkeiten von Werkzeugen und Bibliotheken zur Testautomatisierung**

Die Hauptfunktionen der Testautomatisierung sollten identifiziert und aus den verfügbaren Werkzeugen je nach Anforderung für ein bestimmtes Projekt ausgewählt werden.

**Testgenerierung:** Sie unterstützt den automatisierten Entwurf von Testfällen auf der Grundlage eines Testmodells. Modellbasierte Testwerkzeuge können für den Generierungsprozess genutzt werden (siehe ISTQB-CT-MBT (2015)). Die Generierung von Tests ist eine optionale Fähigkeit.

**Testdefinition:** Sie unterstützt die Definition und Implementierung von Testfällen und/oder Testsuiten, die optional aus einem Testmodell abgeleitet werden können. Sie trennt die Testdefinition vom SUT und/oder den Testwerkzeugen. Sie enthält die Mittel zur Definition von High-Level- und Low-Level-Tests, die in den Komponenten Testdaten, Testfälle und Testbibliothek oder in Kombinationen davon behandelt werden.

**Testdurchführung:** Sie unterstützt Testdurchführung und Testprotokollierung. Sie bietet ein Testausführungswerkzeug, um die ausgewählten Tests automatisch auszuführen, sowie eine Komponente zur Testprotokollierung und Testberichterstattung.

**Testanpassung:** Sie stellt die notwendige Funktionalität zur Verfügung, um die automatisierten Tests für die verschiedenen Komponenten oder Schnittstellen des SUT anzupassen. Sie bietet verschiedene Adapter für die Verbindung zum SUT über APIs, Protokolle und Dienste.

### 3.1.2 Erläutern, wie man eine Testautomatisierungslösung entwirft

Eine Testautomatisierungslösung (TAS) wird durch ein Verständnis der funktionalen, nicht-funktionalen und technischen Anforderungen des SUT sowie der vorhandenen oder erforderlichen Werkzeuge definiert, die zur Implementierung einer Lösung notwendig sind. Eine TAS wird mit kommerziellen oder Open-Source-Werkzeugen implementiert und benötigt möglicherweise zusätzliche SUT-spezifische Anpassungen.

Die TAA definiert den technischen Entwurf für die gesamte TAS. Sie sollte Folgendes beinhalten:

- Auswahl von Werkzeugen zur Testautomatisierung und werkzeugspezifischen Bibliotheken
- Entwicklung von Plugins und/oder Komponenten
- Identifizierung von Anforderungen an Konnektivität und Schnittstellen (z. B. Firewalls, Datenbanken, URLs/Verbindungen, Mocks/Platzhalter, Message Queues und Protokolle)
- Verbindung zu den Werkzeugen für das Testmanagement und Fehlermanagement
- Verwendung eines Versionskontrollsystems und von Repositories

### 3.1.3 Die Schichten von Testautomatisierungsframeworks anwenden

**Testautomatisierungsframework**

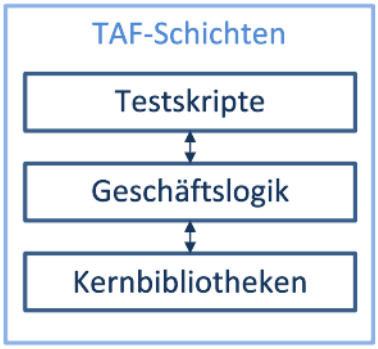


Abbildung 2: TAF-Ebenen

Das TAF ist die Grundlage für eine TAS. Es umfasst häufig einen Testrahmen, auch Test Runner genannt, sowie Testbibliotheken, Testskripte und Testsuiten.

**TAF-Schichten**

TAF-Schichten definieren eine klare Grenze von Klassen, die ähnliche Zwecke verfolgen, wie Testfälle, Testberichterstattung, Testprotokollierung, Verschlüsselung und Testrahmen. Durch die Einführung einer Schicht für jeden einzelnen Zweck kann der Entwurf kompliziert werden. Es wird daher empfohlen, die Anzahl der TAF-Schichten gering zu halten.

**Testskripte**

Diese Schicht dient dazu, ein Repository für Testfälle des SUT und Anmerkungen zur Testsuite bereitzustellen. Sie ruft die Dienste der Geschäftslogikschicht auf, was Testschritte, Benutzerabläufe oder API-Aufrufe beinhalten kann. Es sollten jedoch keine direkten Aufrufe der Kernbibliotheken aus Testskripten erfolgen.

**Geschäftslogik**

Alle SUT-abhängigen Bibliotheken werden in dieser Schicht gespeichert. Diese Bibliotheken erben die Klassen der Kernbibliotheken oder verwenden die von diesen bereitgestellten Fassaden (siehe *Kapitel 3.1.5* bezüglich Vererbung und Fassaden). Die Geschäftslogikschicht wird verwendet, um das TAF so einzurichten, dass es zusammen mit dem SUT und den zusätzlichen Konfigurationen läuft.

**Kernbibliotheken**

Alle Bibliotheken, die unabhängig von einem SUT sind, werden in dieser Schicht gespeichert. Diese Kernbibliotheken können in jeder Art von Projekt wiederverwendet werden, das den gleichen Development-Stack verwendet.

**Skalierung der Testautomatisierung**

Das folgende Beispiel (Abbildung 3) zeigt, wie die Kernbibliotheken eine wiederverwendbare Basis für mehrere TAFs bilden. In Projekt Nr. 1 werden zwei TAFs auf der Grundlage der Kernbibliotheken erstellt,

und ein separates Projekt (Nr. 2) nutzt die bereits vorhandenen Kernbibliotheken, um sein TAF zum Testen von Anwendung Nr. 3 zu erstellen. Ein TAE erstellt die TAFs für Projekt Nr. 1, während ein zweiter TAE das TAF für Projekt Nr. 2 erstellt.

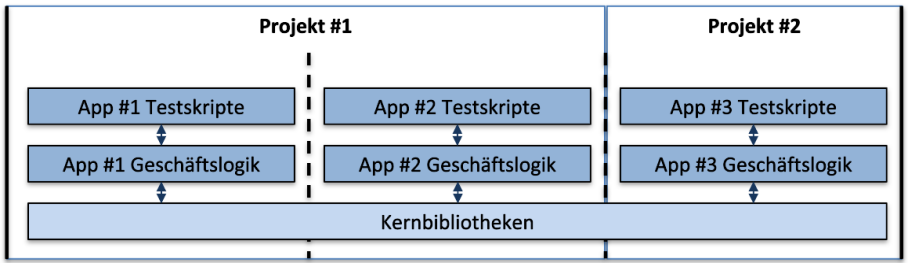


Abbildung 3: Anwendung von Kernbibliotheken auf mehrere TAFs

### 3.1.4 Die verschiedenen Ansätze zur Automatisierung von Testfällen anwenden

Es gibt verschiedene Entwicklungsansätze, aus denen Teams wählen können, um automatisierte Testfälle zu erstellen. Diese können interaktive Skriptsprachen oder kompilierte Programmiersprachen beinhalten. Die verschiedenen Ansätze bieten unterschiedliche Vorteile bei der Automatisierung und können unter verschiedenen Umständen genutzt werden. Obwohl es sich bei der testgetriebenen Entwicklung (engl. Test Driven Development, TDD) und der verhaltensgetriebenen Entwicklung (engl. Behavior Driven Development, BDD) um Entwicklungsmethoden handelt, führen sie bei richtiger Anwendung zu einer automatisierten Entwicklung von Testfällen.

**Mitschnitt**

Der Mitschnitt (engl. Capture/Playback) ist ein Ansatz, der die Interaktionen mit dem SUT aufzeichnet, während eine Sequenz von Aktionen manuell ausgeführt wird. Diese Werkzeuge erzeugen während der Erfassung Testskripte, und je nach verwendetem Werkzeug kann der Code für die Testautomatisierung modifizierbar sein. Die Werkzeuge, die ihren Code nicht freilegen, werden manchmal als No-Code-Testautomatisierung bezeichnet, während die Werkzeuge, die den Code freilegen, als Low-Code-Testautomatisierung bezeichnet werden.

Vorteile:

- Anfänglich einfach einzurichten und zu verwenden.

Nachteile:

- Schwer zu warten, zu erweitern und weiterzuentwickeln.
- Das SUT muss während der Erfassung eines Testfalls verfügbar sein.
- Nur machbar für einen kleinen Umfang und ein SUT, das sich selten ändert.
- Die aufgezeichnete SUT-Ausführung hängt stark von der SUT-Version ab, von der die Aufzeichnung gemacht wurde.
- Die Aufzeichnung jedes einzelnen Testfalls anstelle der Wiederverwendung bestehender Bausteine ist zeitaufwendig.

### Lineare Skripterstellung

Die lineare Skripterstellung ist eine Programmiertätigkeit, die keine von TAEs erstellten Testbibliotheken erfordert und zum Schreiben und Ausführen der Testskripte verwendet wird. Ein TAE kann beliebige Testskripte einsetzen, die von einem Mitschnittwerkzeug aufgezeichnet wurden und dann modifiziert werden können.

Vorteile:

- Einfaches Einrichten und Schreiben von Testskripten.
- Im Vergleich zum Mittschnittansatz können die Testskripte leichter modifiziert werden.

Nachteile:

- Schwer zu warten, zu erweitern und weiterzuentwickeln.
- Das SUT muss während der Aufzeichnung eines Testfalls verfügbar sein.
- Nur machbar für einen kleinen Umfang und ein SUT, das sich selten ändert.
- Im Vergleich zum Mittschnittansatz sind gewisse Programmierkenntnisse erforderlich.

### Strukturierte Skripterstellung

Es werden Testbibliotheken mit wiederverwendbaren Elementen, Testschritten und/oder User Journeys eingeführt. Für die Erstellung und Wartung von Testskripten sind bei diesem Ansatz Programmierkenntnisse erforderlich.

Vorteile:

- Einfache Wartbarkeit, Erweiterbarkeit, Portierung, Anpassbarkeit und Weiterentwicklung.
- Geschäftslogik kann von den Testskripten getrennt werden.

Nachteile:

- Programmierkenntnisse sind erforderlich.
- Die anfängliche Investition in die TAF-Entwicklung und die Definition der Testmittel ist zeitaufwendig.

### Testgetriebene Entwicklung

Testfälle werden als Teil des Entwicklungsprozesses definiert, bevor eine neue Funktion des SUT implementiert wird. Der TDD-Ansatz besteht aus Testen, Programmieren und Refactoring, auch bekannt als Red-Green-Refactor. Ein Entwickler identifiziert und erstellt einen Testfall, der zunächst fehlschlagen wird (rot, engl. red). Dann entwickelt er/sie die Funktionalität, die den Testfall erfüllen wird (grün, engl. green). Der Code wird dann überarbeitet, um ihn zu optimieren und die Grundsätze für Clean Code einzuhalten (Refactoring). Der Prozess wird mit dem nächsten Test und der nächsten Funktionserweiterung fortgesetzt.

Vorteile:

- Vereinfacht die Entwicklung von Testfällen auf Komponentenebene.
- Verbessert die Qualität des Codes und die Struktur des Codes.
- Verbessert die Testbarkeit.

- Erleichtert das Erreichen einer gewünschten Codeüberdeckung.
- Reduziert die Weitergabe von Fehlerzuständen an höhere Teststufen.
- Verbessert die Kommunikation zwischen Entwicklern, Geschäftsvertretern und Testern.
- User-Stories, die nicht durch GUI-Tests und API-Tests verifiziert werden, können durch TDD schnell die Endekriterien erfüllen.

Nachteile:

- Die Eingewöhnung in TDD braucht zunächst mehr Zeit.
- Wenn TDD nicht richtig befolgt wird, kann dies zu einem falschen Vertrauen in die Codequalität führen.

**Datengetriebenes Testen**

Datengetriebenes Testen (engl. Data Driven Testing, DDT) baut auf der strukturierten Skripterstellung auf. Die Testskripte werden mit Testdaten versehen (z. B. .csv-Dateien, .xlsx-Dateien und Datenbank-Dumps). So können dieselben Testskripte mehrfach mit unterschiedlichen Testdaten ausgeführt werden.

Vorteile:

- Ermöglicht die schnelle und einfache Erweiterung von Testfällen durch Datenfeeds.
- Die Kosten für das Hinzufügen neuer automatisierter Tests können erheblich gesenkt werden.
- Testanalysten können automatisierte Tests spezifizieren, indem sie eine oder mehrere Testdatendateien einfügen, die die Tests beschreiben. Dies gibt den Testanalysten mehr Freiheit, automatisierte Tests zu spezifizieren, ohne von den technischen Testanalysten abhängig zu sein.

Nachteile:

- Ein angemessenes Testdatenmanagement kann erforderlich sein.

**Schlüsselwortgetriebenes Testen**

Bei schlüsselwortgetriebenen Tests (engl. Keyword Driven Testing, KDT) handelt es sich um eine Liste oder Tabelle von Testschritten, die aus Schlüsselwörtern und den Testdaten abgeleitet werden, auf denen die Schlüsselwörter operieren. Die Schlüsselwörter werden aus der Sicht des Benutzers definiert. Diese Technik baut oft auf DDT auf.

Vorteile:

- Testanalysten und Unternehmensanalysten können an der Erstellung automatisierter Testfälle beteiligt werden, indem sie den KDT-Ansatz verfolgen.
- KDT kann auch für manuelles Testen unabhängig von der Testautomatisierung verwendet werden (siehe *ISO/IEC/IEEE 29119-5 Software and systems engineering – Software testing* (2016)).

Nachteile:

- Die Implementierung und Pflege von Schlüsselwörtern ist eine komplexe Aufgabe, die TAEs abdecken müssen, was bei wachsendem Umfang zu einer Herausforderung werden kann.
- Es verursacht einen enormen Aufwand für kleinere Systeme.

**Verhaltensgetriebene Entwicklung**

Verhaltensgetriebene Entwicklung (engl. Behavior Driven Development, BDD) nutzt ein natürliches Sprachformat (d. h. given, when, then) bei der Formulierung von Akzeptanzkriterien, die als automatisierte Testfälle verwendet und in Feature-Dateien gespeichert werden können. Ein BDD-Werkzeug kann dann die Sprache verstehen und die Testfälle ausführen.

Vorteile:

- Verbessert die Kommunikation zwischen Entwicklern, Geschäftsvertretern und Testern.
- Automatisierte BDD-Szenarien fungieren als Testfälle und gewährleisten die Überdeckung der Spezifikationen.
- BDD kann bei der Erstellung mehrerer Testarten auf den unterschiedlichen Ebenen der Testpyramide eingesetzt werden.

Nachteile:

- Zusätzliche Testfälle, typischerweise negative Testbedingungen und Randfälle, müssen immer noch vom Team definiert werden, typischerweise von einem Testanalysten oder einem TAE.
- Viele Teams verstehen unter BDD nur das Schreiben von Testfällen in natürlicher Sprache und beziehen die Geschäftsvertreter und Entwickler nicht in den gesamtheitlichen Ansatz ein.
- Die Implementierung und Pflege von Testschritten in natürlicher Sprache ist eine komplexe Aufgabe für TAEs.
- Übermäßig komplexe Testschritte machen das Debugging zu einer schwierigen und kostspieligen Aufgabe.

### 3.1.5 Die Entwurfsprinzipien und Entwurfsmuster in der Testautomatisierung anwenden

Testautomatisierung ist eine Tätigkeit der Softwareentwicklung. Daher sind Entwurfsprinzipien und Entwurfsmuster für einen TAE genauso wichtig wie für einen Softwareentwickler.

**Objektorientierte Programmierprinzipien**

Es gibt vier Hauptprinzipien der objektorientierten Programmierung: Kapselung, Abstraktion, Vererbung und Polymorphismus.

**SOLID-Prinzipien**

Es ist ein Akronym aus Single Responsibility, Open-Closed, Liskov Substitution, Interface Substitution und Dependency Inversion. Diese Prinzipien verbessern die Lesbarkeit, Wartbarkeit und Skalierbarkeit des Codes.

**Entwurfsmuster**

Von den vielen Entwurfsmustern sind drei für TAEs am wichtigsten.

Das Facade Pattern (Fassade) verbirgt Implementierungsdetails, um nur das offenzulegen, was die Tester in den Testfällen erstellen müssen, und das Singleton-Muster wird häufig verwendet, um sicherzustellen, dass es nur einen Treiber gibt, der mit dem SUT kommuniziert.

Im Page Object Model wird eine Klassendatei erstellt, die als Seitenmodell (Page Model) bezeichnet wird. Wann immer sich die Struktur des SUT ändert, muss der TAE nur an einer Stelle Aktualisierungen vornehmen, nämlich am Locator innerhalb eines Seitenmodells, anstatt die Locatoren in jedem Testfall zu aktualisieren.

Das Flow Model Pattern ist eine Erweiterung des Page Object Model. Es führt eine zusätzliche Fassade über das Page Object Model ein, die alle Benutzeraktionen speichert, die mit den Seitenobjekten interagieren. Durch die Einführung eines Doppelfassaden-Designs bietet das Flow Model Pattern eine verbesserte Abstraktion und Wartbarkeit, da Testschritte in mehreren Testskripten wiederverwendet werden können.

# 4 Implementierung der Testautomatisierung - 150 Minuten (K4)

**Schlüsselbegriffe**

Risiko, Testvorrichtung

**Lernziele für Kapitel 4: Die Lernenden können ...**

## 4.1 Entwicklung der Testautomatisierung

TAE-4.1.1 (K3) ... Richtlinien anwenden, die eine effektive Pilotierung und Implementierung der Testautomatisierung unterstützen

## 4.2 Risiken im Zusammenhang mit der Entwicklung der Testautomatisierung

TAE-4.2.1 (K4) ... Risiken bei der Einführung analysieren und Strategien zur Risikominderung für die Testautomatisierung planen

## 4.3 Wartbarkeit einer Testautomatisierungslösung

TAE-4.3.1 (K2) ... erläutern, welche Faktoren die Wartbarkeit einer Testautomatisierungslösung unterstützen und beeinflussen

# 4.1 Entwicklung der Testautomatisierung

## 4.1.1 Richtlinien anwenden, die eine effektive Pilotierung und Implementierung der Testautomatisierung unterstützen

Es ist wichtig, den Umfang der Validierung für ein Pilotprojekt zur Testautomatisierung zu definieren. Die Durchführung eines Pilotprojekts nimmt nicht viel Zeit in Anspruch, aber das Ergebnis kann einen erheblichen Einfluss auf die Richtung des Projekts haben.

Auf der Grundlage der gesammelten Informationen über das SUT und die Anforderungen an das Projekt sollten die folgenden Punkte bewertet werden, um Richtlinien für die Optimierung der Testautomatisierung aufzustellen:

- Programmiersprache(n), die verwendet werden soll(en)
- Geeignete kommerzielle Standardsoftware/Open-Source-Werkzeuge
- Abzudeckende Teststufen
- Ausgewählte Testfälle
- Testfall-Entwicklungsansatz

Auf der Grundlage der oben aufgeführten Punkte können die TAEs einen ersten Ansatz definieren, der verfolgt werden soll. Auf der Grundlage der Anforderungen können mehrere verschiedene erste Prototypen erstellt werden, um die Vor- und Nachteile der verschiedenen Ansätze aufzuzeigen. Auf dieser Grundlage können die TAEs entscheiden, welchen Pfad sie weiterverfolgen wollen.

Die Festlegung von Zeitplänen ist wichtig, um die Planungsziele einzuhalten und den Erfolg des Pilotprojekts zu gewährleisten. Eine allgemeine Empfehlung lautet, den Fortschritt des Pilotprojekts regelmäßig zu überprüfen, um etwaige Risiken zu erkennen und zu mindern.

Während des Pilotprojekts wird auch empfohlen, die Lösung und den bereits implementierten Code in das CI/CD-System zu integrieren. Dadurch können frühzeitig Probleme aufgedeckt werden, entweder im SUT, in der TAS oder in der Gesamtintegration der verschiedenen Werkzeuge innerhalb der Organisation.

Wenn die Zahl der Testfälle wächst, können TAEs darüber nachdenken, das anfängliche CI/CD-Setup zu ändern, um die Tests auf unterschiedliche Weise und zu unterschiedlichen Zeiten auszuführen.

Während des Pilotprojekts müssen auch andere nicht-technische Aspekte bewertet werden, wie z. B.:

- Wissen und Erfahrung der Teammitglieder
- Die Teamstruktur
- Lizenzen und Organisationsregeln
- Die Art der geplanten Tests und die angestrebten Teststufen, die bei der Automatisierung der Testfälle abgedeckt werden sollen.

Nach Abschluss des Pilotprojekts sollte der Aufwand von TAEs und Testmanagern bewertet werden, um den Erfolg oder die Fehlerwirkung zu beurteilen und eine entsprechende Entscheidung zu treffen.

## 4.2 Risiken im Zusammenhang mit der Entwicklung der Testautomatisierung

### 4.2.1 Risiken bei der Einführung analysieren und Strategien zur Risikominderung für die Testautomatisierung planen

Die Schnittstelle zwischen dem TAF und dem SUT muss als Teil des Architekturentwurfs berücksichtigt werden. Dann können die Werkzeuge für die Paketierung, Testprotokollierung und den Testrahmen ausgewählt werden.

Während der Implementierung des Piloten muss die Erweiterung und Wartung des Codes für die Testautomatisierung berücksichtigt werden. Dies sind entscheidende Faktoren in der Phase der Pilotevaluierung und können die endgültige Entscheidung erheblich beeinflussen.

Aus dem Pilotprojekt lassen sich verschiedene Risiken für den Einsatz ableiten:

- Offene Firewall-Ports
- Ressourcennutzung (z. B. CPU und RAM)

Es müssen Vorbereitungen für Einsatzrisiken wie Firewall-Probleme, Ressourcennutzung, Netzwerkverbindung und Zuverlässigkeit getroffen werden. Diese Aspekte sind nicht unbedingt mit der Testautomatisierung verbunden, aber TAEs müssen sicherstellen, dass alle Bedingungen erfüllt sind, um zuverlässige und nützliche Quality Gates in ihrem Entwicklungsprozess bereitzustellen.

Die Verwendung realer Geräte für die Testautomatisierung von mobilen Tests ist ein Beispiel dafür. Mobile Geräte müssen eingeschaltet sein, über eine ausreichende Batterieleistung verfügen, um während des Tests zu funktionieren, mit einem Netzwerk verbunden sein und Zugriff auf das SUT haben.

Zu den technischen Risiken bei der Implementierung können gehören:

- Packaging
- Protokollierung
- Strukturierung von Tests
- Aktualisierung

**Packaging**

Beim Packaging muss berücksichtigt werden, dass die Versionskontrolle der Testautomatisierung genauso wichtig ist wie für das SUT. Testmittel müssen möglicherweise in ein Repository hochgeladen werden, um sie innerhalb einer Organisation zu teilen, entweder vor Ort oder in der Cloud.

**Protokollierung**

Die Testprotokollierung liefert die meisten Informationen über Testergebnisse. Es gibt mehrere Ebenen der Testprotokollierung, die alle aus verschiedenen Gründen für die Testautomatisierung nützlich sind:

- Fatal: Diese Stufe wird verwendet, um Fehlerereignisse zu protokollieren, die zum Abbruch der Testdurchführung führen können.
- Fehler: Diese Stufe wird verwendet, wenn eine Bedingung oder Interaktion fehlgeschlagen ist und somit auch der Testfall fehlgeschlagen ist.
- Warnung: Diese Stufe wird verwendet, wenn eine unerwartete Bedingung/Aktion auftritt, die aber den Ablauf des Testfalls nicht unterbricht.
- Info: Diese Ebene wird verwendet, um grundlegende Informationen über einen Testfall und das, was während der Testdurchführung passiert, anzuzeigen.
- Debug: Diese Ebene wird verwendet, um ausführungsspezifische Details zu speichern, die nicht für grundlegende Protokolle erforderlich sind, aber bei der Untersuchung eines fehlgeschlagenen Tests nützlich sind.
- Trace: Diese Ebene ist ähnlich wie Debug, enthält aber noch mehr Informationen.

**Strukturierung von Tests**

Der wichtigste Teil der TAS ist der Testrahmen und die darin enthaltenen Testvorrichtungen, die für den Testlauf zur Verfügung stehen müssen. Die Testvorrichtungen bieten Freiheit bei der Steuerung einer Testumgebung und der Testdaten. Für die Testdurchführung können Vorbedingungen und Nachbedingungen definiert werden, und die Testfälle können auf verschiedene Weise in Testsuiten gruppiert werden. Diese Aspekte sind auch für die Auswertung während eines Pilotprojekts wichtig. Darüber hinaus ermöglichen die Testvorrichtungen die Erstellung von automatisierten Tests, die wiederholbar und atomar sind.

**Aktualisierung**

Eines der häufigsten technischen Risiken sind automatische Aktualisierungen der Testrahmen (z. B. Agenten) und Versionsänderungen auf den Geräten. Diese Risiken können durch eine passende Stromversorgung, richtige Konnektivität und geeignete Gerätekonfigurationspläne entschärft werden.

## 4.3 Wartbarkeit einer Testautomatisierungslösung

### 4.3.1 Erläutern, welche Faktoren die Wartbarkeit einer Testautomatisierungslösung unterstützen und beeinflussen

Die Wartbarkeit wird in hohem Maße von Programmierstandards und den gegenseitigen Erwartungen der TAEs beeinflusst.

Eine goldene Regel ist der Versuch, die Clean-Code-Prinzipien von Robert C. Martin zu befolgen (Robert C. Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", 2008).

Kurz gesagt, betonen die Grundsätze für Clean Code die folgenden Punkte:

- Verwendung einer gemeinsamen Namenskonvention für Klassen, Methoden und Variablen mit aussagekräftige Namen.
- Verwendung einer logischen und gemeinsamen Projektstruktur.
- Vermeidung von Hardcoding.
- Vermeidung zu vieler Eingabeparameter für Methoden.
- Vermeidung von zu langen und komplexen Methoden.
- Verwendung von Protokollierung.
- Verwendung von Entwurfsmustern, wo sie nützlich und erforderlich sind.
- Fokussierung auf Testbarkeit.

Namenskonventionen sind sehr hilfreich, um den Zweck einer bestimmten Variablen zu identifizieren. Verständliche Variablennamen wie "loginButton", "resetPasswordButton" helfen den TAEs zu verstehen, welche Komponente zu verwenden ist.

Hardcoding ist der Prozess, bei dem Werte in die Software eingebettet werden, ohne dass sie direkt geändert werden können. Es kann durch datengetriebene Tests vermieden werden, so dass die Testdaten aus einer gemeinsamen Quelle stammen, die leichter zu pflegen ist. Hardcoding verkürzt die Entwicklungszeit, ist aber nicht empfehlenswert, da sich die Daten häufig ändern können und die Pflege zeitaufwendig ist. Die Verwendung von Konstanten für Variablen, die sich voraussichtlich nicht häufig ändern, wird ebenfalls empfohlen. Auf diese Weise können die zu pflegenden Quellen und Stellen reduziert werden.

Die Nutzung von Entwurfsmustern ist ebenfalls sehr zu empfehlen. Die Entwurfsmuster - wie in *Kapitel 3.1.5* beschrieben - ermöglichen die Implementierung eines strukturierten und gut wartbaren Codes für die Testautomatisierung, sofern die Entwurfsmuster richtig verwendet werden.

Um eine hohe Qualität des Codes für die Testautomatisierung zu gewährleisten, wird der Einsatz von statischen Analysatoren empfohlen. Code-Formatierer, wie sie in IDEs üblich sind, verbessern die Lesbarkeit des Codes für die Testautomatisierung.

Abgesehen von den Prinzipien des Clean Code wird empfohlen, eine vereinbarte Branchstruktur und -strategie in der Versionskontrolle zu verwenden. Die Verwendung unterschiedlicher Branches für Features, Releases und Fehlerbehebungen ist hilfreich, um die Inhalte der Branches zu verstehen.

# 5 Implementierungs- und Deployment-Strategien für die Testautomatisierung - 90 Minuten (K3)

**Schlüsselbegriffe**

Vertragstest

**Lernziele für Kapitel 5: Die Lernenden können ...**

## 5.1 Integration automatisierter Tests in CI/CD-Pipelines verstehen

TAE-5.1.1 (K3) ... Testautomatisierung auf verschiedenen Teststufen innerhalb von Pipelines anwenden

TAE-5.1.2 (K2) ... Konfigurationsmanagement für Testmittel erklären

TAE-5.1.3 (K2) ... Abhängigkeiten bei der Testautomatisierung für eine API-Infrastruktur erläutern

# 5.1 Integration automatisierter Tests in CI/CD-Pipelines verstehen

## 5.1.1 Testautomatisierung auf verschiedenen Teststufen innerhalb von Pipelines anwenden

Einer der Hauptvorteile der Testautomatisierung besteht darin, dass die implementierten Tests unbeaufsichtigt laufen können, was sie zu idealen Kandidaten für den Einsatz in Pipelines macht. Dies kann über CI/CD-Pipelines erfolgen oder über eine eigene Pipeline, in der die Tests regelmäßig ausgeführt werden.

Die Teststufen sind in der Regel wie folgt integriert:

- Konfigurationstests für TAF/TAS können während des Builds als Unterart der Komponententests betrachtet werden. Diese Tests werden während des Builds eines Testautomatisierungsprojekts (TAF/TAS) ausgeführt und prüfen, ob alle Pfade zu den in den Testskripten verwendeten Dateien korrekt sind, ob die Dateien existieren und sich in den angegebenen Pfaden befinden.
- Komponententests sind Teil des Builds der Pipeline, da sie auf den einzelnen Komponenten ausgeführt werden (z. B. Bibliotheksklassen und Webkomponenten). Sie fungieren als Quality Gates für die Pipeline und sind somit ein wichtiger Bestandteil einer Pipeline für kontinuierliche Integration.
- Komponentenintegrationstests können Teil der kontinuierlichen Integrationspipeline sein, wenn es sich um Tests von Low-Level-Komponenten oder des SUT handelt. In solchen Fällen werden diese Tests und der Komponententest gemeinsam ausgeführt.
- Systemtests können oft in eine kontinuierliche Deployment-Pipeline integriert werden, wo sie als letztes Quality Gate des gelieferten SUT fungieren.
- Systemintegrationstests zwischen verschiedenen Systemkomponenten sind oft Teil einer kontinuierlichen Delivery-Pipeline als Quality Gates. Diese Systemintegrationstests stellen sicher, dass die separat entwickelten Systemkomponenten zusammenarbeiten.

Viele moderne kontinuierliche Integrationssysteme unterscheiden zwischen Build- und Deployment-Phasen der kontinuierlichen Delivery-Pipelines. In diesen Fällen sind Komponententests und Komponentenintegrationstests Teil der ersten Build-Phase. Wenn diese erste Phase erfolgreich verläuft (d. h. Build und Test zusammen), werden die Komponenten/SUT bereitgestellt.

Im Falle von Systemintegrationstests, Systemtests und Abnahmetests gibt es zwei Hauptansätze, um sie in solche Pipelines zu integrieren:

1. Testfälle werden als Teil des Deployments nach der Bereitstellung der Komponente ausgeführt. Dies kann von Vorteil sein, da aufgrund der Testergebnisse das Deployment fehlgeschlagen sein kann und auch wieder rückgängig gemacht werden kann. Allerdings wird ein erneutes Deployment erforderlich, wenn die Tests erneut ausgeführt werden sollen.
2. Testfälle werden in einer separaten Pipeline ausgeführt, die durch ein erfolgreiches Deployment ausgelöst wird. Dies kann von Vorteil sein, wenn zu erwarten ist, dass verschiedene Testsuiten und verschiedenartiger Testautomatisierungcode bei jedem Deployment ausgeführt werden. In diesem Fall fungieren die Tests nicht als Quality Gate. Daher sind andere, in der Regel manuelle Maßnahmen erforderlich, um ein fehlgeschlagenes Deployment rückgängig zu machen. In diesem Fall können einige einfache automatisierte Testskripte zur Überprüfung des Deployments verwendet werden, um sicherzustellen, dass das SUT bereitgestellt ist. Diese automatisierten Testskripte verifizieren die funktionale Eignung des SUT üblicherweise nicht auf breiter Basis.

Pipelines können auch für andere Zwecke der Testautomatisierung verwendet werden, wie z. B.:

- Regelmäßiges Ausführen verschiedener Testsuiten: Eine Regressionstestsuite kann jede Nacht laufen (ein sogenannter Nachtlauf, engl. Nightly Regression), insbesondere bei länger laufenden Testsuiten, so dass das Team am nächsten Morgen ein klares Bild von der Qualität des SUT hat.
- Durchführung nicht-funktionaler Tests: Dies erfolgt entweder als Teil einer kontinuierlichen Deployment-Pipeline oder separat, um bestimmte nicht-funktionale Qualitätsmerkmale des Systems, wie z. B. die Performanz, periodisch zu überwachen.

### 5.1.2 Konfigurationsmanagement für Testmittel erklären

Konfigurationsmanagement ist ein integraler Bestandteil der Testautomatisierung, da die Automatisierung oft in mehreren Testumgebungen und auf verschiedenen Versionen des SUT ausgeführt wird.

Das Konfigurationsmanagement der Testautomatisierung umfasst:

- Konfiguration der Testumgebung
- Testdaten
- Testsuiten/Testfälle

**Konfiguration der Testumgebung**

Jede Testumgebung, die in der Entwicklungspipeline verwendet wird, kann unterschiedliche Konfigurationen haben, z. B. verschiedene URLs oder Anmeldedaten. Die Konfiguration der Testumgebung wird normalerweise zusammen mit den Testmitteln gespeichert. Im Falle einer Testautomatisierung, die für mehrere Projekte oder mehrere TAFs für dasselbe Projekt verwendet wird, kann die Konfiguration der Testumgebung jedoch Teil der gemeinsamen Kernbibliothek oder in einem gemeinsamen Repository sein.

**Testdaten**

Testdaten können auch spezifisch für die Testumgebung oder für das Release und den Funktionsumfang des SUT sein. Wie die Konfiguration der Testumgebung werden auch die Testdaten in der Regel in kleineren TAFs gespeichert, es können aber auch Testdatenmanagement-Systeme verwendet werden.

**Testsuiten/Testfälle**

Eine gängige Praxis ist es, verschiedene Testsuiten für die Testfälle einzurichten, je nach ihrem Zweck, wie z. B. Smoke-Tests oder Regressionstests. Diese Testsuiten werden oft in separaten Teststufen ausgeführt, wobei unterschiedliche Pipelines und Testumgebungen genutzt werden.

Jedes Release des SUT bestimmt ein Feature-Set, das Testfälle und Testsuiten enthält, die die Qualität des jeweiligen Releases bewerten. Es gibt verschiedene Möglichkeiten, diese Testmittel zu handhaben:

- Für jedes Release oder jede Testumgebung kann eine Feature-Toggle-Konfiguration definiert werden. Es gibt Testfälle und Testsuiten zum Testen der einzelnen Funktionalitäten. Das Feature-Toggle kann in den Testmitteln verwendet werden, um zu bestimmen, welche Testsuiten in einem bestimmten Release bzw. einer bestimmten Testumgebung ausgeführt werden sollen.
- Die Testmittel können auch zusammen mit dem SUT in der gleichen Release-Version freigegeben werden. Auf diese Weise gibt es eine exakte Übereinstimmung zwischen der Version des SUT und den Testmitteln, die es testen können. Eine solche Freigabe wird in der Regel über ein Konfigurationsmanagement-System mit Tags oder Branches realisiert.

### 5.1.3 Abhängigkeiten bei der Testautomatisierung für eine API-Infrastruktur erläutern

Bei der Testautomatisierung über APIs ist es wichtig, die folgenden Informationen über Abhängigkeiten zu kennen, um eine geeignete Strategie zu entwickeln:

- API-Verbindungen: Verstehen der Geschäftslogik, die automatisiert getestet werden kann, und der Beziehungen zwischen APIs.
- API-Dokumentation: Dient als Grundlage für die Testautomatisierung mit allen relevanten Informationen (z. B. Parameter, Kopfzeilen und verschiedene Arten von Request-Response von Objekten).

Integrierte automatisierte API-Tests können entweder von den Entwicklern oder den TAEs durchgeführt werden. Mit Shift-Left wird jedoch empfohlen, das Testen auf verschiedenen Teststufen zu unterstützen und aufzuteilen. In ISTQB-CTFL (2023) werden Komponentenintegrationstests und Systemintegrationstests erwähnt, die durch eine bewährte Praxis erweitert werden können, nämlich durch den Vertragstest.

**Vertragstests**

Vertragstests sind eine Art von Integrationstests, bei denen überprüft wird, ob Dienste miteinander kommunizieren können und ob die zwischen den Diensten ausgetauschten Daten mit einem festgelegten Regelwerk übereinstimmen. Durch die Verwendung von Vertragstests wird Kompatibilität zwischen zwei separaten Systemen (z. B. zwei Microservices) sichergestellt, die miteinander kommunizieren. Der Vertragstest geht über die Schemavalidierung hinaus und verlangt von beiden Kommunikationsparteien einen Konsens über die zulässigen Interaktionen, wobei eine Weiterentwicklung im Laufe der Zeit möglich ist. Er erfasst die Interaktionen, die zwischen den einzelnen Diensten ausgetauscht werden, und speichert sie in einem Vertrag, der dann verwendet werden kann, um zu überprüfen, ob sich die Kommunikationsparteien an diesen Vertrag halten. Einer der Hauptvorteile dieser Testart besteht darin, dass Fehlerzustände, die von den zugrunde liegenden Diensten ausgehen, zu einem früheren Zeitpunkt im SDLC gefunden werden können und die Quelle dieser Fehlerzustände leichter identifiziert werden kann.

Beim verbraucherorientierten Ansatz des Vertragstests legt der Verbraucher seine Erwartungen fest und bestimmt, wie der Anbieter auf die Anfragen dieses Verbrauchers reagieren soll. Beim anbietergesteuerten Ansatz für Vertragstests erstellt der Anbieter den Vertrag, der aufzeigt, wie seine Dienste zu verwenden sind.

# 6 Testautomatisierung: Berichtswesen und Metriken - 150 Minuten (K4)

**Schlüsselbegriffe**

Messung, Metrik, Testfortschrittsbericht, Testprotokoll, Testschritt

**Lernziele für Kapitel 6: Die Lernenden können ...**

## 6.1 Erfassung, Analyse und Auswertung von Daten zur Testautomatisierung

TAE-6.1.1 (K3) ... Methoden zur Erfassung von Daten aus der Testautomatisierungslösung und dem System unter Test anwenden

TAE-6.1.2 (K4) ... Daten aus der Testautomatisierungslösung und dem System unter Test analysieren, um die Ergebnisse besser zu verstehen

TAE-6.1.3 (K2) ... erklären, wie ein Testfortschrittsbericht erstellt und veröffentlicht wird

# 6.1 Erfassung, Analyse und Auswertung von Daten zur Testautomatisierung

## 6.1.1 Methoden zur Erfassung von Daten aus der Testautomatisierungslösung und dem System unter Test anwenden

Daten können aus den folgenden Quellen erfasst werden:

- SUT-Protokolle
  - Web/Mobile UI
  - APIs
  - Anwendungen
  - Webserver
  - Datenbankserver
- TAF-Protokolle zur Erstellung eines Prüfprotokolls (engl. audit trail)
- Build-Protokolle
- Deployment-Protokolle
- Produktionsprotokolle zur Überwachung von Daten in der Produktion (siehe *ISTQB-CT-PT, ISTQB®* (2023), Abschnitt 2.3)
  - Performanzüberwachung in der Produktion zur Durchführung von Trendanalysen
  - Performanztestprotokolle in einer Performanztestumgebung (z. B. Last-, Stresstests und Lastspitzentests)
- Screenshots und Bildschirmaufzeichnungen (vom Automatisierungswerkzeug selbst oder von Drittanbietern bereitgestellt)

Da eine TAS im Kern aus automatisierten Testmitteln besteht, können diese erweitert werden, um Informationen über ihre Verwendung bzw. Ausführung aufzuzeichnen. Diese Erweiterungen, können dann von allen übergeordneten automatisierten Testskripten genutzt werden. So wäre beispielsweise eine Erweiterung zur Aufzeichnung der Start- und Endzeit der Testausführung auf alle bestehenden Tests anwendbar.

**Features der Testautomatisierung, die Messungen und die Testberichterstellung unterstützen**

Die Skriptsprachen verschiedener Testwerkzeuge können Informationen vor, während und nach der Testausführung einzelner Tests oder ganzer Testsuiten aufzeichnen und protokollieren, um Messungen und das Berichtswesen zu unterstützen.

Testberichte über eine Reihe von Testläufen sollten über eine Analysefunktion verfügen, die Testergebnisse der vorangegangenen Testläufe berücksichtigt, um Trends aufzuzeigen, wie z. B. Änderungen der Testerfolgsrate.

Die Testautomatisierung erfordert in der Regel eine Automatisierung sowohl der Testausführung als auch der Verifizierung der Tests, wobei Letztere durch den Vergleich bestimmter Elemente der tatsächlichen Ergebnisse mit den erwarteten Ergebnissen erreicht wird. Dieser Vergleich wird am besten von einem Testwerkzeug durchgeführt, das Assertions verwendet. Der Informationsgrad, der als Ergebnis dieses

Vergleichs gemeldet wird, muss berücksichtigt werden. Es ist wichtig, dass der Status des Tests korrekt ermittelt wird (d. h. bestanden oder fehlgeschlagen). Im Falle eines fehlgeschlagenen Tests sind weitere Informationen über die Ursache der Fehlerwirkung erforderlich (z. B. Screenshots).

Unterschiede zwischen den tatsächlichen Ergebnissen und den erwarteten Ergebnissen eines Tests sind nicht immer eindeutig. Eine Werkzeugunterstützung kann helfen, Vergleiche zu erstellen, die z. B. erwartete Unterschiede wie Datumsangaben und Zeiten ignoriert, während alle andere unerwarteten Unterschiede hervorgehoben werden.

**Testprotokollierung**

Testprotokolle sind eine Quelle, die häufig zur Analyse potenzieller Fehlerzustände innerhalb der TAS und des SUT verwendet wird. Im folgenden Abschnitt finden sich Beispiele für die Testprotokollierung, kategorisiert nach TAS und SUT.

**TAS-Protokollierung**

Der Kontext bestimmt, ob das TAF oder die Testausführung für die Protokollierung von Informationen verantwortlich ist. Folgende Inhalte sollten enthalten sein:

- Testfall der gerade ausgeführt wird, einschließlich Start- und Endzeit.
- Der Status der Testausführung, da Fehlerwirkungen zwar leicht in Testprotokollen festgestellt werden können, das TAF aber auch über diese Informationen verfügen und über ein Dashboard berichten sollte. Der Status der Testausführung kann entweder bestanden, fehlgeschlagen oder eine Fehlerwirkung der TAS sein. Der Status kann manchmal nicht eindeutig sein, und es ist wichtig, dass eine Organisation klare und einheitliche Definitionen dafür festlegt. Eine TAS-Fehlerwirkung wird auf Situationen angewendet, in denen der Fehlerzustand nicht im SUT enthalten ist.
- Low-Level-Details des Testprotokolls (z. B. Protokollierung signifikanter Testschritte) einschließlich Zeitinformationen.
- Dynamische Informationen über das SUT (z. B. Speicherlecks), die der Testfall mit Hilfe weiterer Werkzeuge identifizieren konnte. Tatsächliche Ergebnisse und Fehlerwirkungen sollten mit der Testsuite protokolliert werden, die zum Zeitpunkt der Fehlerentdeckung ausgeführt wurde.
- Bei Zuverlässigkeitstests oder Stresstests, bei denen zahlreiche Testzyklen durchgeführt werden, sollte ein Zähler mitlaufen, um leicht feststellen zu können, wie oft die Testfälle ausgeführt wurden.
- Wenn Testfälle zufällige Elemente enthalten (z. B. zufällige Parameter oder zufällige Testschritte bei Zustandsübergangstests), sollte die Zufallszahl/-auswahl protokolliert werden.
- Alle Aktionen, die ein Testfall ausführt, sollten so protokolliert werden, dass die Testprotokolle oder Teile davon wiedergegeben werden können, um den Test mit denselben Testschritten und demselben zeitlichen Ablauf erneut auszuführen. Dies ist nützlich, um eine identifizierte Fehlerwirkung zu reproduzieren und zusätzliche Informationen zu erfassen. Die Informationen zu Testfallaktionen können auch vom SUT protokolliert werden, um sie bei der Reproduktion der vom Kunden identifizierten Fehlerwirkungen zu verwenden. Wenn ein Kunde eine Testsuite durchführt, werden die Testprotokolle aufgezeichnet und können dann vom Entwicklungsteam bei der Behebung eines Fehlerzustands verwendet werden.
- Screenshots können während der Testausführung zur weiteren Verwendung bei der Grundursachenanalyse gespeichert werden.

- Wenn eine Testsuite eine Fehlerwirkung auslöst, sollte die TAS sicherstellen, dass alle für die Analyse des Fehlerzustands erforderlichen Informationen verfügbar/gespeichert sind, ebenso wie alle Informationen über die Fortsetzung der Tests, falls zutreffend. Die TAS sollte alle zugehörigen Crash Dumps und Stack Traces speichern. Außerdem sollten alle Testprotokolle, die überschrieben werden könnten (z. B. werden häufig zyklische Puffer für Testprotokolle auf dem SUT verwendet), an einem Ort gespeichert werden, an dem sie für eine spätere Analyse verfügbar sind.
- Die Verwendung von Farben kann helfen, verschiedene Arten von Informationen in Testprotokollen zu unterscheiden (z. B. Fehlerzustände in Rot und Fortschrittsinformationen in Grün).

**SUT-Protokollierung**

Werden Ergebnisse der Testautomatisierung mit den SUT-Protokollen in Korrelation gesetzt, kann die Identifikation der Grundursache von Fehlerzuständen im SUT und in der TAS unterstützt werden.

- Wenn ein Fehlerzustand im SUT identifiziert wird, sollten alle notwendigen Informationen zur Analyse des Fehlerzustands protokolliert werden, einschließlich Datums- und Zeitstempel, Quellort des Fehlerzustands und Fehlermeldungen.
- Beim Start eines Systems sollten Konfigurationsinformationen in einer Datei protokolliert werden, die z. B. die verschiedenen Software/Firmware-Versionen, die Konfiguration des SUT und die Konfiguration des Betriebssystems enthalten.
- Mit Hilfe der Testautomatisierung können SUT-Protokolle einfach durchsucht werden. Eine Fehlerwirkung, die im Testprotokoll der TAS identifiziert wurde, sollte leicht im Testprotokoll des SUT identifiziert werden können, und umgekehrt, mit oder ohne zusätzliche Werkzeuge. Die Synchronisierung verschiedener Testprotokolle über Zeitstempel erleichtert die Zuordnung der Ereignisse, wenn eine Fehlerwirkung gemeldet wird.

**Integration mit anderen Werkzeugen von Drittanbietern (z. B. Tabellenkalkulationen, XML, Dokumente, Datenbanken und Berichtswerkzeuge)**

Wenn Informationen über die Ausführung automatisierter Testfälle in anderen Werkzeugen für die Verfolgung und Berichterstattung verwendet werden (z. B. zur Aktualisierung der Verfolgbarkeit), ist es möglich, die Informationen in einem Format bereitzustellen, das für diese anderen Werkzeuge von Drittanbietern geeignet ist. Dies wird gewöhnlich durch bereits vorhandene Funktionen der Testwerkzeuge erreicht (z. B. Exportformate für die Testberichte) oder durch die Erstellung benutzerdefinierter Berichte, die in einem für andere Software passenden Format ausgegeben werden.

**Visualisierung von Testergebnissen**

Testergebnisse können mit Hilfe von Diagrammen sichtbar gemacht werden. Damit Entscheidungen auf der Grundlage der gemeldeten Informationen getroffen werden können, sollten farbige Symbole wie z. B. Ampeln verwendet werden, um den Gesamtstatus der Testausführung/Testautomatisierung anzuzeigen. Das Management ist besonders an visuellen Zusammenfassungen der Testergebnisse interessiert, die die Entscheidungsfindung erleichtern. Wenn weitere Informationen benötigt werden, kann man jederzeit noch tiefer ins Detail gehen.

### 6.1.2 Daten aus der Testautomatisierungslösung und dem System unter Test analysieren, um die Ergebnisse besser zu verstehen

Nach der Testausführung ist es wichtig, die Testergebnisse zu analysieren, um mögliche Fehlerwirkung(en) sowohl im SUT als auch in der TAS zu identifizieren. Für eine solche Analyse sind die von der TAS gesammelten Daten primär und die vom SUT gesammelten Daten sekundär.

- Analyse der Daten der Testumgebung zur Unterstützung der richtigen Dimensionierung der Testautomatisierung (z. B. in der Cloud).
  - Cluster und Ressourcen (z. B. CPU und RAM)
  - Testausführung mit einem oder mehreren Browsern (d. h. browserübergreifend)
- Vergleich von Testergebnissen aus früheren Testausführungen.
- Festlegen, wie Webprotokolle zur Überwachung der Softwarenutzung verwendet werden sollen.

Fehlerwirkungen bei der Testausführung müssen analysiert werden, da es potenzielle Probleme sind:

1. Es muss geprüft werden, ob bei den vorherigen Testausführungen dieselbe Fehlerwirkung auftrat. Dies könnte ein bekannter Fehlerzustand entweder im SUT oder in der TAS sein. Die TAS kann so aufgebaut werden, dass sie die historischen Testergebnisse der Testfälle protokolliert und so die Analyse weiter unterstützt.
2. Wenn der Fehlerzustand nicht bekannt ist, sollte der Testfall und das, was er testet, identifiziert werden. Entweder ist der Testfall selbsterklärend oder er kann im Testmanagementsystem anhand seiner während der Testausführung protokollierten ID identifiziert werden.
3. Es ist herauszufinden, in welchem Testschritt des Testfalls die Fehlerwirkung aufgetreten ist. Die TAS protokolliert dies.
4. Die Informationen aus dem Testprotokoll bezüglich des Zustands des SUT sind zu analysieren und es muss festgestellt werden, ob dieser mit den erwarteten Ergebnissen übereinstimmt. Dazu können Screenshots, API- und Netzwerkprotokolle oder sonstige aufgezeichnete Protokolle, die den Zustand des SUT kenntlich machen, ausgewertet werden.
5. Wenn der Zustand des SUT nicht den Erwartungen entspricht, sollte ein Fehlerbericht im Fehlermanagementsystem protokolliert werden. Dabei sollten auch alle notwendigen Fehlerinformationen und Protokolle hinterlegt werden, die den Fehlerzustand dokumentieren.

Bei einer Fehlerwirkung ist es möglich, dass das tatsächliche Ergebnis und das erwartete Ergebnis des SUT übereinstimmen. In diesem Fall enthält die TAS entweder selbst einen Fehlerzustand, der behoben werden muss, oder der Unterschied der Ergebnisse ist nicht erkennbar.

Ein anderes Problem kann auftreten, wenn die Testumgebung während des Testlaufs nicht oder nur teilweise verfügbar ist. In diesem Fall könnten alle Testfälle fehlgeschlagen sein, entweder aufgrund desselben Fehlerzustands oder wenn Teile des Systems wegen scheinbar echter Fehler ausgefallen sind. Um die Grundursache für solche Fehlerzustände zu identifizieren, können die SUT-Protokolle analysiert werden, die zeigen, ob es zum Zeitpunkt des Testlaufs Ausfälle der Testumgebung gab.

Wenn das SUT Audit-Protokolle für Benutzerinteraktionen (d. h. UI-Sitzungen oder API-Aufrufe) implementiert, hilft dies bei der Analyse von Testergebnissen. In der Regel wird der Interaktion eine eindeutige ID hinzugefügt, die für jeden nachfolgenden Aufruf und jede Integration in das System gleich bleibt. Auf diese Weise kann das Verhalten des Systems bei Kenntnis der eindeutigen ID einer Anfrage/Interaktion beobachtet und zurückverfolgt werden.

Diese eindeutige ID wird gewöhnlich als Korrelations-ID oder Trace-ID bezeichnet. Sie kann von der TAS protokolliert werden, um die Analyse von Testergebnissen zu erleichtern.

### 6.1.3 Erklären, wie ein Testfortschrittsbericht erstellt und veröffentlicht wird

Testprotokolle enthalten detaillierte Informationen zu den Testschritten, den durchzuführenden Aktionen und den erwarteten Reaktionen eines Testfalls und/oder einer Testsuite. Allerdings können die Testprotokolle allein keinen ausreichenden Überblick über die gesamten Testergebnisse geben. Hierfür ist eine Testberichterstattung erforderlich. Nach der Ausführung einer Testsuite muss ein übersichtlicher Testfortschrittsbericht erstellt und veröffentlicht werden. Hierfür kann ein Berichtsgenerator verwendet werden.

**Inhalt eines Testfortschrittsberichts**

Der Testfortschrittsbericht muss die Testergebnisse, Informationen über das SUT sowie die Dokumentation der Testumgebung, in der die Tests ausgeführt wurden, in einem für alle Stakeholder geeigneten Format enthalten.

Es ist wichtig zu wissen, welche Testfälle fehlgeschlagen sind und aus welche Gründen. Um die Fehlersuche zu erleichtern, ist es wichtig, den Verlauf der Testausführungen zu kennen und zu wissen, wer ihn gemeldet hat (d. h. gewöhnlich die Person, die die Testberichte erstellt oder zuletzt aktualisiert hat). Diese verantwortliche Person muss die Ursache der Fehlerwirkung untersuchen, den damit verbundenen Fehlerzustand melden, die Behebung des Fehlers weiterverfolgen und prüfen, ob sie korrekt umgesetzt wurde.

Die Testberichterstattung dient auch der Diagnose von Fehlerwirkungen der TAF-Komponenten.

**Veröffentlichung der Testberichte**

Ein Testbericht sollte für alle relevanten Stakeholder veröffentlicht werden. Er kann auf einer Webseite, in der Cloud oder lokal abgelegt werden, oder er kann an eine Mailingliste geschickt oder in ein anderes Werkzeug wie ein Testmanagementsystem hochgeladen werden. So wird sichergestellt, dass Berichte überprüft und analysiert werden, sofern der Empfang dieser Nachricht via E-Mail oder via Chatbot abonniert wurde.

Möglicherweise ist es sinnvoll, problematische Teile des SUT zu identifizieren, um gezielte Trendanalysen zu erstellen. Zu diesem Zweck können ältere Testberichte aufbewahrt werden, um Statistiken über Testfälle oder Testsuiten, die häufig Regressionen aufzeigen, zu sammeln.

**Zu den Stakeholdern, an die berichtet werden sollten, gehören:**

- Management-Stakeholder
  - Typische Rollen: Lösungs- oder Unternehmensarchitekt, Projekt-/Delivery-Manager, Programmmanager, Testmanager oder Leitender Testmanager
- Operative Stakeholder
  - Typische Rollen: Product Owner/Manager, Business Representative oder Business-Analysten
- Technische Stakeholder
  - Typische Rollen: Teamleiter, Scrum Master, Webadministrator, Datenbankentwickler/-administrator, Testleiter, TAEs, Tester oder Entwickler

Testberichte können sich je nach Empfänger in Inhalt und Detailtiefe unterscheiden. Während die technischen Stakeholder eher an Details auf den unteren Ebenen interessiert sind, konzentriert sich das Management vor allem auf Trends, z. B. wie viele Testfälle seit dem letzten Testlauf hinzugekommen sind, auf Veränderungen im Verhältnis zwischen bestandenen und fehlgeschlagenen Tests sowie auf die Zuverlässigkeit der TAS und des SUT. Operative Stakeholder legen in der Regel mehr Wert auf produktnutzungsbezogene Metriken.

**Erstellung von Dashboards**

Moderne Reporting-Werkzeuge bieten verschiedene Berichtsoptionen in Form von Dashboards, farbigen Diagrammen, detaillierten Protokollsammlungen oder einer automatisierten Testprotokollanalyse. Auf dem Markt gibt es eine Vielzahl von Werkzeugen zur Auswahl.

Diese Werkzeuge unterstützen die Datenaggregation aus Quellen wie Testprotokollen der Pipeline-Ausführung, Projektmanagementwerkzeugen und Code-Repositories. Die von diesen Werkzeugen bereitgestellte Visualisierung der Daten hilft Stakeholdern, Trends zu erkennen und darauf basierend entsprechende Entscheidungen zu treffen. Zu diesen Trends gehören Fehlerhäufungen, Zunahme/Abnahme der Fehlerverbreitung in bestimmten Testumgebungen, Verschlechterung des SUT-Leistungszustands oder die Zuverlässigkeit der Builds.

**Analyse von Testprotokollen durch künstliche Intelligenz / maschinelles Lernen**

Seit einigen Jahren enthalten einige Werkzeuge zur Testautomatisierung Algorithmen des maschinellen Lernens (ML) oder basieren sogar auf diesen. Die automatisierte Analyse großer Datenmengen in Testprotokollen kann dem TAE helfen, den Zeitaufwand zum Finden fehlerhafter Locator, zur Analyse der Gründe für Fehlerwirkungen (d. h., handelt es sich um einen Fehler im SUT oder in der TAS?) oder auch zur Gruppierung bestimmter Fehler für die Testberichterstattung zu reduzieren (siehe ISTQB-CT-AI (2021)).

# 7 Verifizierung der Testautomatisierungslösung - 135 Minuten (K3)

**Schlüsselbegriffe**

statische Analyse

**Lernziele für Kapitel 7: Die Lernenden können ...**

## 7.1 Verifizierung der Testautomatisierungs-Infrastruktur

TAE-7.1.1 (K3) ... die Verifizierung der Testautomatisierungsumgebung planen, einschließlich der Einrichtung der Testwerkzeuge

TAE-7.1.2 (K2) ... das korrekte Verhalten für ein gegebenes automatisiertes Testskript und/oder eine Testsuite erläutern

TAE-7.1.3 (K2) ... identifizieren, wo die Testautomatisierung unerwartete Ergebnisse liefert

TAE-7.1.4 (K2) ... erläutern, wie statische Analyse helfen kann, die Qualität des Codes für die Testautomatisierung zu verbessern

# 7.1 Verifizierung der Testautomatisierungs-Infrastruktur

## 7.1.1 Die Verifizierung der Testautomatisierungsumgebung planen, einschließlich der Einrichtung der Testwerkzeuge

Vor ihrem Einsatz sollte die Testautomatisierungsumgebung und alle weiteren Komponenten der TAS dahingehend verifiziert werden, ob sie sich wie erwartet verhalten. Zur Überprüfung der Komponenten der Testautomatisierungsumgebung können verschiedene Schritte unternommen werden, die im Folgenden näher erläutert werden.

**Installation, Einrichtung, Konfiguration und Anpassung von Testwerkzeugen**

Die TAS besteht aus vielen Komponenten, die eine zuverlässige und wiederholbare Leistung sicherstellen müssen. Den Kern der TAS bilden dabei die ausführbaren Komponenten, die entsprechenden Funktionsbibliotheken sowie die unterstützenden Daten- und Konfigurationsdateien. Der Prozess der Konfiguration einer TAS kann von der Verwendung automatischer Installationsskripte bis hin zum manuellen Anlegen von Konfigurationsdateien in passenden Ordnern reichen. Ähnlich wie bei Betriebssystemen und anderer Software gibt es für Testwerkzeuge regelmäßig Service Packs bzw. optionale und erforderliche Add-ins, um die Kompatibilität mit einer bestimmten SUT-Umgebung zu gewährleisten.

Eine automatisierte Installation oder auch das Kopieren aus einem zentralen Repository hat Vorteile. Dies garantiert, dass Tests auf unterschiedlichen SUTs mit der gleichen Version und der gleichen Konfiguration der TAS durchgeführt werden, falls dies gefordert wird. Upgrades der TAS können über das Repository durchgeführt werden. Die Verwendung des Repository und das Verfahren für ein Upgrade auf eine neue Version der TAS sollte dem für die in der Entwicklung genutzten Werkzeuge entsprechen.

**Wiederholbarkeit bei Setup/Teardown der Testumgebung**

Eine TAS wird auf einer Vielzahl von Systemen und Servern implementiert und soll CI/CD-Pipelines unterstützen. Um sicherzustellen, dass die TAS in jeder Testumgebung ordnungsgemäß arbeitet, ist ein systematischer Ansatz für das Setup und Teardown der TAS in einer bestimmten Testumgebung erforderlich. Dies ist dann erreicht, wenn der Build und Rebuild der TAS keine erkennbaren Unterschiede in der Verhaltensweise innerhalb der Testumgebung und über mehrere Testumgebungen hinweg aufweist. Durch Konfigurationsmanagement der Komponenten der TAS stellt man sicher, dass bestimmte Konfigurationen zuverlässig erstellt werden können. Die Komponenten, aus denen die TAS besteht, sollten so dokumentiert werden, dass offensichtlich ist, welche Aspekte der TAS betroffen sind oder angepasst werden müssen, wenn sich die SUT-Umgebung ändert.

**Konnektivität der internen und externen Systeme/Schnittstellen**

Sobald eine TAS in einer bestimmten SUT-Umgebung installiert ist und bevor das SUT benutzt wird, sollten eine Reihe von Prüfungen oder Vorbedingungen der TAS durchgeführt werden, um sicherzustellen, dass die Verbindung zu internen und externen Systemen und Schnittstellen verfügbar ist. Es ist zum Beispiel eine bewährte Verfahrensweise, sich bei Servern anzumelden, Testautomatisierungswerkzeuge zu starten, zu überprüfen, ob die Testautomatisierungswerkzeuge auf das SUT zugreifen können, die Konfigurationseinstellungen manuell zu inspizieren und dafür zu sorgen, dass die Berechtigungen für die Testprotokollierung und Testberichterstattung zwischen Systemen korrekt gesetzt sind. Die Festlegung von Vorbedingungen für die Testautomatisierung ist wichtig, um sicherzustellen, dass die TAS korrekt installiert und konfiguriert wurde.

**TAF-Komponententests**

Wie in jedem Softwareentwicklungsprojekt üblich, müssen auch die TAF-Komponenten einzeln getestet und überprüft werden. Dies kann funktionale und nicht-funktionale Tests umfassen (z. B. Leistungseffizienz und Ressourcennutzung). So müssen beispielsweise Komponenten, die eine Objektverifizierung auf GUI-Systemen ermöglichen, für eine Vielzahl von Objektklassen getestet werden, um festzustellen, ob die Objektverifizierung korrekt funktioniert. Ebenso sollten Testprotokolle und Testberichte genaue Informationen über den Status der Testautomatisierung und das Verhalten des SUT liefern. Beispiele für nicht-funktionale Tests sind Untersuchungen zur Verschlechterung der Performanz sowie die Überwachung der Nutzung der Systemressourcen, um dadurch Fehlerzustände wie Speicherlecks und mangelnde Interoperabilität von Komponenten innerhalb und/oder außerhalb des TAF bemerken zu können.

### 7.1.2 Das korrekte Verhalten für ein gegebenes automatisiertes Testskript und/oder eine Testsuite erläutern

Automatisierte Testsuiten müssen auf Vollständigkeit, Konsistenz und korrektes Verhalten geprüft werden. Es können verschiedene Arten von Verifizierungen durchgeführt werden, um sicherzustellen, dass die automatisierte Testsuite zu einem bestimmten Zeitpunkt verfügbar ist, oder um festzustellen, ob sie für den Einsatz geeignet ist.

Zur Verifizierung der automatisierten Testsuite können verschiedene Schritte unternommen werden. Dazu gehören:

- Überprüfung der Zusammensetzung der Testsuite.
- Verifizierung neuer Tests, die sich auf neue Features des TAF konzentrieren.
- Berücksichtigung der Wiederholbarkeit von Tests.
- Berücksichtigung der Intrusivität von Testautomatisierungswerkzeugen.

Jeder dieser Punkte wird im Folgenden ausführlicher erläutert.

**Überprüfung der Zusammensetzung der Testsuite**

Es müssen die Vollständigkeit (z. B. alle Testfälle haben die erwarteten Ergebnisse und die Testdaten sind vorhanden) und die korrekte Version des TAF und des SUT überprüft werden.

**Verifizierung neuer Tests, die sich auf neue Features des TAF konzentrieren**

Wenn ein neues Feature des TAF zum ersten Mal in Testfällen verwendet wird, sollte es genau verifiziert und überwacht werden, um sicherzustellen, dass die Funktion korrekt funktioniert.

**Berücksichtigung der Wiederholbarkeit von Tests**

Bei der Wiederholung von Tests sollten die Testergebnisse immer identisch sein. Testfälle in der Testsuite, die kein zuverlässiges Testergebnis liefern (z. B. aufgrund von Race Conditions), sollten aus der aktiven automatisierten Testsuite herausgenommen und separat analysiert werden, um die Grundursache zu finden. Andernfalls entsteht für diese Testläufe wiederholt zusätzlicher Aufwand, um die immer gleiche Fehlerwirkung zu analysieren.

**Berücksichtigung der Intrusivität von Testautomatisierungswerkzeugen**

Die TAS ist oft eng mit dem SUT gekoppelt. Dies geschieht, um eine bessere Kompatibilität auf der Interaktionsebene zu erreichen. Eine solche enge Integration kann jedoch auch zu nachteiligen Ergebnissen führen. Befindet sich die TAS beispielsweise in der SUT-Umgebung, kann sich die Funktionalität des SUT von der bei manuell durchgeführten Tests unterscheiden, ebenfalls kann dies Auswirkungen auf die Performanz haben.

Ein hoher Grad an Intrusion kann Fehlerwirkungen beim Testen hervorrufen, die in der Produktionsumgebung nicht auftreten. Wenn dies zum Fehlschlagen der automatisierten Tests führt, kann dadurch das Vertrauen in die TAS drastisch sinken. Entwickler können es für erforderlich halten, dass Fehlerwirkungen, die durch Testautomatisierung identifiziert wurden, manuell reproduziert werden - falls möglich -, um die Analyse zu unterstützen.

### 7.1.3 Identifizieren, wo die Testautomatisierung unerwartete Ergebnisse liefert

Wenn ein Testskript unerwartet erfolgreich ist bzw. unerwartet fehlschlägt, muss eine Grundursachenanalyse durchgeführt werden. Dazu gehört die Prüfung der Testprotokolle, der Testdaten, der Performanz sowie des Setup und Teardown des Testskripts.

Es ist ebenfalls hilfreich, mehrere isolierte Tests durchzuführen. Intermittierende Fehlerwirkungen sind schwierig zu analysieren. Der Fehlerzustand kann im Testfall, im SUT, im TAF, in der Hardware oder im Netzwerk liegen. Die Überwachung der Systemressourcen kann Hinweise auf die Grundursache liefern. Die Analyse der Testprotokolldateien des Testfalls, des SUT und des TAF kann helfen, die Grundursache des Fehlerzustands zu identifizieren. Auch Debugging kann notwendig sein. Um die Grundursache zu identifizieren, kann die Unterstützung eines Testanalysten, Business-Analysten, Entwicklers oder Systemingenieurs erforderlich sein.

Das Vorhandensein aller Assertions muss überprüft werden. Fehlende Assertions können zu uneindeutigen Testergebnissen (inconclusive) führen.

### 7.1.4 Erläutern, wie die statische Analyse helfen kann, die Qualität des Codes für die Testautomatisierung zu verbessern

Die statische Analyse des Quellcodes kann helfen, Schwachstellen und Fehlerzustände im Programmcode zu finden. Dies kann das SUT oder das TAF umfassen.

Automatisierte Scans können den Code inspizieren, um Risiken zu mindern. Dies ermöglicht ein Review des SUT auf Fehlerzustände und stellt sicher, dass die Codequalitätsstandards eingehalten und angewendet werden. Dies kann als proaktive Technik zur Erkennung von Fehlerzuständen betrachtet werden und spielt eine wichtige Rolle bei DevSecOps-Implementierungen (z. B. DevOps mit Schwerpunkt auf Sicherheit). Diese Scans werden in einem frühen Stadium des SDLC mittels Pipelines durchgeführt, um den Entwicklungsteams ein unmittelbares Feedback zu geben.

Die Meldungen über Fehlerzustände werden in der Regel in Schweregrade wie kritisch, hoch, mittel oder niedrig eingestuft, so dass die Entwicklungsteams die Möglichkeit haben, die zu behebenden Fehlerzustände zu priorisieren. Bestimmte Werkzeuge für die statische Analyse können auch Vorschläge für Codekorrekturen zur Behebung der gefundenen Fehlerzustände liefern. Dabei wird den Entwicklungsteams eine Kopie der beanstandeten Codezeilen vorgelegt und den Entwicklern eine mögliche Abhilfemaßnahme zur Umsetzung vorgeschlagen.

Darüber hinaus helfen diese Werkzeuge den TAEs bei der Messung der Qualität, schlagen Bereiche vor, in denen der Code kommentiert werden sollte, verbessern das Codedesign für eine optimierte Ressourcenbehandlung (z. B. Verwendung von Try/Catch-Blöcken und besseren Schleifenstrukturen) und entfernen mangelhafte Bibliotheksaufrufe.

Da Werkzeuge zur Testautomatisierung Programmiersprachen verwenden, besteht das Risiko, dass unzulänglicher Code für die Testautomatisierung in den SDLC eingebracht werden kann. Ein einfaches Beispiel: Eine gängige Praxis bei der Testautomatisierung ist die Verwendung eines Benutzernamens und eines Passworts. Es ist denkbar, dass ein TAE das Passwort fälschlicherweise im Klartext in ein oder mehrere Testskripte einfügt.

Werkzeuge für die statische Analyse können bei der Testautomatisierung von Vorteil sein. Sie können verwendet werden, um den Code für die Testautomatisierung auf IT-Sicherheitsverstöße, wie z. B. ein Klartext-Passwort innerhalb des Codes, zu analysieren. Werkzeuge für die statische Analyse unterstützen viele Programmiersprachen, darunter auch die in der Testautomatisierung verwendeten. Daher ist es zwingend erforderlich, dass der TAE die bewährten Verfahren zum Scannen von Code auch auf den Code der Testautomatisierung ausweitet. Auch wenn der Code für die Testautomatisierung nicht notwendigerweise mit der gesamten Software bereitgestellt wird, gibt es eindeutig potenzielle Schwachstellen, wenn das Passwort in einem begleitenden Testskript für die Testautomatisierung entdeckt wird (siehe ISTQB-CT-SEC (2016)).

# 8 Kontinuierliche Verbesserung - 210 Minuten (K4)

**Schlüsselbegriffe**

Schemavalidierung, Testhistogramm

**Lernziele für Kapitel 8: Die Lernenden können ...**

## 8.1 Kontinuierliche Verbesserungsmöglichkeiten bei der Testautomatisierung

TAE-8.1.1 (K3) ... Möglichkeiten zur Verbesserung von Testfällen durch Datensammlung und Datenanalyse entdecken

TAE-8.1.2 (K4) ... technische Aspekte einer eingesetzten Testautomatisierungslösung analysieren und Verbesserungen empfehlen

TAE-8.1.3 (K3) ... automatisierte Testmittel zwecks Anpassung an Updates des System unter Test restrukturieren

TAE-8.1.4 (K2) ... Möglichkeiten für den Einsatz von Testautomatisierungswerkzeugen zusammenfassen

# 8.1 Kontinuierliche Verbesserungsmöglichkeiten bei der Testautomatisierung

## 8.1.1 Möglichkeiten zur Verbesserung von Testfällen durch Datensammlung und Datenanalyse entdecken

Die Datensammlung und -analyse kann unter Berücksichtigung verschiedener Arten von Daten mit den nachfolgend beschriebenen Ansätzen verbessert werden.

**Testhistogramm**

Ein visueller Bericht über Testdaten als Testhistogramm liefert potenzielle Verbesserungsbereiche in Bezug auf Trends bei den Testfalldaten. TAEs können über mögliche Verbesserungsbereiche entscheiden, da viele CI/CD- und Testberichterstattungswerkzeuge über die Fähigkeit verfügen, verschiedene Testergebnisse und ihre jeweiligen Testdaten (z. B. Ausnahmeprotokolle, Fehlermeldungen und Screenshots) anzuzeigen. Das Testhistogramm ermöglicht es den TAEs auch, unzuverlässige Testfälle zu identifizieren und diese durch Refactoring zu verbessern oder deren bisherige Implementierung zu überdenken.

**Künstliche Intelligenz**

Eine weitere Möglichkeit für die Unterstützung des Testens und der Testautomatisierung ist der Einsatz von künstlicher Intelligenz (KI). Zum Beispiel umfassen bei UI-Testfällen die Eingabedaten auch UI-Locator-Werte. Neueste Werkzeuge sind in der Lage, zu erkennen, ob sich ein bestimmter Locator in Bezug auf den bisher verwendeten Wert verändert hat. Auf der Grundlage von ML-Algorithmen und Bilderkennung können die neuen Locator identifiziert und für einen selbstheilenden Algorithmus verwendet werden, um den Testfall zu korrigieren und die geänderten Locator in den Testbericht aufzunehmen. Dies kann die nachfolgenden Schritte wie Änderungen in der Versionskontrolle und die Wartung des Codes beschleunigen.

**Schemavalidierung**

Schemavalidierung kann bei der Analyse von API-Daten (z. B. Eigenschaften, die von Zielendpunkten abgeleitet werden) und bei der Datenbankanalyse (z. B. Validierungsregeln für Eingabefelder, wie zulässige Datentypen und Wertebereiche) angewendet werden.

Mittels Schemavalidierung kann die TAS prüfen, ob eine Antwort mit der tatsächlichen Anforderung aus dem Geschäftsbereich übereinstimmt. Diese Art der Prüfung kann verwendet werden, um festzustellen, ob obligatorische Elemente in der Antwort eines Dienstes vorhanden sind und ob ihr Objekttyp mit dem im Schema definierten übereinstimmt. Im Falle einer Schemaverletzung gibt die TAS das tatsächliche Validierungsergebnis zurück, das den TAEs hilft, die Grundursache des Problems zu identifizieren.

Beispiel: Eine API hat sechs obligatorische Elemente, die in der Antwort enthalten sein müssen. Mit Schemavalidierungswerkzeugen ist es nicht erforderlich, einzelne Assertions zu schreiben, um zu prüfen, ob diese Elemente aus Zeichenketten bestehen und ihre Werte ungleich null sind. Die Schemavalidierung übernimmt diese Prüfungen, wodurch der implementierte Code für die Testautomatisierung viel kürzer und die Erkennung von Fehlerzuständen im Backend-Dienst effizienter wird.

### 8.1.2 Technische Aspekte einer eingesetzten Testautomatisierungslösung analysieren und Verbesserungen empfehlen

Zusätzlich zu den fortlaufenden Wartungsaufgaben, die notwendig sind, um die TAS mit dem SUT synchron zu halten, gibt es viele Möglichkeiten, die TAS zu verbessern. Diese Verbesserungen können vorgenommen werden, um eine Reihe von Vorteilen zu erzielen, darunter eine größere Effizienz (z. B. eine weitere Reduzierung manueller Eingriffe), eine bessere Gebrauchstauglichkeit, zusätzliche Funktionen und eine verbesserte Unterstützung für das Testen. Die Entscheidung darüber, wie die TAS verbessert wird, hängt davon ab, welche Funktionen den größten Mehrwert für ein Projekt erzielen.

Zu den Bereichen einer TAS, die für eine Verbesserung in Frage kommen, zählen die Skripterstellung, die Testausführung, die Verifizierung, die TAA, das TAF, die Setup- und Teardown-Methoden, die Dokumentation, die TAS-Funktionalitäten sowie Updates und Upgrades der TAS. Diese werden im Folgenden ausführlicher beschrieben.

**Skripterstellung**

Ansätze für die Skripterstellung reichen von der linearen Skripterstellung über den datengetriebenen Testansatz bis hin zum anspruchsvolleren schlüsselwortgetriebenen Testansatz, wie in *Kapitel 3.1.4* beschrieben. Es kann sinnvoll sein, den aktuellen Ansatz der TAS für die Skripterstellung für alle neuen automatisierten Tests zu überarbeiten. Der Ansatz kann für alle bestehenden automatisierten Tests nachgerüstet werden oder zumindest für diejenigen, die den größten Aufwand für die Wartung erfordern.

Ein weiterer Bereich für die Verbesserung der Testskripte kann sich auf deren Implementierung konzentrieren, z. B.:

- Bewertung der Überschneidung von Testskript/Testfall/Testschritt, um automatisierte Tests zu konsolidieren. Testfälle, die ähnliche Aktionssequenzen enthalten, sollten Testschritte nicht mehrfach implementieren. Testschritte sollten in eine Funktion zusammengefasst und einer Bibliothek hinzugefügt werden, damit sie wiederverwendet werden können. Diese Bibliotheksfunktionen können dann von verschiedenen Testfällen verwendet werden. Dadurch wird die Wartbarkeit der Testmittel erhöht. Wenn Testschritte nicht identisch, aber ähnlich sind, kann eine Parametrisierung erforderlich sein. Hinweis: Dies ist ein typischer Ansatz für schlüsselwortgetriebene Tests.
- Etablierung eines Prozesses zur Wiederherstellung der TAS und des SUT nach Fehlerwirkungen. Wenn während der Ausführung einer Testsuite eine Fehlerwirkung auftritt, sollte die TAS in der Lage sein, diesen Zustand zurückzusetzen, um mit dem nächstmöglichen Test fortzufahren. Wenn eine Fehlerwirkung im SUT auftritt, muss die TAS die notwendigen Wiederherstellungsmaßnahmen am SUT durchführen (z. B. einen Neustart des SUT), sofern dies möglich und praktikabel ist.
- Evaluierung von Wartemechanismen, um sicherzustellen, dass der beste Mechanismus verwendet wird. Es gibt drei gängige Wartemechanismen:
  - Hartcodierte Wartezeiten (d. h. eine bestimmte Anzahl von Millisekunden warten), die angesichts der Unvorhersehbarkeit von Software-Antwortzeiten die Grundursache für viele Fehlerzustände in der Testautomatisierung sein können.
  - Dynamisches Warten durch Polling (z. B. Überprüfung, ob eine bestimmte Zustandsänderung oder Aktion stattgefunden hat) ist viel flexibler und effizienter:
    * Die TAS wartet nur die benötigte Zeit und es wird keine Testzeit verschwendet.

    * Wenn der Prozess länger dauert als erwartet, wartet das Polling, bis die Fortsetzungsbedingung erfüllt ist. Damit der Test nicht ewig in einem Fehlerzustand wartet, sollte ein Timeout-Mechanismus eingebaut werden.
  - Ein noch besserer Weg ist es, den Ereignismechanismus des SUT zu abonnieren. Dies ist viel zuverlässiger als die anderen beiden Optionen, allerdings muss die Testskriptsprache die Ereignisabonnierung unterstützen und das SUT muss der TAS diese Ereignisse anbieten. Außerdem ist ein Timeout-Mechanismus erforderlich, sonst kann der Test bei einem Fehlerzustand ewig warten.

**Testausführung**

Wenn die Ausführung einer automatisierten Regressionstestsuite aufgrund zu langer Ausführungszeit nicht beendet ist, kann es notwendig sein, gleichzeitig in verschiedenen Testumgebungen zu testen, wenn dies möglich ist. Wenn teure Systeme zum Testen verwendet werden, kann es eine Einschränkung sein, dass alle Tests auf einem einzigen Zielsystem durchgeführt werden müssen. Es kann erforderlich sein, die Testsuite für Regressionstests in mehrere Teile aufzuteilen, die jeweils in einem bestimmten Zeitraum (z. B. in einer einzigen Nacht) ausgeführt werden. Eine weitere Analyse der Überdeckung der Testautomatisierung kann Doppelungen aufdecken. Die Beseitigung von Duplikaten kann die Testausführung verkürzen und weitere Effizienzgewinne bringen. Im Falle von CI/CD ist es eine gute Praxis, Batch-Jobs parallel laufen zu lassen, um die Testausführungszeit zu optimieren. Außerdem empfiehlt es sich, automatisierte Batch-Jobs zu planen, um die verschiedenen Pipelines zu einem bestimmten Zeitpunkt auszuführen, z. B. jeden Morgen, um manuelle Interaktionen zu reduzieren und den Entwicklungsprozess zu beschleunigen.

**Verifizierung**

Bevor neue Verifizierungsfunktionen erstellt werden, sollte die Einführung einer Reihe von Standardverifizierungsfunktionen für die automatisierten Tests geprüft werden. Dadurch wird die mehrfache Implementierung von Verifizierungsfunktionen für unterschiedliche Tests vermieden. Wenn die Verifizierungsfunktionen nicht identisch, aber ähnlich sind, hilft die Verwendung von Parametern, damit eine Funktion für mehrere Objekttypen genutzt werden kann.

**TAA**

Es kann notwendig sein, die TAA zu ändern, um Verbesserungen der Testbarkeit des SUT zu unterstützen. Diese Änderungen können in der Architektur des SUT und/oder in der TAA der TAS vorgenommen werden. Dies kann eine wesentliche Verbesserung der Testautomatisierung erzielen, kann aber auch erhebliche Änderungen und Investitionen in das SUT / die TAS erfordern. Wenn z. B. das SUT geändert wird, um APIs für das Testen bereitzustellen, dann sollte auch die TAS entsprechend umgestaltet werden. Das Hinzufügen solcher Funktionalitäten zu einem späteren Zeitpunkt im SDLC kann sehr teuer werden; es ist besser, dies zu Beginn der Testautomatisierung und in den frühen SDLC-Phasen des SUT zu bedenken.

**TAF**

Neue Versionen der Kernbibliotheken eines TAF werden häufiger bereitgestellt. Manchmal handelt es sich dabei um größere Aktualisierungen, so dass die neueste Version nicht sofort referenziert werden kann, da sonst die Tests vieler Teams beeinträchtigt würden. Daher ist es besser, zunächst ein Pilotprojekt und eine Auswirkungsanalyse durchzuführen. Danach kann ein Plan für die Einführung erstellt werden. Entweder übernehmen alle Teams die neue Version der Kernbibliotheken gleichzeitig, indem sie die Abhängigkeit in der Build-Datei der Kernbibliotheksschicht aktualisieren, oder jedes Team entscheidet individuell, wann es die Aktualisierung in seiner Geschäftslogikschicht vornimmt. Sobald alle Teams bereit sind, die neue

Version der Kernbibliotheken zu verwenden, können die Abhängigkeiten in der Kernbibliotheksschicht aktualisiert werden (siehe *Kapitel 3.1.3*).

**Setup und Teardown**

Aktionen und Konfigurationen, die vor oder nach jedem Testskript oder jeder Testsuite wiederholt werden, sollten in die Setup- oder Teardown-Methoden verschoben werden. Auf diese Weise können alle Änderungen, die sich auf den Code auswirken, an einer Stelle aktualisiert werden, was zu einem geringeren Wartungsaufwand führt. Beispielsweise können Webservice-Aufrufe verwendet werden, um Vorbedingungen oder Nachbedingungen für UI-Tests zu erfüllen (z. B. Benutzerregistrierung, Benutzerbereinigung und Profileinrichtung).

**Dokumentation**

Dies umfasst alle Formen der Dokumentation, von der Dokumentation zur Testautomatisierung (z. B. was der Testautomatisierungscode macht und wie er verwendet werden sollte) über die Benutzerdokumentation für die TAS bis hin zu den von der TAS erstellten Testberichten und Testprotokollen.

**TAS-Funktionalitäten**

TAS-Funktionalitäten, wie detaillierte Testberichterstattung, Testprotokolle und Integration in andere Systeme, werden hinzugefügt. Dabei sollte es sich nur um neue Funktionalitäten handeln, die auch genutzt werden. Das Hinzufügen ungenutzter Funktionalitäten erhöht die Komplexität und verringert die Zuverlässigkeit und Wartbarkeit.

**TAF-Updates und -Upgrades**

Durch Aktualisierungen oder Upgrades auf neue TAF-Versionen können neue Funktionen verfügbar werden, die von den Testfällen genutzt werden können, oder es können Fehlerwirkungen korrigiert werden. Das Risiko besteht darin, dass eine Aktualisierung des TAF durch ein Upgrade der bestehenden Testwerkzeuge oder die Einführung neuer Werkzeuge negative Auswirkungen auf die bestehenden Testfälle haben kann. Die neueste Version des Testwerkzeugs muss getestet werden, indem Beispieltests ausgeführt werden, bevor sie eingeführt wird. Die Beispieltests sollten repräsentativ für die automatisierten Tests verschiedener SUTs, verschiedener Testarten und ggf. verschiedener Testumgebungen sein.

### 8.1.3 Automatisierte Testmittel zwecks Anpassung an Updates des System unter Test restrukturieren

Änderungen an einem bestehenden SUT erfordern Aktualisierungen der TAS einschließlich des TAF und der Komponentenbibliotheken. Jede noch so triviale Änderung kann weitreichende negative Auswirkungen auf die Zuverlässigkeit und Leistungsfähigkeit der TAS haben.

**Identifizieren von Änderungen an den Komponenten der Testumgebung**

Es ist zu beurteilen, welche Änderungen und Verbesserungen vorgenommen werden müssen und ob dadurch Änderungen an den Testmitteln, den angepassten Funktionsbibliotheken oder dem Betriebssystem erforderlich sind. Jede dieser Änderungen hat Auswirkungen auf die Leistung der TAS. Das übergeordnete Ziel ist es, sicherzustellen, dass automatisierte Tests weiterhin effizient ablaufen. Änderungen sollten schrittweise vorgenommen werden, wobei von einem "Minimum Viable Product"-Mindset ausgegangen werden sollte, damit die Auswirkungen auf die TAS durch eine möglichst geringe Anzahl an auszuführenden Tests bewertet werden kann. Sobald klar ist, dass keine Nebenwirkungen aufgetreten sind, können die Änderungen vollständig implementiert werden. Ein vollständiger Regressionstestlauf ist der letzte

Schritt, um zu überprüfen, ob sich die Änderung nicht negativ auf die automatisierten Testskripte ausgewirkt hat. Während der Ausführung dieser Regressionstests können Fehlerwirkungen festgestellt werden. Die Identifizierung der Grundursache dieser Fehlerwirkungen (z. B. durch Testberichterstattung, Testprotokolle und Testdatenanalyse) stellt sicher, dass diese nicht auf die Verbesserung der Testautomatisierung zurückzuführen sind.

**Steigerung der Effizienz und Effektivität der zentralen TAS-Funktionsbibliotheken**

Mit zunehmender Reife einer TAS werden neue Möglichkeiten entdeckt, Aufgaben effizienter zu erledigen. Diese neuen Verfahren (z. B. die Optimierung von Funktionscode und die Verwendung neuerer Betriebssystembibliotheken) müssen in die Kernfunktionsbibliotheken aufgenommen werden, die vom aktuellen Projekt und von zukünftigen Projekten verwendet werden.

**Konsolidierung mehrerer Funktionen, die auf dasselbe Steuerelement wirken**

Ein großer Teil eines automatisierten Testlaufs besteht aus der Abfrage von Steuerelementen der GUI. Diese Abfrage dient dazu, Informationen von einem Steuerelement zu erhalten (z. B. sichtbar/nicht sichtbar, aktiviert/nicht aktiviert, Größe und Abmessungen sowie Daten). Mit diesen Informationen kann ein automatisierter Test ein Element aus einer Dropdown-Liste auswählen, Daten in ein Feld eingeben und einen Wert aus einem Feld auslesen. Es gibt mehrere Funktionen, die auf Steuerelemente zugreifen, um diese Informationen zu ermitteln. Einige Funktionen sind sehr spezialisiert, während andere eher allgemeiner Natur sind. So kann es beispielsweise eine spezielle Funktion geben, die nur mit Dropdown-Listen interagiert. Es kann aber auch eine Funktion geben, die weitere Funktionen aufruft, indem sie eine Funktion als einen ihrer Parameter angibt. Daher können mehrere Funktionen in weniger Funktionen zusammengefasst und vom TAE verwendet werden, um die gleichen Ergebnisse zu erzielen und die Wartung zu minimieren.

**Refaktorisierung der TAA, um mit den Änderungen im SUT Schritt zu halten**

Während des Lebenszyklus einer TAS müssen Änderungen vorgenommen werden, um Änderungen im SUT zu berücksichtigen. Wenn sich das SUT weiterentwickelt und reift, muss sich auch die zugrunde liegende TAA weiterentwickeln, um sicherzustellen, dass die Unterstützung des SUT weiterhin vorhanden ist. Bei der Erweiterung von Funktionalitäten muss darauf geachtet werden, dass diese nicht ad hoc implementiert werden, sondern analysiert und in der TAA geändert werden. So wird sichergestellt, dass, wenn neue SUT-Funktionalität zusätzliche Testskripte erfordert, kompatible Komponenten vorhanden sind, um diese neuen automatisierten Tests zu unterstützen.

**Namenskonventionen und Standardisierung**

Bei der Einführung von Änderungen müssen Namenskonventionen für den neuen Testautomatisierungscode und neue Funktionsbibliotheken mit den zuvor definierten Standards übereinstimmen (siehe *Kapitel 4.3.1*).

**Evaluierung bestehender Testskripte zwecks Überarbeitung/Löschung**

Zum Änderungs- und Verbesserungsprozess gehört auch eine Bewertung der vorhandenen Testskripte, ihrer Verwendung und ihres fortlaufenden Mehrwerts. Wenn beispielsweise bestimmte Tests komplex und zeitaufwendig sind, kann es sinnvoller und effizienter sein, sie in kleinere Tests zu zerlegen. Durch die gezielte Löschung von Tests, die nur selten oder gar nicht ausgeführt werden, wird die Komplexität der TAS reduziert und es wird klarer, was gewartet werden muss.

### 8.1.4 Möglichkeiten für den Einsatz von Testautomatisierungswerkzeugen zusammenfassen

Abgesehen vom eigentlichen Testen kann die Testautomatisierung bei weiteren Testaktivitäten helfen, wie z. B. den folgenden:

**Einrichtung und Steuerung der Umgebung**

Bestimmte Testskripte (z. B. die Erstellung von Testdaten) können in Setup-Methoden wirksam eingesetzt werden, um verschiedene Testdaten in einer neuen Testumgebung zu erstellen. In einer Situation, in der Benutzer mit mehreren Profilen auf der Grundlage verschiedener Dateneingaben erstellt werden müssen, kann ein Team ein automatisiertes Testskript verwenden, um einen Webservice-Endpunkt aufzurufen, der diese Benutzer registriert. Testskripte können die Einrichtung der Testinfrastruktur und die Bereinigung im Anschluss an den Prozess steuern. So lässt sich Zeit sparen und sicherstellen, dass in jeder neuen Testumgebung die richtigen Benutzer vorhanden sind. Beispielsweise könnten auch Testprotokolle und andere Testmittel automatisch aus der Testumgebung entfernt werden, wodurch die Organisation und Nutzung der Testumgebung effizienter wird.

**Datenalterung**

Mit Hilfe der Testautomatisierung lassen sich Testdaten in der Testumgebung manipulieren. In Datenbanken können z. B. die Datumsfelder überprüft und kontrolliert werden, um sie in Bezug auf das Jahr auf dem neuesten Stand zu halten.

**Screenshot- und Videoerstellung**

Die meisten modernen UI-Testautomatisierungswerkzeuge verfügen über eine eingebaute Funktion zur Erstellung und Speicherung von Screenshots oder Videos, wenn bestimmte Bedingungen eintreten. Mit diesen Testwerkzeugen können die Teams die Fachbereiche dabei unterstützen, Screenshots und Videos der tatsächlichen Nutzung für die Dokumentation von Softwarereleases oder für Marketingzwecke zu erstellen.

# 9 Anhänge A – Lernziele/Kognitiver Wissenstand

Die nachfolgenden Lernziele sind für diesen Lehrplan definiert. Jedes Thema des Lehrplans wird anhand des jeweiligen Lernziels untersucht.

Die Lernziele beginnen mit einem der nachfolgenden Aktionsverben, das dem jeweiligen kognitiven Wissensstand entspricht (siehe unten).

**Stufe 2: Verstehen (K2)**

Der Kandidat kann die Gründe oder Erklärungen für Aussagen zum Thema auswählen und kann das Testkonzept zusammenfassen, vergleichen, einordnen und Beispiele nennen.

**Aktionsverben**: Klassifizieren, vergleichen, differenzieren, unterscheiden, erklären, Beispiele nennen, schließen (auf), zusammenfassen

| Beispiele | Anmerkungen |
|---|---|
| Klassifizieren Sie Testwerkzeuge nach ihrem Zweck und den Testaktivitäten, die sie unterstützen. | |
| Vergleichen Sie die verschiedenen Teststufen. | Kann verwendet werden, um nach Ähnlichkeiten, Unterschieden oder beidem zu suchen. |
| Differenzieren Sie zwischen Testen und Debugging. | Sucht nach Unterschieden zwischen Konzepten. |
| Unterscheiden Sie zwischen Projektrisiken und Produktrisiken. | Ermöglicht die separate Klassifizierung von zwei (oder mehr) Konzepten. |
| Erklären Sie, wie sich der Kontext auf den Testprozess auswirkt. | |
| Nennen Sie Beispiele dafür, warum Testen notwendig ist. | |
| Schließen Sie aus einem gegebenen Profil von Fehlerzuständen auf die Grundursache von Fehlern. | |
| Fassen Sie die Aktivitäten des Review-Prozesses für Arbeitsergebnisse zusammen. | |

**Stufe 3: Anwenden (K3)**

Der Kandidat kann ein Verfahren ausführen, wenn er mit einer vertrauten Aufgabe konfrontiert wird, oder das richtige Verfahren auswählen und es auf einen gegebenen Kontext anwenden.

**Aktionsverben**: Anwenden, umsetzen, erstellen, ableiten, nutzen

| Beispiele | Anmerkungen |
|---|---|
| Wenden Sie die Grenzwertanalyse an, um Testfälle aus gegebenen Anforderungen abzuleiten. | Sollte sich auf ein Verfahren/eine Technik/einen Prozess etc. beziehen. |
| Setzen Sie Methoden zur Erfassung von Metriken um, um technische und Managementanforderungen zu unterstützen. | |
| Erstellen Sie Installierbarkeitstests für mobile Anwendungen. | |
| Nutzen Sie die Verfolgbarkeit, um den Testfortschritt auf Vollständigkeit und Konsistenz mit den Testzielen, der Teststrategie und dem Testkonzept zu überwachen. | Könnte in einem LO verwendet werden, in dem der Kandidat in der Lage sein soll, ein Verfahren oder eine Technik zu nutzen. Ähnlich wie 'anwenden'. |

**Stufe 4: Analysieren (K4)**

Der Kandidat kann Informationen, die sich auf ein Verfahren oder eine Technik beziehen, zum besseren Verständnis in ihre Bestandteile zerlegen und zwischen Fakten und Schlussfolgerungen unterscheiden. Eine typische Anwendung ist die Analyse eines Dokuments, einer Software oder einer Projektsituation und das Vorschlagen geeigneter Maßnahmen zur Lösung eines Problems oder einer Aufgabe.

**Aktionsverben:** Analysieren, dekonstruieren, gliedern, priorisieren, auswählen

| Beispiele | Anmerkungen |
|---|---|
| Analysieren Sie eine gegebene Projektsituation, um festzustellen, welche Black-Box-Testverfahren oder erfahrungsbasierten Testverfahren zur Erreichung bestimmter Ziele eingesetzt werden sollten. | Nur in Verbindung mit einem messbaren Ziel der Analyse prüfbar. Sollte die Form 'Analysieren Sie xxxx bis xxxx' (oder ähnlich) haben. |
| Priorisieren Sie Testfälle in einer bestimmten Testsuite für die Ausführung auf der Grundlage der damit verbundenen Produktrisiken. | |
| Wählen Sie geeignete Teststufen und Testarten zur Verifizierung einer gegebenen Menge von Anforderungen aus. | Ist notwendig, wenn die Auswahl eine Analyse erfordert. |

**Referenz**

(Für die kognitiven Ebenen von Lernzielen)

Anderson, L. W. und Krathwohl, D. R. (Hrsg.) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon.

# 10 Anhänge B – Matrix zur Verfolgbarkeit des geschäftlichen Nutzen (Business Outcomes) mit Lernzielen (Learning Objectives)

Dieser Abschnitt listet die Anzahl der Lernziele des vorliegenden Lehrplans auf, die mit dem geschäftlichen Nutzen in Verbindung stehen, sowie die Verfolgbarkeit zwischen dem geschäftlichen Nutzen des Lehrplans und den Lernzielen des Advanced Level Test Automation Engineering.

| Geschäftlicher Nutzen: Advanced Level Test Automation Engineering | | TAE-BO1 | TAE-BO2 | TAE-BO3 | TAE-BO4 | TAE-BO5 | TAE-BO6 | TAE-BO7 | TAE-BO8 | TAE-BO9 | TAE-BO10 | TAE-BO11 | TAE-BO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAE-BO1 | Zweck der Testautomatisierung beschreiben | 1 | | | | | | | | | | | |
| TAE-BO2 | Testautomatisierung im Softwareentwicklungslebenszyklus verstehen | | 2 | | | | | | | | | | |
| TAE-BO3 | Konfiguration einer Infrastruktur zur Ermöglichung der Testautomatisierung verstehen | | | 2 | | | | | | | | | |
| TAE-BO4 | Den Evaluierungsprozess für die Auswahl der richtigen Werkzeuge und Strategien kennen | | | | 2 | | | | | | | | |
| TAE-BO5 | Designkonzepte für den Aufbau modularer und skalierbarer Testautomatisierungslösungen verstehen | | | | | 5 | | | | | | | |
| TAE-BO6 | Einen Ansatz, einschließlich eines Piloten, zur Planung der Testautomatisierung innerhalb des Softwareentwicklungslebenszyklus auswählen | | | | | | 1 | | | | | | |
| TAE-BO7 | (Neue oder modifizierte) Testautomatisierungslösungen, die den technischen Anforderungen entsprechen, entwerfen und entwickeln | | | | | | | 1 | | | | | |
| TAE-BO8 | Umfang und Ansatz der Testautomatisierung und Wartung von Testmitteln berücksichtigen | | | | | | | | 1 | | | | |

| Geschäftlicher Nutzen: Advanced Level Test Automation Engineering | | TAE-BO1 | TAE-BO2 | TAE-BO3 | TAE-BO4 | TAE-BO5 | TAE-BO6 | TAE-BO7 | TAE-BO8 | TAE-BO9 | TAE-BO10 | TAE-BO11 | TAE-BO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAE-BO9 | Die Integration automatisierter Tests in CI/CD-Pipelines verstehen | | | | | | | | | 3 | | | |
| TAE-BO10 | Verstehen, wie man Testautomatisierungsdaten sammelt, analysiert und darüber berichtet, um Stakeholder zu informieren | | | | | | | | | | 3 | | |
| TAE-BO11 | Testautomatisierungs-Infrastruktur verifizieren | | | | | | | | | | | 4 | |
| TAE-BO12 | Möglichkeiten zur kontinuierlichen Verbesserung der Testautomatisierung definieren | | | | | | | | | | | | 4 |

| Geschäftlicher Nutzen: Advanced Level Test Automation Engineering | | TAE-BO1 | TAE-BO2 | TAE-BO3 | TAE-BO4 | TAE-BO5 | TAE-BO6 | TAE-BO7 | TAE-BO8 | TAE-BO9 | TAE-BO10 | TAE-BO11 | TAE-BO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LO-Nummer | Lernziel (K-level) | | | | | | | | | | | | |
| 1 | Einführung und Ziele der Testautomatisierung - 45 Minuten (K2) | | | | | | | | | | | | |
| 1.1 | Zweck der Testautomatisierung | | | | | | | | | | | | |
| TAE-1.1.1 | ... die Vor- und Nachteile der Testautomatisierung erklären (K2) | X | | | | | | | | | | | |
| 1.2 | Testautomatisierung im Softwareentwicklungslebenszyklus | | | | | | | | | | | | |
| TAE-1.2.1 | ... den Einsatz der Testautomatisierung in verschiedenen Softwareentwicklungslebenszyklus-Modellen erklären (K2) | | X | | | | | | | | | | |
| TAE-1.2.2 | ... geeignete Testautomatisierungswerkzeuge für ein bestimmtes System unter Test auswählen (K2) | | X | | | | | | | | | | |
| 2 | Vorbereitung auf die Testautomatisierung - 180 Minuten (K4) | | | | | | | | | | | | |
| 2.1 | Die Konfiguration einer Infrastruktur zur Ermöglichung einer Testautomatisierung verstehen | | | | | | | | | | | | |
| TAE-2.1.1 | ... Anforderungen an die Konfiguration einer Infrastruktur beschreiben, um Testautomatisierung zu ermöglichen (K2) | | | X | | | | | | | | | |
| TAE-2.1.2 | ... den Einsatz einer Testautomatisierung in verschiedenen Umgebungen erläutern (K2) | | | X | | | | | | | | | |
| 2.2 | Den Evaluierungsprozess für die Auswahl der richtigen Werkzeuge und Strategien kennen | | | | | | | | | | | | |
| TAE-2.2.1 | ... ein System unter Test analysieren, um die geeignete Testautomatisierungslösung zu bestimmen (K4) | | | | X | | | | | | | | |

| Geschäftlicher Nutzen: Advanced Level Test Automation Engineering | | TAE-BO1 | TAE-BO2 | TAE-BO3 | TAE-BO4 | TAE-BO5 | TAE-BO6 | TAE-BO7 | TAE-BO8 | TAE-BO9 | TAE-BO10 | TAE-BO11 | TAE-BO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAE-2.2.2 | ... technische Befunde einer Werkzeugevaluation veranschaulichen (K4) | | | | X | | | | | | | | |
| 3 | Testautomatisierungsarchitektur - 210 Minuten (K3) | | | | | | | | | | | | |
| 3.1 | Entwurfskonzepte in der Testautomatisierung anwenden | | | | | | | | | | | | |
| TAE-3.1.1 | ... die wichtigsten Funktionen einer Testautomatisierungsarchitektur erläutern (K2) | | | | | X | | | | | | | |
| TAE-3.1.2 | ... erläutern, wie man eine Testautomatisierungslösung entwirft (K2) | | | | | X | | | | | | | |
| TAE-3.1.3 | ... die Schichten von Testautomatisierungsframeworks anwenden (K3) | | | | | X | | | | | | | |
| TAE-3.1.4 | ... die verschiedenen Ansätze zur Automatisierung von Testfällen anwenden (K3) | | | | | X | | | | | | | |
| TAE-3.1.5 | ... die Entwurfsprinzipien und Entwurfsmuster in der Testautomatisierung anwenden (K3) | | | | | X | | | | | | | |
| 4 | Implementierung der Testautomatisierung - 150 Minuten (K4) | | | | | | | | | | | | |
| 4.1 | Entwicklung der Testautomatisierung | | | | | | | | | | | | |
| TAE-4.1.1 | ... Richtlinien anwenden, die eine effektive Pilotierung und Implementierung der Testautomatisierung unterstützen (K3) | | | | | | X | | | | | | |
| 4.2 | Risiken im Zusammenhang mit der Entwicklung der Testautomatisierung | | | | | | | | | | | | |
| TAE-4.2.1 | ... Risiken bei der Einführung analysieren und Strategien zur Risikominderung für die Testautomatisierung planen (K4) | | | | | | | X | | | | | |
| 4.3 | Wartbarkeit einer Testautomatisierungslösung | | | | | | | | | | | | |

| Geschäftlicher Nutzen: Advanced Level Test Automation Engineering | | TAE-BO1 | TAE-BO2 | TAE-BO3 | TAE-BO4 | TAE-BO5 | TAE-BO6 | TAE-BO7 | TAE-BO8 | TAE-BO9 | TAE-BO10 | TAE-BO11 | TAE-BO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAE-4.3.1 | ... erläutern, welche Faktoren die Wartbarkeit einer Testautomatisierungslösung unterstützen und beeinflussen (K2) | | | | | | | | X | | | | |
| 5 | Implementierungs- und Deployment-Strategien für die Testautomatisierung - 90 Minuten (K3) | | | | | | | | | | | | |
| 5.1 | Integration automatisierter Tests in CI/CD-Pipelines verstehen | | | | | | | | | | | | |
| TAE-5.1.1 | ... Testautomatisierung auf verschiedenen Teststufen innerhalb von Pipelines anwenden (K3) | | | | | | | | | X | | | |
| TAE-5.1.2 | ... Konfigurationsmanagement für Testmittel erklären (K2) | | | | | | | | | X | | | |
| TAE-5.1.3 | ... Abhängigkeiten bei der Testautomatisierung für eine API-Infrastruktur erläutern (K2) | | | | | | | | | X | | | |
| 6 | Testautomatisierung: Berichtswesen und Metriken - 150 Minuten (K4) | | | | | | | | | | | | |
| 6.1 | Erfassung, Analyse und Auswertung von Daten zur Testautomatisierung | | | | | | | | | | | | |
| TAE-6.1.1 | ... Methoden zur Erfassung von Daten aus der Testautomatisierungslösung und dem System unter Test anwenden (K3) | | | | | | | | | | X | | |
| TAE-6.1.2 | ... Daten aus der Testautomatisierungslösung und dem System unter Test analysieren, um die Ergebnisse besser zu verstehen (K4) | | | | | | | | | | X | | |
| TAE-6.1.3 | ... erklären, wie ein Testfortschrittsbericht erstellt und veröffentlicht wird (K2) | | | | | | | | | | X | | |
| 7 | Verifizierung der Testautomatisierungslösung - 135 Minuten (K3) | | | | | | | | | | | | |
| 7.1 | Verifizierung der Testautomatisierungs-Infrastruktur | | | | | | | | | | | | |

| Geschäftlicher Nutzen: Advanced Level Test Automation Engineering | | TAE-BO1 | TAE-BO2 | TAE-BO3 | TAE-BO4 | TAE-BO5 | TAE-BO6 | TAE-BO7 | TAE-BO8 | TAE-BO9 | TAE-BO10 | TAE-BO11 | TAE-BO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAE-7.1.1 | ... die Verifizierung der Testautomatisierungsumgebung planen, einschließlich der Einrichtung der Testwerkzeuge (K3) | | | | | | | | | | | X | |
| TAE-7.1.2 | ... das korrekte Verhalten für ein gegebenes automatisiertes Testskript und/oder eine Testsuite erläutern (K2) | | | | | | | | | | | X | |
| TAE-7.1.3 | ... identifizieren, wo die Testautomatisierung unerwartete Ergebnisse liefert (K2) | | | | | | | | | | | X | |
| TAE-7.1.4 | ... erläutern, wie statische Analyse helfen kann, die Qualität des Codes für die Testautomatisierung zu verbessern (K2) | | | | | | | | | | | X | |
| 8 | Kontinuierliche Verbesserung - 210 Minuten (K4) | | | | | | | | | | | | |
| 8.1 | Kontinuierliche Verbesserungsmöglichkeiten bei der Testautomatisierung | | | | | | | | | | | | |
| TAE-8.1.1 | ... Möglichkeiten zur Verbesserung von Testfällen durch Datensammlung und Datenanalyse entdecken (K3) | | | | | | | | | | | | X |
| TAE-8.1.2 | ... technische Aspekte einer eingesetzten Testautomatisierungslösung analysieren und Verbesserungen empfehlen (K4) | | | | | | | | | | | | X |
| TAE-8.1.3 | ... automatisierte Testmittel zwecks Anpassung an Updates des System unter Test restrukturieren (K3) | | | | | | | | | | | | X |
| TAE-8.1.4 | ... Möglichkeiten für den Einsatz von Testautomatisierungswerkzeugen zusammenfassen (K2) | | | | | | | | | | | | X |

# 11 Anhänge C – Release Notes

Der Syllabus ISTQB® Test Automation Engineering 2024 ist eine umfassende Aktualisierung und Neufassung der Version 2016. Aus diesem Grund gibt es keine detaillierten Versionshinweise pro Kapitel und Abschnitt. Der Inhalt wurde für die Rolle des Testautomatisierungsentwicklers erheblich erweitert, während strategische Inhalte in den neuen Syllabus ISTQB® Testautomatisierungsstrategie 2024 verschoben wurden.

# 12 Anhänge D – Bereichsspezifische Begriffe

| Term-Name | Definition |
|---|---|
| DevSecOps | Eine Methodik, die Entwicklung, IT-Sicherheit und Betrieb kombiniert und einen hohen Automatisierungsgrad aufweist. |
| Flow Model Pattern | Eine High-Level-Ansicht des Arbeitsbereichs, seiner Komponenten und der Verbindungen zwischen ihnen. |
| User Journey | Eine Reihe von Schritten, die aus der Perspektive der Erfahrung einer Person zeigen, wie ein Benutzer mit einem System unter Test interagiert. |
| Page Object Pattern | Ein Entwurfsmuster der Testautomatisierung zur Verbesserung der Wartung von Tests und zur Verringerung von dupliziertem Code. |
| Versionskontrolle | Ein Verfahren zum Einchecken und Speichern bestimmter Versionen von Quellcode. |

# 13 Referenzen

## Normen

*ISO/IEC 25010 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE): Product quality mode*, 2023. Standard. International Organization for Standardization.

*ISO/IEC/IEEE 29119-5 Software and systems engineering – Software testing: Part 5: Keyword-Driven Testing*, 2016. Standard. International Organization for Standardization.

## ISTQB® Dokumente

ISTQB-CT-AI, ISTQB®, 2021. *Certified Tester AI Testing Syllabus*. v1.0.

ISTQB-CT-MBT, ISTQB®, 2015. *Certified Tester Model-Based Tester Syllabus*. v1.1.

*ISTQB-CT-PT, ISTQB®: Certified Tester Performance Testing Syllabus*, 2023. v1.0.

ISTQB-CT-SEC, ISTQB®, 2016. *Certified Tester Security Tester Syllabus*. v1.0.

ISTQB-CT-TAS, ISTQB®, 2024. *Certified Tester Test Automation Strategy Syllabus*. v1.0.

ISTQB-CTFL, ISTQB®, 2023. *Certified Tester Foundation Level Syllabus*. v4.0.

# 14 Weitere Literatur

BAKER, Paul; DAI, Zhen Ru; GRABOWSKI, Jens; HAUGEN, Oystein; SCHIEFERDECKER, Ina; WILLIAMS, Clay E., 2007. *Model-Driven Testing - Using the UML Testing Profile*. Springer-Verlag Berlin Heidelberg. ISBN 9783540725626.

BINDER, Robert V.; MILLER, Suzanne, 2017. Five Keys to Effective Agile Test Automation for Government Programs. *Software Engineering Institute, Carnegie Mellon University*. Auch verfügbar unter: https://resources.sei.cmu.edu/asset_files/Webinar/2017_018_101_503516.pdf.

CIO, DoD, 2023. Modern Software Practices - DevSecOps Fundamentals Guidebook: Activities and Tools. Jg. v2.2. Auch verfügbar unter: https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebookTables.pdf.

DUSTIN, Elfriede; GARRETT, Thom; GAUF, Bernie, 2009. *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality*. Addison-Wesley Professional. ISBN 0321580516.

DUSTIN, Elfriede; RASHKA, Jeff; PAUL, John, 1999. *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley Professional. ISBN 9780201432879.

FÖLDHÁZI, Péter, 2022. Tri-Layer Testing Architecture. Auch verfügbar unter: https://www.pnsqc.org/docs/PROP53522057-FoldhaziDraftFinal.pdf.

GRAHAM, Dorothy; FEWSTER, Mark, 1999. *Software Test Automation: Effective Use of Test Execution Tools*. Addison-Wesley Professional. ISBN 9780201331400.

GRAHAM, Dorothy; FEWSTER, Mark, 2012. *Experiences of Test Automation: Case Studies of Software Test Automation*. Addison-Wesley Professional. ISBN 9780321754066.

*ISO/IEC/IEEE 29119-1 Software and systems engineering – Software testing: Part 1: General Concepts*, 2022. Standard. International Organization for Standardization.

ISTQB-CTAL-TAE, ISTQB®, 2024. *Certified Tester Test Automation Engineering Syllabus*. v2.0.

ISTQB-CTAL-TM, ISTQB®, 2024. *Certified Tester Advanced Level Test Manager Syllabus*. v2.0.

MARTIN, Robert C., 2008. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson. ISBN 9780132350884.

MCCAFFREY, James D., 2006. *.NET Test Automation Recipes: A Problem-Solution Approach*. APRESS. ISBN 9781590596630.

MOSLEY, Daniel J.; POSEY, Bruce A., 2002. *Just Enough Software Test Automation*. Prentice Hall. ISBN 9780130084682.

PESTAK, Thomas; ROWELL, William, 2018. Automated Software Testing Practices and Pitfalls. Auch verfügbar unter: https://www.afit.edu/stat/statcoe_files/Automated%20Software%20Testing%20Practices%20and%20Pitfalls%20Rev%201.pdf.

POLLNER, Andrew; SIMPSON, Jim; WISNOWSKI, Jim, 2018. Automated Software Testing Implementation Guide for Managers and Practitioners. Auch verfügbar unter: https://www.afit.edu/stat/statcoe_files/Automated_Software_Testing_Implementation_Guide.pdf.

PROJECT, The Selenium, 2024. Page object models. Auch verfügbar unter: https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models.

SAMBAMURTHY, Manikandan, 2023. *Test Automation Engineering Handbook*. Packt Publishing. ISBN 9781804615492.

WILLCOCK, Colin; DEISS, Thomas; TOBIES, Stephan; KEIL, Stefan; ENGLER, Federico; SCHULZ, Stephan, 2011. *An Introduction to TTCN-3*. Wiley. ISBN 9780470663066.

# 15 Abbildungsverzeichnis

# 16 Index