Certified Tester
Advanced Level Syllabus
Testautomatisierungsentwickler

Version 2019

German Testing Board e.V.

Übersetzung des englischsprachigen Lehrplans des International Software Testing Qualifications Board (ISTQB®), Originaltitel: Certified Tester, Advanced Level Syllabus, Test Automation Engineer Syllabus des ISTQB, Fassung 2016.

Änderungsübersicht

Version	Datum	Anmerkungen
2019 D	12.02.2019	Deutschsprachige Fassung des ISTQB-Release 2016
2019 E	25.02.2019	Freigabereview
2019 F	07.05.2019	Formatierung und Abstimmung der Glossarbegriffe
2019 G	03.09.2019	Einarbeitung Review-Anmerkungen
2019 H	16.12.2019	Kleine Korrekturen

Inhaltsverzeichnis

Hintergrund ISTQB CTAL TAE Syllabus

Der ISTQB CTAL TAE Syllabus wurde von einem Kernteam der Arbeitsuntergruppe „Advanced Level Syllabus – Test Automation“ des International Software Testing Qualifications Board erstellt. Das Kernteam bedankt sich beim Reviewteam und den nationalen Boards für die konstruktiven Vorschläge und Beiträge. Bei Fertigstellung des Advanced Level-Lehrplans hatte die Arbeitsgruppe „Advanced Level – Test Automation“ die folgenden Mitglieder: Bryan Bakker, Graham Bath (Leiter der Advanced Level-Arbeitsgruppe), Armin Beer, Inga Birthe, Armin Born, Alessandro Collino, Massimo Di Carlo, Mark Fewster, Mieke Gevers, Jani Haukinen, Skule Johansen, Eli Margolin, Judy McKay (stellvertr. Leiterin der Advanced Level-Arbeitsgruppe), Kateryna Nesmyelova, Mahantesh (Monty) Pattan, Andrew Pollner (Leiter der Gruppe „Advanced Level Test Automation“), Raluca Popescu, Ioana Prundaru, Riccardo Rosci, Ina Schieferdecker, Gil Shekel, Chris Van Bael. Die Autoren des Kernteams für diesen Lehrplan waren: Andrew Pollner (Leiter), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker. Folgende Personen haben an Review, Kommentierung und Abstimmung über diesen Lehrplan mitgearbeitet (in alphabetischer Reihenfolge): Armin Beer, Tibor Csöndes, Massimo Di Carlo, Chen Geng, Cheryl George, Kari Kakkonen, Jen Leger, Singh Manku, Ana Paiva, Raluca Popescu, Meile Posthuma, Darshan Preet, Ioana Prundaru, Stephanie Ulrich, Erik van Veenendaal, Rahul Verma. Der ISTQB CTAL TAE Syllabus wurde am 21. Oktober 2016 von der Hauptversammlung des ISTQB® offiziell freigegeben.

Die hier vorliegende Übersetzung in die deutsche Sprache wurde von Ina Schieferdecker, Marc-Florian Wendland, Holger Schlösser, Baris Güldali, Florian Spohn, Armin Metzger, Richard Seidl, Stephan Weißleder und Jan Giesen (Leiter der Arbeitsgruppe „Testautomatisierung“ des GTB) erarbeitet.

0. Einführung in diesen Lehrplan

0.1 Zweck dieses Dokuments

Dieser Lehrplan definiert die Advanced Level (Aufbaustufe) für das Spezialmodul „Test Automation – Engineering“ des Softwaretest-Ausbildungsprogramms des International Software Testing Qualifications Board (im Folgenden ISTQB® genannt). Das German Testing Board (im Folgenden GTB® genannt) hat diesen Lehrplan in die deutsche Sprache übersetzt. Das GTB® und ISTQB® und stellen den Lehrplan folgenden Adressaten zur Verfügung:

- Nationalen Boards zur Übersetzung in die jeweilige Landessprache und zur Akkreditierung von Ausbildungsanbietern. Die nationalen Boards können den Lehrplan an die eigenen sprachlichen Anforderungen anpassen sowie die Querverweise ändern und an die bei ihnen vorliegenden Veröffentlichungen angleichen.
- Prüfungsinstitutionen zur Erarbeitung von Prüfungsfragen in der jeweiligen Landessprache, die sich an den Lernzielen der jeweiligen Lehrpläne orientieren.
- Ausbildungsanbietern zur Erstellung ihrer Kursunterlagen und zur Bestimmung einer geeigneten Unterrichtsmethodik.
- Prüfungskandidaten zur Vorbereitung auf die Prüfung (im Rahmen eines Schulungs-kurses oder des freien Lernens).
- Allen Personen, die im Bereich Software- und Systementwicklung tätig sind und ihre fachliche Kompetenz beim Testen von Software verbessern möchten, sowie als Grundlage für Bücher und Fachartikel.

Das GTB® und ISTQB® können die Nutzung dieses Lehrplans auch anderen Personenkreisen oder Institutionen für andere Zwecke genehmigen, sofern diese vorab eine entsprechende schriftliche Genehmigung einholen.

0.2 Geltungsbereich dieses Dokuments

0.2.1 Innerhalb des Geltungsbereichs

In diesem Dokument werden die Aufgaben von Testautomatisierungsentwicklern (in Englisch Test Automation Engineer, kurz TAE) beim Konzipieren, Entwickeln und Warten von Testautomatisierungslösungen beschrieben. Im Mittelpunkt stehen die Konzepte, Methoden, Werkzeuge und Prozesse für die Automatisierung dynamischer funktionaler Tests sowie der Zusammenhang zwischen diesen Tests und dem Testmanagement, dem Konfigurationsmanagement, dem Fehlermanagement, Softwareentwicklungsprozessen und der Qualitätssicherung.

Die beschriebenen Methoden sind auf eine Vielzahl von Softwarelebenszyklus-Ansätzen (z. B. agil, sequenziell, inkrementell, iterativ), Arten von Softwaresystemen (z. B. eingebettet, verteilt, mobil) und Testarten (funktionale und nicht-funktionale Tests) anwendbar.

0.2.2 Außerhalb des Geltungsbereichs

Die folgenden Aspekte werden in diesem Lehrplan für die Testautomatisierungsentwickler nicht behandelt:

- Testmanagement, automatisierte Erstellung von Testspezifikationen und automatisierte Testgenerierung
- Aufgaben des Testautomatisierungsmanagers (TAM) bei der Planung, Überwachung und Anpassung der Entwicklung und Weiterentwicklung von Testautomatisierungslösungen
- Aspekte der Automatisierung nichtfunktionaler Tests (z. B. Performanz)
- Automatisierung der statischen Analyse (z. B. Schwachstellenanalyse) und statischer Testwerkzeuge
- Vermitteln von Methoden der Softwareentwicklung und Programmierung (z. B. welche Standards zu verwenden sind und welche Kompetenzen für die Realisierung einer Testautomatisierungslösung benötigt werden)

- Vermitteln von Softwaretechnologien (z. B., welche Techniken der Skripterstellung für die Implementierung einer Testautomatisierungslösung zu verwenden sind)
- Auswahl von Softwaretestprodukten und -diensten (z. B. welche Produkte und Dienste für eine Testautomatisierungslösung zu verwenden sind)

0.3 Testautomatisierungsentwickler mit Advanced Level-Zertifizierung

0.3.1 Erwartungen

Zielgruppe dieses Aufbaukurses sind Personen, die sich aufbauend auf ihren bei der „Foundation Level"-Zertifizierung erworbenen Kenntnissen und Fertigkeiten Fachwissen in einem oder mehreren Spezialgebieten aneignen wollen. Die im Advanced Level für Spezialisten angebotenen Module decken ein breites Spektrum von testbezogenen Themen ab.

Ein Testautomatisierungsentwickler verfügt über ein breites Wissen über das Testen im Allgemeinen sowie fundierte Kenntnisse auf dem Spezialgebiet der Testautomatisierung. Mit fundierten Kenntnissen sind Kenntnisse über die Theorie und Praxis der Testautomatisierung gemeint, die umfassend genug sind, um die Richtung vorzugeben, die ein Unternehmen und/oder Projekt im Hinblick auf das Entwerfen, Entwickeln und Warten von Testautomatisierungslösungen für funktionale Tests nimmt.

Die Geschäftsziele für diesen Aufbaukurs sind in dem entsprechenden Übersichtsdokument beschrieben [ISTQB-AL-Modules].

0.3.2 Zulassungs- und Verlängerungskriterien

Die allgemeinen Zulassungskriterien für das Advanced Level werden auf der ISTQB-Website [ISTQB-Web] im Bereich „Advanced Level" beschrieben.

Neben diesen allgemeinen Zulassungskriterien müssen die Kandidaten das ISTQB Foundation Level-Zertifikat [ISTQB-CTFL] erworben haben, damit sie an der Prüfung für die Advanced Level-Zertifizierung Testautomatisierungsentwickler teilnehmen dürfen.

0.3.3 Wissensstand

Die Lernziele für diesen Lehrplan werden zu Beginn jedes Kapitels ganz klar genannt. Jedes Thema im Lehrplan wird anhand der Lernziele für dieses Thema abgeprüft.

Die kognitiven Stufen der einzelnen Lernziele („K-Level") werden auf der ISTQB-Website [ISTQB-Web] beschrieben.

0.3.4 Prüfung

Die Prüfung für dieses Advanced Level-Zertifikat muss auf diesem Lehrplan und dem Lehrplan für das Foundation Level [ISTQB-FL] basieren. Um bestimmte Prüfungsfragen beantworten zu können, muss unter Umständen Material genutzt werden, das auf mehreren Abschnitten dieser Lehrpläne basiert.

Das Prüfungsformat wird auf der ISTQB-Website [ISTQB-Web] im Bereich „Advanced Level" beschrieben. Auf der Website finden sich zudem hilfreiche Informationen für Absolventen der Prüfung.

0.3.5 Akkreditierung

Ein nationales ISTQB-Board kann Schulungsanbieter akkreditieren, deren Kursmaterial sich an diesem Lehrplan orientiert.

Auf der ISTQB-Website [ISTQB-Web] im Bereich „Advanced Level" sind die besonderen Regeln aufgeführt, die für Schulungsanbieter im Hinblick auf die Akkreditierung von Kursen gelten.

0.4 Normative und informative Teile im Vergleich

Die normativen Teile des Lehrplans sind prüfungsrelevant. Dabei handelt es sich um:

- Lernziele
- Schlüsselwörter

Der übrige Lehrplan ist informativ und befasst sich eingehender mit den Lernzielen.

0.5 Detailtiefe

Die Detailtiefe in diesem Lehrplan ermöglicht ein international einheitliches Lehren und Prüfen. Zum Erreichen dieses Ziels setzt sich der Lehrplan aus folgenden Elementen zusammen:

- Lernziele für jeden Wissensbereich, die das Ergebnis des kognitiven Lernens und die Denkweise beschreiben, die erreicht werden sollen (normativ).
- Eine Liste mit zu vermittelnden Informationen, einschließlich einer Beschreibung der wichtigsten zu vermittelnden Konzepte, Quellen wie akzeptierter Literatur oder Standards und ggf. Verweise auf zusätzliche Quellen (informativ).

Der Inhalt des Lehrplans ist kein Abriss des gesamten Wissens im Bereich der Testautomatisierung; er spiegelt die Detailtiefe wider, die in einem akkreditierten „Advanced Level"-Ausbildungskurs abgedeckt wird.

0.6 Aufbau dieses Lehrplans

Der Lehrplan gliedert sich in acht Kapitel. In der Kapitelüberschrift ist jeweils die Zeit für das Kapitel angegeben. Zum Beispiel:

3. Die generische Testautomatisierungsarchitektur – 270 min

bedeutet, dass für Kapitel 3 eine Zeit von 270 Minuten für die Vermittlung des Lernstoffs im Kapitel veranschlagt wird. Am Anfang der einzelnen Kapitel sind die jeweiligen Lernziele des Kapitels angegeben.

0.7 Geschlechtsneutrale Formulierungen

Aus Gründen der einfacheren Lesbarkeit wird auf die geschlechtsneutrale Differenzierung, wie beispielsweise Benutzer/innen, verzichtet. Sämtliche Rollenbezeichnungen gelten im Sinne der Gleichbehandlung grundsätzlich für alle Geschlechter.

0.8 Begriffe, Definitionen und Akronyme

In der Softwareliteratur werden viele Begriffe uneinheitlich benannt und verwendet. Die Definitionen der in diesem Advanced Level-Lehrplan verwendeten Testbegriffe stehen im ISTQB/GTB-Standardglossar der Testbegriffe [GTB-Glossar], bzw. im multilingualen ISTQB Glossary of Testing Terms [ISTQB-Glossary] zur Verfügung.

Jedes der am Anfang der einzelnen Kapitel in diesem Advanced Level-Lehrplan aufgeführten Schlüsselwörter ist im ISTQB-Glossar [ISTQB-Glossary] bzw. dem ISTQB/GTB-Glossar [GTB-Glossar] definiert.

Die folgenden Akronyme werden diesem Dokument verwendet:

API	Abkürzung von Application Programming Interface.
CLI	Kommandozeilenschnittstelle (in Englisch: Command Line Interface)
EMTE	Äquivalenter manueller Testaufwand (in Englisch: Equivalent Manual Test Effort): der manuelle Testaufwand, der für ein bestimmtes Ergebnis erforderlich ist, das mit automatisierten Tests erreicht wurde

gTAA	Generische Testautomatisierungsarchitektur (als Blaupause für Testautomatisierungslösungen, in Englisch: Generic Test Automation Architecture)
GUI	Graphische Benutzungsoberfläche (in Englisch: Graphical User Interface)
SUT	System unter Test (siehe auch Testobjekt, in Englisch: System under Test)
TAA	Testautomatisierungsarchitektur (eine Instanziierung der gTAA zum Definieren der Architektur einer TAS, in Englisch: Test Automation Architecture)
TAE	Testautomatisierungsentwickler (Person, die für das Design, die Implementierung, die Wartung und diee technische Weiterentwicklung einer TAS zuständig ist, in Englisch: Test Automation Engineer)
TAF	Testautomatisierungsframework (für die Testautomatisierung benötigte Umgebung inklusive der Testrahmen und weiterer Artefakte wie Testbibliotheken, in Englisch: Test Automation Framework)
TAM	Testautomatisierungsmanager (Person, die für die Planung und Überwachung der Neu- und Weiterentwicklung einer TAS zuständig ist, in Englisch: Test Automation Manager)
TAS	Testautomatisierungslösung (Realisierung/Implementierung einer TAA inklusive Testrahmen und Testbibliotheken, in Englisch: Test Automation Solution)
UI	Benutzungsschnittstelle (in Englisch: User Interface)

1. Einführung in die Testautomatisierung und ihre Ziele – 30 min

Begriffe

API-Testen, CLI-Testen, GUI-Testen, System unter Test (SUT), Testautomatisierung, Testautomatisierungsarchitektur (TAA), Testautomatisierungsframework (TAF), Testautomatisierungsstrategie, Testskript, Testmittel

Lernziele für „Einführung in die Testautomatisierung und ihre Ziele“

1.1 Zweck der Testautomatisierung

ALTA-E-1.1.1 (K2) Erläutern der Ziele, Vorteile, Nachteile und Beschränkungen der Testautomatisierung

1.2 Erfolgsfaktoren bei der Testautomatisierung

ALTA-E-1.2.1 (K2) Ermitteln technischer Erfolgsfaktoren für ein Testautomatisierungsprojekt

1.1 Zweck der Testautomatisierung

Im Kontext des Softwaretestens bezeichnet der Begriff "Testautomatisierung" das Testen (Prüfen) von Software mittels spezieller Werkzeuge oder Software und umfasst eine oder mehrere der folgenden Aufgaben:

- Steuerung und Erfüllung von Vorbedingungen
- Ausführung von Tests
- Vergleich der tatsächlichen mit den erwarteten Ergebnissen

Es ist eine bewährte Vorgehensweise, die für das Testen zu verwendende Software vom zu testenden System (SUT) zu trennen, um eine gegenseitige Beeinflussung zu vermeiden. Ausnahmen von dieser Regel bilden z. B. eingebettete Systeme, bei denen die Testsoftware im SUT bereitgestellt werden muss.

Von einer Testautomatisierung wird erwartet, dass sie die einheitliche und wiederholte Ausführung von Testfällen gegen verschiedene Versionen des SUT und/oder in verschiedenen Umgebungen unterstützt. Testautomatisierung ist jedoch mehr als nur ein Mechanismus für die Ausführung einer Testsuite ohne menschliche Intervention. Sie umfasst den gesamten Prozess des Entwurfs der Testmittel. Zu den relevanten Testmittel zählen unter anderem:

- Software
- Dokumentation
- Testfälle
- Testumgebungen
- Testdaten

Die Testmittel werden unterstützend für die Ausführung der folgenden Testaktivitäten verwendet:

- Implementierung automatisierter Testfälle
- Überwachung und Steuerung der Ausführung automatisierter Testfälle
- Interpretation, Berichterstattung und Protokollierung der Ergebnisse automatisierter Testfälle

Bei einer Testautomatisierung gibt es verschiedene Ansätze um mit dem SUT zu interagieren bzw. zu kommunizieren:

- Testen über die öffentlichen Schnittstellen von Klassen, Modulen oder Bibliotheken des SUT (API-Testen)
- Testen über die UI oder die Kommandozeilenschnittstelle des SUT (GUI- oder CLI-Tests)
- Testen über einen Dienst (Service) oder ein Protokoll

Ziele einer Testautomatisierung umfassen:

- Erhöhung der Testeffizienz
- Erhöhung der Funktionsüberdeckung
- Senkung der Gesamtkosten für Tests
- Durchführung von Tests, die manuell nicht oder nicht zielführend durchgeführt werden können
- Verkürzung der Testausführungszeit
- Erhöhung der Testausführungshäufigkeit/Verkürzung der für die Testzyklen benötigten Zeit

Zu den Vorteilen einer Testautomatisierung zählen:

- Je Software-Version können mehr Tests durchgeführt werden.
- Es lassen sich Tests entwickeln, die sich manuell nicht durchführen lassen (beispielsweise Echtzeit-Tests, verteilte Tests, parallele Tests).
- Tests können komplexer sein.
- Tests laufen schneller.
- Tests sind weniger anfällig für Bedienfehler.
- Testressourcen werden effektiv und effizienter genutzt.
- Es gibt eine schnellere Rückmeldung bezüglich der Softwarequalität.
- Testautomatisierung trägt zu einer größeren Ausfallsicherheit des Systems (z. B. Wiederholbarkeit, Konsistenz) bei.
- Testautomatisierung verbessert die Konsistenz der Tests.

Mögliche Nachteile der Testautomatisierung sind:

- Es fallen zusätzliche Kosten an.
- Anfangsinvestitionen für die Entwicklung einer TAS werden benötigt.
- Es werden zusätzliche Technologien benötigt.
- Das Testteam benötigt Kompetenzen im Bereich der Softwareentwicklung und Testautomatisierung.
- Es besteht ein ständiger Wartungsbedarf der TAS.
- Testautomatisierung kann von den eigentlichen Testzielen ablenken, z. B. durch Fokussierung auf die Automatisierung von Testfällen zu Lasten der Testausführung.
- Tests können komplexer werden.
- Durch Testautomatisierung können zusätzliche Fehler eingeschleust werden.

Zu den Beschränkungen der Testautomatisierung gehören:

- Nicht alle manuellen Tests lassen sich automatisieren.
- Nur maschineninterpretierbare Ergebnisse sind prüfbar.
- Es lassen sich nur tatsächliche Ergebnisse prüfen, die von einem automatisierten Testorakel verifiziert werden können.
- Testautomatisierung ist kein Ersatz für exploratives Testen.

1.2 Erfolgsfaktoren für die Testautomatisierung

Die nachstehend genannten Erfolgsfaktoren beziehen sich auf laufende Testautomatisierungsprojekte. Ihr Fokus liegt auf der Sicherstellung des langfristigen Erfolgs eines Testautomatisierungsprojektes. Erfolgsfaktoren für die Pilotierung von Testautomatisierungsprojekten werden hier nicht berücksichtigt.

Zu den wichtigsten Erfolgsfaktoren für die Testautomatisierung zählen:

Testautomatisierungsarchitektur (TAA)

Die Testautomatisierungsarchitektur (TAA) orientiert sich an der Architektur des Softwareprodukts. Es muss klar sein, welche funktionalen und nicht-funktionalen Anforderungen die TAA unterstützen soll. In der

Regel sind dies die wichtigsten Anforderungen des Softwareprodukts.

Häufig stehen beim Entwurf der TAA Wartbarkeit/Änderbarkeit, Performanz und Erlernbarkeit im Vordergrund. (Weitere Informationen zu nichtfunktionalen Merkmalen finden sich in der ISO/IEC 25000:2014.) Hierbei ist es hilfreich Softwareentwickler und -architekten einzubeziehen, die die Architektur des SUT verstehen.

Testbarkeit des SUT

Die Auslegung auf Testbarkeit des SUT zielt darauf ab, die automatisierte Testausführung zu unterstützen. Im Fall des automatisierten GUI-Testens kann dies beispielsweise bedeuten, die Elemente und Daten für die Interaktion mit der GUI soweit wie möglich von ihrem Layout zu entkoppeln. Im Fall des automatisierten API-Testens wird es mitunter notwendig, zusätzliche (Test-)Schnittstellen zum SUT einzubinden oder weitere SUT-Schnittstellen (von Klassen, Modulen/Komponenten etc. oder der Kommandozeile) öffentlich bereitzustellen.

Die Bereiche (Klassen, Module, funktionale Einheiten) des SUT, die eine hohe Testbarkeit aufweisen, sollten zuerst von der Automatisierung adressiert werden. Ein wichtiger Erfolgsfaktor für die Testautomatisierung ist die möglichst einfache Implementierung und Verteilung automatisierter Testskripte. Mit diesem Ziel im Hinterkopf und auch für den erfolgreichen Nachweis der Durchführbarkeit muss der Testautomatisierungsentwickler (TAE) die Module oder Komponenten des SUT ermitteln, die sich durch Automatisierung einfach testen lassen, und dort mit der Automatisierung beginnen.

Testautomatisierungsstrategie

Es bedarf einer pragmatischen und konsistenten Testautomatisierungsstrategie, die der Wartbarkeit und Konsistenz des SUT Rechnung trägt.

Möglicherweise ist die gewählte Testautomatisierungsstrategie nicht gleichermaßen auf alte und neue/geänderte Bereiche des SUT anwendbar. Bei der Ausarbeitung der Testautomatisierungsstrategie sind daher Kosten, Vorteile und Risiken der Anwendung der Strategie auf die unterschiedlichen Bereiche des SUT zu berücksichtigen.

Die Testautomatisierungsstrategie muss zudem die Vergleichbarkeit der Testergebnisse automatisierter Testfälle berücksichtigen, die über verschiedene Schnittstellen (bspw. API und GUI) des SUT durchgeführt wurden.

Testautomatisierungsframework (TAF)

Es bedarf eines Testautomatisierungsframeworks (TAF), das einfach zu verwenden, gut dokumentiert und gut wartbar ist, sowie einen einheitlichen Ansatz für die Automatisierung von Tests unterstützt.

Ein wartbares und benutzerfreundliches TAF ergibt sich unter anderem durch folgende Faktoren:

- Implementierung von Berichtsfunktionen: Die Testberichte müssen Informationen zur Qualität des SUT liefern (bestanden/fehlgeschlagen/Fehler/nicht ausgeführt/abgebrochen, statistische Daten usw.). Die Berichte müssen diese Informationen für verschiedene Stakeholder (die beteiligten Tester, Testmanager, Entwickler, Projektmanager und andere Beteiligte) in adäquater Form aufbereiten, damit diese einen Überblick über die Qualität des SUT bekommen.
- Unterstützung einer einfachen Fehlersuche und -beseitigung: Zusätzlich zur Ausführung und Protokollierung von Tests muss das TAF eine einfache Möglichkeit der Fehlersuche bei fehlgeschlagenen Tests bieten. Das Fehlschlagen von Tests kann folgende Ursachen haben:
  - Fehler im SUT
  - Fehler in der Testautomatisierungslösung (TAS)
  - Probleme mit den Testskripten oder der Testumgebung
- Korrekte Einrichtung der Testumgebung: Für die automatisierte Testausführung wird eine dedizierte Testumgebung benötigt, welche die verschiedenen Testwerkzeuge konsistent integriert.

Mangelnde Eingriffs- oder Konfigurationsmöglichkeiten der automatisierten Testumgebung oder der verwendeten Testdaten führen unter Umständen dazu, dass das Setup der Testskripte nicht gemäß den Anforderungen für die Testausführung durchgeführt werden kann. Dies kann wiederum dazu führen, dass keine verlässlichen oder sogar falsche Testergebnisse erzeugt werden (falsch-positive oder falsch-negative Ergebnisse).

- Dokumentation der automatisierten Testfälle: Die Ziele der Testautomatisierung müssen klar definiert sein. Welche Teile der Software sind in welchem Umfang zu testen? Welcher Testautomatisierungsansatz ist zu nehmen? Welche (funktionalen und nicht-funktionalen) Merkmale des SUT sind durch die Automatisierung zu testen. Diese Ziele müssen klar beschrieben und dokumentiert werden.
- Rückverfolgbarkeit der automatisierten Tests: Das TAF muss die Rückverfolgbarkeit einzelner Testfallschritte zu Testfällen durch den TAE unterstützen.
- Einfache Wartung: Im Idealfall ist der Wartungsaufwand der automatisierten Testfälle so gering, dass die Wartung keinen großen Teil der eigentlichen Testautomatisierungsarbeit beansprucht. Generell muss der Wartungsaufwand der Testautomatisierung in einem sinnvollen Verhältnis zum Umfang der Änderungen am SUT stehen. Dazu müssen die Testfälle einfach analysierbar, änder- und erweiterbar sein. Zudem sollte der Wiederverwendungsgrad automatisierter Testmittel hoch sein, um die Anzahl der anzupassenden Artefakte zu reduzieren.
- Aktualität der automatisierten Testfälle: Wenn neue oder geänderte Anforderungen das Fehlschlagen von Tests oder kompletten Testsuites zur Folge haben, sollten die fehlgeschlagenen Tests nicht verworfen, sondern von Fehlern befreit werden.
- Planung der Softwareverteilung: Es muss sichergestellt werden, dass Testskripte problemlos verteilt, geändert und erneut verteilt werden können.
- Außerbetriebnahme von automatisierten Tests: Das TAF unterstützt die einfache Außerbetriebnahme nicht mehr hilfreicher oder benötigter Tests.
- Überwachung und Wiederherstellung des SUT: In der Regel muss das SUT für eine kontinuierliche Ausführung eines oder mehrere Testfälle fortwährend überwacht werden. Wenn beim SUT ein schwerer Fehler (z. B. ein Absturz) auftritt, muss das TAF in der Lage sein, den aktuellen Testfall zu überspringen, das SUT in einen konsistenten Zustand wiederherzustellen und mit der Ausführung des nächsten Testfalls fortzusetzen.

Die Wartung des Testautomatisierungscode kann sehr komplex, der damit verbundene Wartungsaufwand hoch sein. Grund dafür sind die verschiedenen verwendeten Testwerkzeuge, die verschiedenen genutzten Arten der Verifizierung und Validierung und die unterschiedlichen zu wartenden Testmittel (z. B. Testeingabedaten, Testorakel, Testberichte). Es ist nicht ungewöhnlich, dass Test- und Entwicklungscode denselben Umfang haben. Daher ist es unerlässlich, dass der Testcode gut und einfach wartbar ist.

Nachfolgende Empfehlungen wirken sich auf den Wartungsaufwand positiv aus:

- Es sollte Testcode mit technischen Abhängigkeiten zu den verwendeten Schnittstellen vermieden werden, wie z. B. Testcode, der von Änderungen an der grafischen GUI oder an unwesentlichen Teilen der API betroffen wäre.
- Die Testautomatisierung sollte nicht anfällig gegenüber Datenänderungen sein oder eine hohe Abhängigkeit von bestimmten Datenwerten aufweisen, wie beispielsweise Testeingaben, die von Ausgaben anderer Tests abhängen.
- Bei der Automatisierungsumgebung sollte darauf geachtet werden, dass sie unabhängig vom technischen Kontext ist, beispielsweise von Datum und Uhrzeit des Betriebssystems, von Lokalisierungsparameter des Betriebssystems oder vom Inhalt anderer Anwendungen. In diesem Fall ist es besser, Testplatzhalter zu verwenden um den Kontext für die Testumgebung zu etablieren und zu kontrollieren.

Je mehr Erfolgskriterien erfüllt sind, desto wahrscheinlicher ist der Erfolg des gesamten Testautomatisierungsprojekts. Nicht alle Kriterien sind zwingend notwendig, und in der Praxis lassen sich selten alle erfüllen. Vor dem Start eines Testautomatisierungsprojekts müssen die Erfolgsaussichten für das Projekt analysiert werden. Dazu ist unter Berücksichtigung der Risiken des gewählten Ansatzes und des Projektkontexts zu prüfen, welche Kriterien erfüllt und welche nicht erfüllt sind. Sobald die TAA steht, muss untersucht werden, welche Elemente fehlen bzw. Nacharbeiten erfordern.

2. Vorbereitungen für die Testautomatisierung – 165 min

Begriffe
Grad der Intrusion, Platzhalter, Testbarkeit, Testausführungswerkzeug, Testautomatisierungsmanager, Test Hook, Treiber

Lernziele für „Vorbereitungen für die Testautomatisierung“

2.1 SUT-Faktoren mit Einfluss auf die Testautomatisierung
ALTA-E-2.1.1 (K4) Analyse eines zu testenden Systems (SUT) zur Ermittlung der geeigneten Automatisierungslösung

2.2 Bewertung und Auswahl von Werkzeugen
ALTA-E-2.2.1 (K4) Analyse von Testautomatisierungswerkzeugen für ein Projekt und Protokollierung der technischen Erkenntnisse und Empfehlungen

2.3 Auslegung auf Testbarkeit und Automatisierung
ALTA-E-2.3.1 (K2) Verstehen der Methoden der „Auslegung auf Testbarkeit“ und der „Auslegung auf Testautomatisierung“, die auf das SUT anwendbar sind

2.1 SUT-Faktoren mit Einfluss auf die Testautomatisierung

Bei der Bewertung des Kontexts des SUT und seiner Umgebung müssen Faktoren mit Einfluss auf die Testautomatisierung ermittelt werden, um eine angemessene Lösung zu erarbeiten. Dabei kann es sich um folgende Faktoren handeln:

- SUT-Schnittstellen
  Automatisierte Testfälle lösen Aktionen am SUT aus. Dazu muss das SUT Schnittstellen anbieten, über die es sich steuern und beobachten lässt. Das können UI-Steuerelemente, aber auch Softwareschnittstellen (API) auf niedrigerer Ebene sein. Manche Testfälle verwenden zudem Schnittstellen auf der Kommunikationsebene (z. B. mittels TCP/IP, USB oder proprietären Messaging-Schnittstellen).
  Durch eine sinnvolle Dekomposition des SUT ist es möglich auf unterschiedlichen Teststufen unterschiedliche Schnittstellen anzusprechen. So sind auf Komponententeststufe die Tests oft einfacher zu automatisieren im Vergleich zu anderen Teststufen. Bietet das SUT auf einer Teststufe keine adäquaten Schnittstellen für die Automatisierung an, so sind mitunter dedizierte (Test-)Schnittstellen bereitzustellen (so genannte "Test Hooks"), um dennoch eine Testautomatisierung auf diesen Teststufen zu etablieren.
- Fremdsoftware
  Häufig besteht das SUT nicht nur aus selbst entwickelter Software, sondern umfasst auch Fremdsoftware. In einigen Projekten ist es notwendig, auch diese Fremdsoftware zu testen. Wird nun auch für den Test der Fremdsoftware eine Testautomatisierung eingesetzt, so wird unter Umständen sogar eine andere Testautomatisierungslösung oder eine Testautomatisierungsstrategie benötigt (bspw. unter Verwendung einer API für den Test der Fremdsoftware, während die proprietären Teile des SUT über eine GUI getestet werden).
- Grad der Intrusion
  Verschiedene Testautomatisierungsansätze weisen durch die Verwendung verschiedener Werkzeuge unterschiedliche Intrusionsgrade auf. Je mehr Änderungen zum Zwecke der automatisierten Testdurchführung am SUT vorgenommen werden müssen, desto höher ist der Grad der Intrusion. So geht die Verwendung dedizierter Test-/Softwareschnittstellen mit einem hohen Intrusionsgrad einher, während die Verwendung bestehender UI-Elemente einen geringeren Grad der Intrusion aufweist. Werden Hardwareelemente des SUT (z. B. Tastaturen, Handschalter, Touchscreens, Kommunikationsschnittstellen)

verwendet werden, ist der Grad der Intrusion sogar noch höher.

Höhere Intrusionsgrade weisen das Risiko auf, falsch-positive Ergebnisse (Fehlalarme) zu erzeugen. Die TAS kann Fehlerwirkungen aufweisen, die auf einen hohen Intrusionsgrad zurückzuführen sind. Wird das SUT hingegen in der Betriebsumgebung ausgeführt, treten diese Fehlerwirkungen aber wahrscheinlich nicht auf. Dennoch ist ein Testautomatisierungsansatz mit einem hohen Intrusionsgrad in der Regel eine einfache zu realisierende Lösung.

- Unterschiedliche SUT-Architekturen
  Unterschiedliche SUT-Architekturen erfordern gemeinhin unterschiedliche Testautomatisierungsarchitekturen und -lösungen. Für ein SUT, das in C++ unter Verwendung der COM-Komponententechnologie geschrieben ist, wird ein anderer Ansatz benötigt als für ein in Python geschriebenes SUT. Unterschiedliche SUT-Architekturen können dennoch von derselben Testautomatisierungsstrategie adressiert werden. Man spricht dann auch von einer hybriden Testautomatisierungsstrategie.
- Größe und Komplexität des SUT
  Die Größe und die Komplexität, sowie geplante Weiterentwicklungen des SUT müssen berücksichtigt werden. Für ein kleines, vergleichsweise einfaches SUT ist ein entsprechend leichtgewichtiger Testautomatisierungsansatz möglicherweise eher zielführend als ein komplexer und hochflexibler Ansatz. Umgekehrt gilt: Bei einem großen und komplexen SUT ist es womöglich keine gute Idee, mit einem starren, nicht anpassbaren Ansatz zu arbeiten. Als Übergangslösung kann es jedoch ratsam sein, zunächst mit einem kleinen, recht einfachen Ansatz zu beginnen und schrittweise auf einen komplexeren Testautomatisierungsansatz zu migrieren (siehe dazu auch Kapitel 3).

Einige der hier beschriebenen Faktoren (z. B. Größe und Komplexität, verfügbare Softwareschnittstellen) sind erst bekannt, wenn das SUT bereitgestellt wird. Oftmals beginnt die Entwicklung der Testautomatisierung jedoch bevor das SUT verfügbar ist. In diesem Fall müssen einige dieser Faktoren geschätzt werden. Zudem obliegt es dem TAE bereits a priori dedizierte (Test)-Schnittstellen für die Testautomatisierung festzulegen (siehe dazu Abschnitt 2.3).

Auch wenn das SUT noch nicht (oder nur teilweise) bereitgestellt wurde, können bereits entwicklerseitige Arbeitsergebnisse herangezogen werden, um die Planung und Entwicklung der Testautomatisierung frühzeitig zu beginnen:

- Wenn die Anforderungen (funktional oder nicht-funktional) des SUT bekannt sind, können Kandidaten für die Automatisierung aus diesen Anforderungen ausgewählt und die benötigten Testmittel zum Testen dieser Anforderungen identifiziert werden. Die Planung der Automatisierung kann für diese Kandidaten beginnen. Dies schließt auch die Identifikation der Anforderungen für die Automatisierung sowie der Testautomatisierungsstrategie ein.
- Wenn die Architektur und der technische Entwurf entwickelt werden, kann mit dem Entwurf der für das Testen benötigten Softwareschnittstellen begonnen werden.

2.2 Bewertung und Auswahl von Werkzeugen

Für die Auswahl und Bewertung von Werkzeugen ist in erster Linie der Testautomatisierungsmanager (TAM) verantwortlich. Es gehört jedoch zu den Aufgaben des TAE, dem TAM diesbezüglich zuzuarbeiten. So werden die Werkzeugevaluation sowie die schlussendliche Auswahlempfehlung zumeist vom TAE durchgeführt. Das Konzept der Werkzeugbewertung und -auswahl wurden bereits im Grundkurs Foundation Level [ISTQB-FL] vorgestellt. Weitere Details dieses Prozesses werden zudem im Lehrplan „Advanced Level – Test Manager Syllabus“ [ISTQB-AL-TM] beschrieben.

Der TAE wird in den gesamten Prozess der Werkzeugbewertung und -auswahl eingebunden und leistet für die folgenden Aktivitäten wertvolle Beiträge:

- Bewertung der organisatorischen Reife und der Möglichkeiten einer Testwerkzeugunterstützung

- Bewertung realistischer Zielvorstellungen durch eine Testwerkzeugunterstützung
- Ermittlung und Sammlung von Informationen zu potentiell einsetzbaren Werkzeugen
- Analyse von Werkzeugeneigenschaften unter Berücksichtigung der Ziele und Projektrestriktionen
- Abschätzung des Kosten-Nutzen-Verhältnisses auf Basis eines soliden Business Case
- Aussprechen von Empfehlungen für geeignete Werkzeuge
- Ermittlung der Werkzeugkompatibilität dem SUT (bzw. seinen Komponenten).

Funktionale Testautomatisierungswerkzeuge erfüllen häufig nicht alle Erwartungen oder greifen nicht in allen Situationen, die bei einem Automatisierungsprojekt auftreten können. Nachstehend sind einige Probleme aufgeführt, die den Einsatz von Testautomatisierungswerkzeugen erschweren (die Liste ist definitiv nicht vollständig):

Problem	Beispiele	Mögliche Lösungen
Die Werkzeugschnittstellen passen nicht zu anderen, bereits genutzten Werkzeugen.	• Das Testmanagementwerkzeug wurde aktualisiert und die genutzte Schnittstelle hat sich geändert. • Die Informationen aus der Beratung vor dem Kauf waren falsch und nicht alle Daten lassen sich an das Berichtswerkzeug übertragen.	• Beachtung von Release Notes vor dem Update und Testen des Systems bei großen Migrationsprozessen vor der Überführung in die Produktion. • Vor-Ort-Demonstration des Werkzeugs unter Verwendung des echten SUT zu erhalten. • Support vom Anbieter und/oder durch Recherchen in Foren der Benutzer-Community.
Einige Eigenschaften des SUT verändern sich so, dass sie vom Testwerkzeug nicht unterstützt werden.	• Die Entwicklungsabteilung hat ein Update auf die neueste Java-Version vorgenommen.	• Synchronisierung der Upgrades für die Entwicklungs-/ Testumgebung und das Testautomatisierungswerkzeug.
GUI-Elemente können nicht erfasst werden	• Das GUI-Element ist sichtbar, aber das Testautomatisierungswerkzeug kann nicht mit ihm interagieren.	• Bei der Entwicklung nur mit bekannten Technologien oder GUI-Elemente arbeiten. • Durchführen eines Pilotprojektes vor dem Kauf eines Testautomatisierungswerkzeugs. • Entwickler definieren Standards für GUI-Elemente.
Die Anwendung des Werkzeugs ist sehr kompliziert	• Das Werkzeug besitzt einen großen Funktionsumfang, von dem aber nur ein Teil benötigt wird.	• Funktionsumfang durch das Entfernen nicht benötigter Funktionen aus der Funktionspalette zu begrenzen. • Wählen einer Lizenz, die Ihrem Bedarf entspricht. • Suche nach alternativen Werkzeugen mit stärkerem Fokus auf den benötigten Funktionsumfang

Konflikt mit anderen Systemen	• Nach der Installation anderer Software funktioniert das Testautomatisierungswerkzeug nicht mehr - oder umgekehrt.	• Prüfung der Release Notes oder technischen Voraussetzungen vor der Installation. • Bestätigung vom Anbieter, dass andere Werkzeuge nicht beeinträchtigt werden. • Recherche in Benutzer-Foren.
Beeinträchtigung des SUT	• Während/nach Nutzung des Testautomatisierungswerkzeugs reagiert das SUT anders (z. B. längere Reaktions-zeiten).	• Verwendung eines Werkzeugs, das keine Änderungen am SUT vornimmt (z. B. Installation von Bibliotheken usw).
Zugriff auf Code	• Das Testautomatisierungswerkzeug ändert Teile des Quellcodes.	• Verwendung eines Werkzeugs, das keine Änderungen am Quellcode vornimmt (z. B. Installation von Bibliotheken usw.).
Limitierte Ressourcen (vorrangig in eingebetteten Umgebungen)	• Die Testumgebung hat nur begrenzte oder keine freien Ressourcen mehr (z. B. Arbeitsspeicher).	• Lesen von Release Notes und Bestätigung vom Werkzeuganbieter, dass dies nicht zu Problemen führt. • Mögliche Probleme in Benutzer-Foren ansprechen.
Updates	• Beim Update werden nicht alle Daten migriert oder automatisierte Testskripts, Daten oder Konfigurationen beschädigt. • Für Upgrades wird eine andere (bessere) Umgebung benötigt.	• Testen von Upgrades in der Testumgebung sowie beim Anbieter anfragen, ob Probleme bei der Migration zu erwarten sind. • Ermittlung der Voraussetzungen für das Update und Abwägung, ob das Update den Aufwand rechtfertigt. • Recherche und Unterstützung durch Benutzer-Foren.
Sicherheit	• Das Testautomatisierungswerkzeug benötigt Informationen, die dem TAE nicht zur Verfügung stehen.	• Dem TAE muss der Zugang zu diesen Informationen gewährt werden.
Inkompatibilität zwischen verschiedenen Umgebungen und Plattformen	• Die Testautomatisierung funktioniert nicht in allen Umgebungen bzw. auf allen Plattformen.	• Implementierung automatisierter Tests so, dass die Unabhängigkeit von Werkzeugen maximiert und die Kosten für den Einsatz mehrerer Werkzeuge minimiert werden.

2.3 Auslegung auf Testbarkeit und Automatisierung

Parallel zu Entwurf und Implementierung der Funktionen des SUT sollte das SUT auf Testbarkeit geprüft werden, wie z. B. die Verfügbarkeit von Schnittstellen, die das Testen unterstützen, um beispielsweise die Steuerung und Beobachtbarkeit des SUT zu ermöglichen. Häufig wird zu Sicherstellung der Testbarkeit des SUT die Unterstützung des TAE durch den Systemarchitekten angefordert. Theoretisch könnte der Systemarchitekt dies auch alleine bewerkstelligen, da Testbarkeit (als Untermerkmal des Qualitätsmerkmals Wartbarkeit) nur ein weiteres nicht-funktionales Merkmal des Systems ist.

Die Auslegung auf Testbarkeit umfasst mehrere Aspekte:

- Beobachtbarkeit: Das SUT muss Schnittstellen bieten, die Einblicke in das System ermöglichen. Testfälle können diese dann nutzen, um z. B. zu prüfen, ob das tatsächliche Verhalten dem erwarteten Verhalten entspricht.
- Steuerung bzw. Steuerbarkeit: Das SUT muss Schnittstellen bieten, die für die Ausführung von Aktionen am SUT genutzt werden können. Das können UI-Elemente, Funktionsaufrufe, Kommunikationselemente (z. B. TCP/IP- oder USB-Protokoll), elektronische Signale (für physische Schalter) usw. sein.
- Klar definierte Architektur: Der dritte wichtige Aspekt der Auslegung auf Testbarkeit ist eine Architektur, die durch Bereitstellung klarer und eindeutiger Schnittstellen auf allen Teststufen für Kontrolle und Transparenz sorgen.

Der TAE stellt sicher, dass das SUT auf effektive (Testen der richtigen Bereiche und Finden kritischer Fehler) und effiziente (ohne zu hohen Aufwand) Weise durch die Testautomatisierung getestet wird. Werden dedizierte Test-/Softwareschnittstellen benötigt, müssen diese vom TAE spezifiziert und vom Entwickler für die Testautomatisierung implementiert werden. Generell ist es ratsam, die Testbarkeit eines SUT frühzeitig im Projekt zu adressieren. Insbesondere wenn dedizierte Test-/Softwareschnittstellen benötigt werden, muss der damit verbundene Entwicklungsaufwand budgetiert und koordiniert werden.

Einige Beispiele für Softwareschnittstellen, die das Testen unterstützen:

- Die leistungsfähigen Skripterstellungsfunktionen moderner Kalkulationstabellen.
- Die Verwendung von Platzhaltern zur Simulation von Software und/oder Hardware (z. B. elektronische Finanztransaktionen, Softwaredienste, dedizierte Server, elektronische Platinen, mechanische Teile), die noch nicht verfügbar oder zu teuer ist, ermöglicht das Testen von Software, ohne dass diese tatsächlichen Software-/Handwarekomponenten verfügbar sind.
- Softwareschnittstellen (oder Platzhalter und Treiber) können für das Testen von Fehlerbedingungen genutzt werden, wie z. B. an einem Gerät mit internem Festplattenlaufwerk (HDD). Die Software, die diese HDD steuert (der Treiber) muss auf Fehler und die HDD selbst auf Verschleiß getestet werden. Solange zu warten bis die HDD tatsächlich ausfällt ist ineffizient und unzuverlässig. Durch Implementierung von Softwareschnittstellen, die defekte oder langsame HDDs simulieren, lässt sich verifizieren, ob die Treibersoftware richtig funktioniert und z. B. eine Fehlermeldung ausgibt, einen neuen Versuch startet.
- Dedizierte (Test-)Schnittstellen erlauben es, ein SUT zu testen, wenn (noch) keine grafische Benutzeroberfläche verfügbar ist. Eingebettete Software in technischen Anlagen muss häufig die Innentemperatur überwachen und den Start einer Kühlfunktion auslösen, wenn die Temperatur einen bestimmten Wert übersteigt. Das lässt sich mittels einer Softwareschnittstelle zur Angabe der Temperatur auch ohne Hardware testen. Die Verwendung dedizierter (Test-)Schnittstellen gilt ohnehin zumeist als der effizientere Testautomatisierungsansatz.
- Zustandsbasierte Tests dienen der Evaluierung des Zustandsverhaltens des SUT. Ob sich das SUT im richtigen Zustand befindet, lässt sich z. B. durch Abfrage des Zustands über eine für diesen Zweck entwickelte Softwareschnittstelle prüfen (auch wenn das ein Risiko birgt; siehe dazu den Grad der Intrusion in Abschnitt 2.1).

Bei der Auslegung für die Automatisierung muss Folgendes berücksichtigt werden:

- Die Kompatibilität mit bestehenden Testwerkzeugen muss schon in einer frühen Phase gewährleistet werden.
- Die Frage der Kompatibilität von Testwerkzeugen ist entscheidend, da sie die Fähigkeit zur Automatisierung des Testens wichtiger Funktionen beeinträchtigen kann. So verhindert die Inkompatibilität mit einer Netzregelung beispielsweise alle Tests, die diese Netzregelung verwenden.
- Lösungen für eine bessere Auslegung auf Automatisierbarkeit können die Entwicklung von Programmcode und Aufrufe an APIs erfordern.

Die Auslegung auf Testbarkeit ist für einen verlässlichen Testautomatisierungsansatz von größter Bedeutung, durch die auch manuelle Testausführung profitieren kann.

3. Die generische Testautomatisierungsarchitektur – 270 min

Begriffe
Datengetriebenes Testen, generische Testautomatisierungsarchitektur, Mitschnitt (Capture/Playback), modellbasiertes Testen, lineare Skripterstellung, prozessgetriebene Skripterstellung, Testadaptierungsschicht, Testautomatisierungsarchitektur, Testautomatisierungsframework, Testautomatisierungslösung, Testausführungsschicht, Testdefinitionsschicht, Testgenerierungsschicht, schlüsselwortgetriebener Test, strukturierte Skripterstellung

Lernziele für „Die generische Testautomatisierungsarchitektur“

3.1 Einführung in die gTAA
ALTA-E-3.1.1 (K2) Erläutern des Aufbaus der gTAA

3.2 TAA-Entwurf
ALTA-E-3.2.1 (K4) Entwurf einer geeigneten TAA für ein Projekt
ALTA-E-3.2.2 (K2) Erläutern der Schichten in einer TAA
ALTA-E-3.2.3 (K2) Verstehen der Entwurfsüberlegungen für eine TAA
ALTA-E-3.2.4 (K4) Analyse von Faktoren für die Implementierungs-, Nutzungs- und Wartungsanforderungen für eine TAS

3.3 TAS-Entwicklung
ALTA-E-3.3.1 (K3) Anwendung von Komponenten der generischen TAA (gTAA) zur Entwicklung einer TAA für einen speziellen Zweck
ALTA-E-3.3.2 (K2) Erläutern der Faktoren, die bei der Ermittlung der Wiederverwendbarkeit von Komponenten zu berücksichtigen sind

3.1 Einführung in die gTAA

Ein Testautomatisierungsentwickler (TAE) hat die Aufgabe, Testautomatisierungslösungen (TAS) zu entwerfen, zu entwickeln, zu implementieren und zu warten. Dabei ähneln sich Probleme, Fragestellungen und Aufgaben, die bei der Entwicklung von Testautomatisierungslösungen adressiert werden müssen. Diese wiederkehrenden Konzepte, Schritte und Ansätze bei der Automatisierung des Testens bilden die Basis der generischen Testautomatisierungsarchitektur (kurz: gTAA).

Die gTAA bildet die Schichten, Komponenten und Schnittstellen ab, die dann zu einer konkreten TAA für eine bestimmte TAS weiterentwickelt werden. Sie ermöglicht einen strukturierten und modularen Ansatz bei der Entwicklung einer Testautomatisierungslösung durch:

- Definieren des funktionalen und nicht-funktionalen Umfangs, der Schichten, Dienste und Schnittstellen einer TAS, um die Realisierung der TAS mittels selbst oder extern entwickelter Komponenten zu ermöglichen
- Unterstützung einfacher Komponenten für die wirksame und effiziente Entwicklung der Testautomatisierung
- Wiederverwendung von Komponenten der Testautomatisierung für eine verschiedene oder sich weitentwickelnde TAS für Softwareproduktreihen und -familien und über Softwaretechnologien und -werkzeuge hinweg
- Vereinfachung der Wartung und Weiterentwicklung von einer TAS
- Definieren der essentiellen Funktionen für einen Benutzer einer TAS

Eine TAS besteht aus der Testumgebung (und ihrer Artefakte) und den Testsuites (eine Gruppe von Testfällen einschließlich der zugehörigen Testdaten). Für die Realisierung einer TAS kann ein

Testautomatisierungsframework (TAF) verwendet werden. Es bietet Unterstützung für die Realisierung der Testumgebung und bietet Werkzeuge, Testrahmen oder unterstützende Bibliotheken.

Es wird empfohlen, dass die TAA einer TAS den folgenden Grundsätzen genügt, die eine einfache Neu- und Weiterentwicklung sowie Wartung der TAS unterstützen:

- Fest definierte Aufgabe: Jede TAS-Komponente hat eine fest definierte Aufgabe. Diese muss komplett in der Komponente gekapselt sein. Mit anderen Worten: Jede Komponente einer TAS ist genau für eine Aufgabe zuständig, z.B. das Erzeugen von Schlüsselwörtern oder Daten, das Erstellen von Testszenarios, die Ausführung von Testfällen, die Protokollierung von Ergebnissen oder die Erzeugung von Ausführungsberichten.
- Erweiterbarkeit (siehe z. B. das Open-Closed-Prinzip von Bertrand Meyer): Jede TAS-Komponente ist offen für Erweiterung, aber geschlossen gegenüber Modifizierung. Dieses Prinzip bedeutet, dass es möglich sein muss, das Verhalten der Komponenten zu modifizieren oder zu erweitern, ohne die Rückwärtskompatibilität zu beeinträchtigen.
- Ersetzbarkeit (siehe z. B. das Liskovsche Substitutionsprinzip): Jede TAS-Komponente ist ersetzbar, ohne Auswirkung auf das Gesamtverhalten der TAS. Die Komponente kann durch eine oder mehrere substituierende Komponenten ersetzt werden, das gezeigte Verhalten darf sich dadurch jedoch nicht ändern.
- Komponentensegregation (siehe z. B. das Interface-Segregation-Prinzip oder Schnittstellenaufteilungsprinzip von R. C. Martin): Statt einer allgemeinen Komponente mit mehreren Aufgaben ist es besser, spezifischere Komponenten zu haben. Das erleichtert die Ersetzung und Wartung durch Wegfall unnötiger Abhängigkeiten.
- Abhängigkeitsumkehr: Die Komponenten einer TAS müssen statt von Details niederer Ebenen von Abstraktionen abhängen. Mit anderen Worten: Die Komponenten dürfen nicht von spezifischen automatisierten Testszenarios abhängen.

Eine TAS, die auf der gTAA basiert, wird in der Regel durch ein Set von Werkzeugen, ihre Plug-Ins und/oder Komponenten implementiert. Dabei muss beachtet werden, dass die gTAA anbieterneutral ist: Sie schreibt keine konkrete Methode, Technologie oder Werkzeug für die Realisierung einer TAS vor. Die gTAA kann mit einem beliebigen Softwareentwicklungsansatz implementiert werden, z. B. strukturiert, objektorientiert, serviceorientiert, modellgetrieben, sowie mit beliebigen Softwaretechnologien und -werkzeugen. Eine TAS wird in der Praxis häufig mit Werkzeugen von der Stange implementiert, benötigt aber in der Regel SUT-spezifische Zusätze und/oder Anpassungen.

Sonstige Richtlinien und Referenzmodelle in Bezug auf TAS sind Softwareentwicklungsstandards für den gewählten SDLC (Software Development Lifecycle), Programmiertechniken, Formatierungsstandards usw. Die Lehre von Softwareentwicklung im Allgemeinen ist nicht Bestandteil dieses Lehrplans. Von einem TAE werden jedoch Kompetenzen, Erfahrung und Fachwissen in der Softwareentwicklung erwartet.

Zudem benötigt ein TAE Wissen über branchenübliche Programmier- und Dokumentationsstandards und Best Practices, auf die er bei der Entwicklung einer TAS zurückgreifen kann. Diese Praktiken können die Wartbarkeit, Zuverlässigkeit und Sicherheit der TAS erhöhen. Solche Standards sind in der Regel domänenspezifisch. Zu den verbreiteten Standards zählen:

- MISRA für C oder C++
- JSF-Standard für C++
- AUTOSAR-Regeln für MathWorks Matlab/Simulink®

3.1.1 Die gTAA im Überblick

Die gTAA ist in horizontale Schichten gegliedert:

- Testgenerierung
- Testdefinition

- Testausführung
- Testadaptierung

Die gTAA (siehe Abbildung 1: Die generische Testautomatisierungsarchitektur) umfasst Folgendes:

- Die Testgenerierungsschicht, die den manuellen oder automatisierten Entwurf von Testfällen unterstützt. Sie bietet die Mittel für den Entwurf von Testfällen.
- Die Testdefinitionsschicht unterstützt die Definition und Implementierung von Testsuites und/oder Testfällen. Sie trennt die Testdefinition vom SUT bzw. von Testsystemtechnologien und Werkzeugen. Sie umfasst Mittel für das Definieren abstrakter und konkreter Tests, die in den Testdaten, Testfällen, Testabläufen und Testbibliothekskomponenten oder Kombinationen dieser Elemente gehandhabt werden.
- Die Testausführungsschicht, die die Ausführung von Testfällen und die Testprotokollierung unterstützt. Sie bietet ein Testausführungswerkzeug für die automatische Ausführung der ausgewählten Tests sowie eine Protokollierungs- und Berichtskomponente.
- Die Testadaptierungsschicht, die den benötigten Code für die Adaptierung der automatisierten Tests für die verschiedenen Komponenten oder Schnittstellen des SUT bietet. Sie stellt verschiedene Adapter für die Verbindungsherstellung zum SUT über APIs, Protokolle, Dienste und sonstiges bereit.
- Zudem verfügt sie über die Schnittstellen für das Projektmanagement, das Konfigurationsmanagement und das Testmanagement in Bezug auf die Testautomatisierung. So wickelt die Schnittstelle zwischen Testmanagement und Testadaptierungsschicht z. B. die Auswahl und Konfiguration der geeigneten Adapter in Bezug auf die gewählte Testkonfiguration ab.

Die Schnittstellen zwischen den gTAA-Schichten und ihren Komponenten sind in der Regel spezifisch und werden daher hier nicht weiter behandelt.

Diese Schichten können, müssen aber nicht in einer TAS vorhanden sein. Zum Beispiel:

- Wenn die Testausführung automatisiert werden soll, müssen die Testausführungs- und die Testadaptierungsschicht verwendet werden. Sie müssen nicht getrennt, sondern können auch gemeinsam realisiert werden, z. B. in Unittest-Frameworks.
- Wenn die Testdefinition automatisiert werden soll, wird die Testdefinitionsschicht benötigt.
- Wenn die Testgenerierung automatisiert werden soll, wird die Testgenerierungsschicht benötigt.

Häufig beginnt man mit der Implementierung einer TAS von unten nach oben wie in der Darstellungsreihenfolge in der Abbildung 1. Andere Ansätze wie die automatisierte Testgenerierung für manuelle Tests können ebenfalls hilfreich sein. Im Allgemeinen ist es ratsam, die TAS inkrementell (z. B. in Sprints) zu implementieren, damit sie so schnell wie möglich eingesetzt werden und ihren Wert unter Beweis stellen kann. Auch Konzeptnachweise im Rahmen des Testautomatisierungsprojekts sind ratsam.

Jedes Testautomatisierungsprojekt muss als Softwareentwicklungsprojekt verstanden, eingerichtet und verwaltet werden und erfordert ein dediziertes Projektmanagement. Das Projektmanagement für die Entwicklung des TAF (d. h. Unterstützung der Testautomatisierung für ganze Unternehmen, Produktfamilien oder Produktreihen) kann vom Projektmanagement für die TAS getrennt werden (d. h. die Testautomatisierung für ein konkretes Produkt).

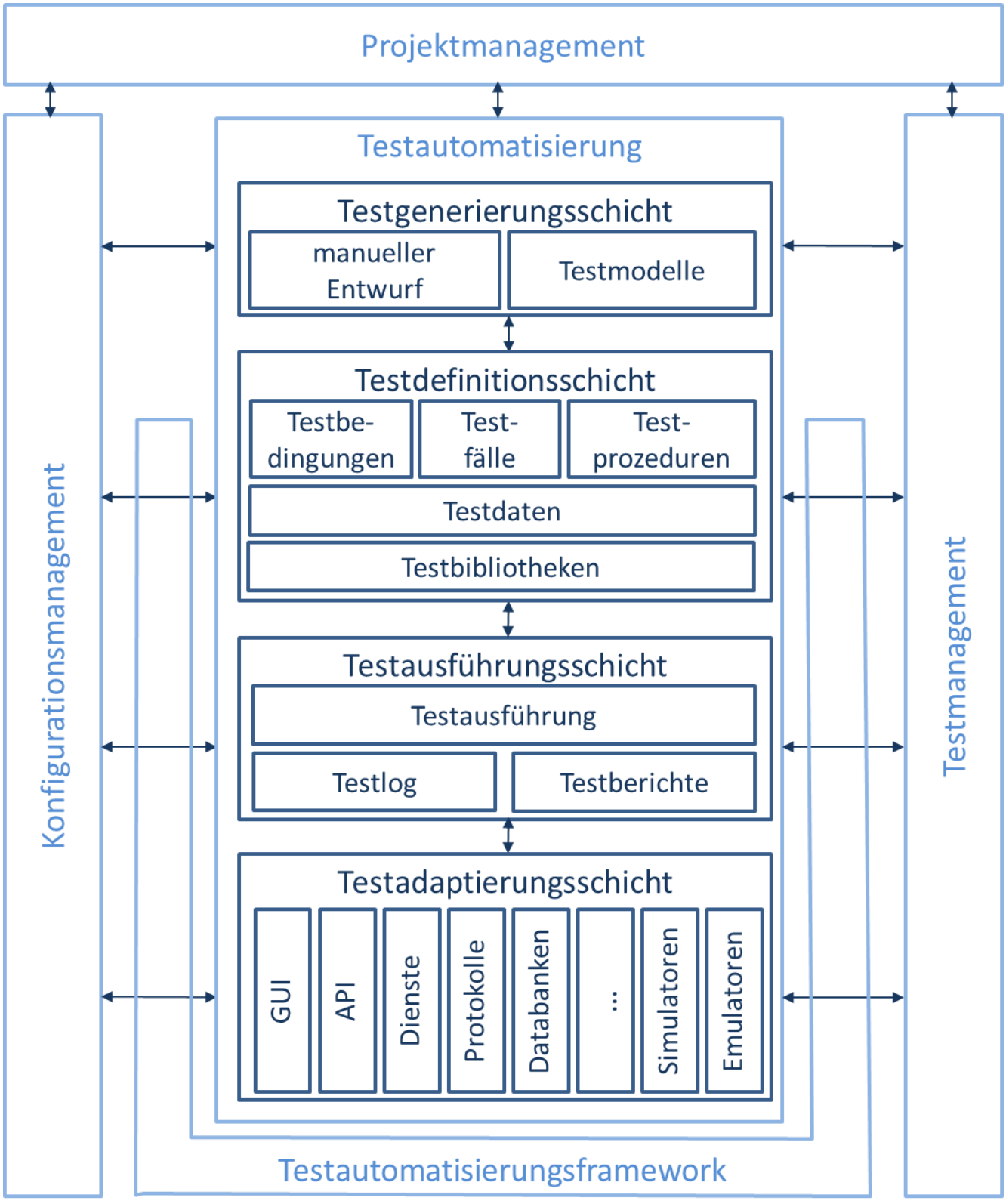

Abbildung 1: Die generische Testautomatisierungsarchitektur

3.1.2 Testgenerierungsschicht

Die Testgenerierungsschicht bietet Unterstützung durch Werkzeuge für folgende Arbeiten:

- manueller Entwurf von Testfällen
- Entwicklung, Erfassung oder Ableitung von Testdaten
- automatische Generierung von Testfällen anhand von Modellen, die das SUT und/oder seine Umgebung definieren (d. h. automatisiertes, modellgestütztes Testen)

Die Komponenten in dieser Schicht dienen folgenden Zwecken:

- Bearbeitung und Navigation in den Testsuite-Strukturen
- Verknüpfen von Testfällen mit Testzielen oder SUT-Anforderungen
- Dokumentieren des Testentwurfs

Für die automatisierte Testgenerierung können zudem folgende Funktionen inbegriffen sein:

- Fähigkeit, das SUT, seine Umgebung und/oder das Testsystem zu modellieren
- Fähigkeit, Testrichtlinien zu definieren und Testgenerierungsalgorithmen zu konfigurieren/parametrisieren
- Fähigkeit, die generierten Tests zum Modell zurückzuverfolgen (Elemente)

3.1.3 Testdefinitionsschicht

Die Testdefinitionsschicht bietet Unterstützung von Werkzeugen für folgende Arbeiten:

- Festlegen von Testfällen (auf abstrakter oder konkreter Ebene)
- Definieren von Testdaten für konkrete Testfälle
- Spezifizieren von Testabläufen für einen Testfall oder eine Gruppe von Testfällen
- Definieren von Testskripten für die Ausführung der Testfälle
- Ermöglichen des Zugangs zu Testbibliotheken, wenn dieser benötigt wird (z. B. bei schlüsselwortgetriebenen Ansätzen)

Die Komponenten in dieser Schicht dienen folgenden Zwecken:

- Partitionieren/Einschränken, Parametrisieren oder Instanziieren von Testdaten
- Spezifizieren von Testsequenzen oder voll ausgeprägten Testverhaltensmustern (einschließlich Steueranweisungen und Ausdrücken), um diese zu parametrisieren und/oder zu gruppieren
- Dokumentieren der Testdaten, Testfälle und/oder Testabläufe

3.1.4 Testausführungsschicht

Die Testausführungsschicht bietet Unterstützung von Werkzeugen für folgende Arbeiten:

- automatische Ausführung von Testfällen
- Protokollierung der Ausführungen von Testfällen
- Aufbereiten der Testergebnisse in Berichten

Die Testausführungsschicht kann aus Komponenten bestehen, die folgende Fähigkeiten bereitstellen:

- Einrichten und Aufräumen des SUT zur Testausführung
- Einrichten und Aufräumen von Testsuites (d. h. Gruppen von Testfällen einschließlich der zugehörigen Testdaten)
- Konfigurieren und Parametrisieren des Testumgebung
- Interpretieren von Testdaten und Testfällen und deren Umwandlung in ausführbare Skripts
- Instrumentierung des Testobjekts bzw. des SUT für die (gefilterte) Protokollierung der Testausführung

und/oder die Fehlereinfügung
- Analysieren der Reaktionen des SUT während der Testausführung zur Steuerung nachfolgender Testläufe
- Validieren der Reaktionen des SUT (Vergleich von tatsächlichen und erwarteten Ergebnissen) anhand der Ergebnisse der automatisch ausgeführten Testfälle
- zeitgerechte Steuerung der automatisierten Ausführung der Tests

3.1.5 Testadaptierungsschicht

Die Testadaptierungsschicht bietet Unterstützung von Werkzeugen für folgende Arbeiten:

- Steuerung des Testrahmens
- Interaktion mit dem SUT
- Überwachung des SUT
- Simulieren oder Emulieren der SUT-Umgebung

Die Testadaptierungsschicht bietet folgende Funktionen:

- Vermitteln zwischen den technologieneutralen Testdefinitionen und den spezifischen Technologieanforderungen des SUT und der Testgeräte
- Anwendung verschiedener technologiespezifischer Adapter für die Interaktion mit dem SUT
- Verteilung der Testausführung über mehrere Testgeräte/Testschnittstellen oder lokale Ausführung von Tests

3.1.6 Konfigurationsmanagement einer TAS

Eine TAS wird im Regelfall in mehreren Iterationen/Versionen entwickelt und muss mit den Iterationen/ Versionen des SUT kompatibel sein. Das Konfigurationsmanagement einer TAS muss Folgendes umfassen:

- Testmodelle
- Testdefinitionen/-spezifikationen einschließlich Testdaten, Testfällen und Bibliotheken
- Testskripte
- Maschine für die Testausführung und ergänzende Werkzeuge und Komponenten
- Testadapter für das SUT
- Simulatoren und Emulatoren für die SUT-Umgebung
- Testergebnisse und Testberichte

Diese Elemente bilden zusammen die Testmittel und müssen in der richtigen Version vorliegen, um der Version des SUT zu entsprechen. In bestimmten Situationen kann es notwendig sein, auf ältere Versionen der TAS zurückzugreifen, z. B. falls Fehler im Produktivbetrieb mit älteren SUT-Versionen reproduziert werden müssen. Ein gutes Konfigurationsmanagement macht dies möglich.

3.1.7 Projektmanagement einer TAS

Weil jedes Testautomatisierungsprojekt ein Softwareprojekt ist, bedarf es desselben Projektmanagements wie jedes andere Softwareprojekt. Bei der Entwicklung einer TAS muss der TAE die Aufgaben für alle Phasen der etablierten SDLC-Methodik durchführen. Zudem muss der TAE wissen, dass die Entwicklungsumgebung der TAS so ausgelegt sein muss, dass Statusinformationen (Metriken) einfach extrahiert oder dem Projektmanagement der TAS automatisch in Berichtsform übermittelt werden kann.

3.1.8 TAS-Unterstützung für das Testmanagement

Eine TAS muss das Testmanagement für das SUT unterstützen. Testberichte einschließlich Testprotokolle und Testergebnisse müssen einfach extrahiert werden können oder dem Testmanagement (Mensch oder System)

des SUT automatisch übermittelt werden.

3.2 TAA-Entwurf

3.2.1 Einführung in den TAA-Entwurf

Es gibt einige Hauptaktivitäten, die für den Entwurf einer TAA erforderlich sind. Sie lassen sich anhand der Erfordernisse des Testautomatisierungsprojekts oder Unternehmens in eine Reihenfolge setzen. Diese Aktivitäten werden in den folgenden Abschnitten besprochen. Je nach Komplexität der TAA sind mehr oder weniger Aktivitäten erforderlich.

Erfassung der Anforderungen, die für die Definition einer geeigneten TAA erforderlich sind
Bei den Anforderungen für einen Testautomatisierungsansatz muss Folgendes berücksichtigt werden:

- Welche Aktivität oder Phase des Testprozesses automatisiert werden soll: Testmanagement, Testentwurf, Testgenerierung oder Testausführung. Dabei ist zu beachten, dass die Testautomatisierung den fundamentalen Testprozess durch Einplanen der Testgenerierung zwischen Testentwurf und Testimplementierung verfeinert.
- Welche Teststufe unterstützt werden soll: Komponententest, Integrationstest, Systemtest
- Welche Art von Test unterstützt werden soll: funktionale Tests, Konformitätstests, Interoperabilitätstests
- Welche Testrolle unterstützt werden soll: Testausführender, Testanalyst, Testarchitekt, Testmanager
- Welches Softwareprodukt, welche Softwareproduktreihe oder Softwareproduktfamilie unterstützt werden soll, z. B. um die Spanne und Lebensdauer der implementierten TAS zu definieren
- Welche SUT-Technologien unterstützt werden sollen, z. B. um die TAS in Hinblick auf die Kompatibilität mit SUT-Technologien zu definieren

Vergleich und Gegenüberstellung verschiedener Entwurfs-/Architekturansätze
Der TAE muss beim Entwurf der ausgewählten Schichten der TAA die Pros und Kontras der verschiedenen Ansätze analysieren. Dazu gehören u. a.:

Überlegungen im Hinblick auf die Testgenerierungsschicht:

- Entscheidung für die manuelle oder automatisierte Testgenerierung
- Entscheidung für z. B. eine anforderungsbasierte, datenbasierte, szenariobasierte oder verhaltensbasierte Testgenerierung
- Auswahl von Testgenerierungsstrategien (z. B. Modellüberdeckung wie Klassifikationsbäume für datenbasierte Ansätze, Anwendungsfall-/Ausnahmefall-Überdeckung bei szenariobasierten Ansätzen, Übergangs-/Zustands-/Pfadüberdeckung bei verhaltensbasierten Ansätzen usw.)
- Wahl der Testauswahlstrategie. In der Praxis ist die uneingeschränkte kombinatorische Testgenerierung unmöglich, weil sie zur explosionsartigen Zunahme der Testfälle führen könnte. Daher sollten praktische Überdeckungskriterien, Gewichtungen, Risikobewertungen usw. als Orientierung bei der Testgenerierung und anschließenden Testauswahl dienen.

Überlegungen im Hinblick auf die Testdefinitionsschicht:

- Entscheidung für eine datengetriebene, schlüsselwortgetriebene, musterbasierte oder modellgetriebene Testdefinition
- Auswahl der Notation für die Testdefinition (z. B. Tabellen, zustandsbasierte Notation, stochastische Notation, Datenfluss-Notation, Geschäftsprozess-Notation, szenariobasierte Notation usw. durch Verwendung von Kalkulationstabellen, domänenspezifischer Testsprachen, der TTCN-3 (Testing and Test Control Notation), des UTP (UML Testing Profile) usw.)
- Auswahl der Styleguides und Richtlinien für die Definition hochwertiger Tests

- Auswahl von Testfall-Repositories (Kalkulationstabellen, Datenbanken, Dateien usw.)

Überlegungen im Hinblick auf die Testausführungsschicht:

- Auswahl des Testausführungswerkzeugs
- Auswahl eines Interpretations- (durch Verwendung einer virtuellen Maschine) oder Kompilierungsansatzes zur Implementierung von Testabläufen – hängt in der Regel vom gewählten Testausführungswerkzeug ab
- Auswahl der Implementierungstechnologie für die Implementierung von Testabläufen (imperativ, z. B. C; funktional, z. B. Haskell oder Erlang; objektorientiert, z. B. C++, C#, Java; Skripterstellung, z. B. Python oder Ruby, oder eine werkzeugspezifische Technologie) – hängt in der Regel vom gewählten Testausführungswerkzeug ab
- Auswahl von Hilfsbibliotheken zur Vereinfachung der Testausführung (z. B. Testgeräte-Bibliotheken, Kodierungs-/Dekodierungsbibliotheken usw.)

Überlegungen im Hinblick auf die Testadaptierungsschicht:

- Auswahl der Testschnittstellen zum SUT
- Auswahl der Werkzeuge zur Stimulierung und Beobachtung der Testschnittstellen
- Auswahl der Werkzeuge zur Überwachung des SUT während der Testausführung
- Auswahl der Werkzeuge zur Rückverfolgung der Testausführung (z. B. einschließlich Timing der Testausführung)

Ermitteln von Bereichen, in denen Abstraktion Vorteile bieten kann
Abstraktion in eine TAA ermöglicht die Unabhängigkeit von Technologie, weil dieselbe Testsuite in verschiedenen Testumgebungen oder für verschiedene Zieltechnologien genutzt werden kann. Dadurch erhöht sich die Portabilität der Testartefakte. Außerdem ist die Anbieterneutralität gewährleistet. Das verhindert die Bindung an einen Anbieter für eine TAS. Abstraktion verbessert auch die Wartbarkeit und Adaptierbarkeit an neue oder sich weiterentwickelnde SUT-Technologien. Zudem hilft Abstraktion, eine TAA und ihre Instanziierungen durch TASs zugänglicher für Nicht-Techniker zu machen. Die Testsuites werden hierbei auf einer abstrakteren Ebene, auch mit grafischen Mitteln, dokumentiert und erläutert, was die Lesbarkeit und Verständlichkeit verbessert.

Der TAE muss mit den an der Softwareentwicklung, Qualitätssicherung und Testdurchführung beteiligten Personen klären, mit welchem Abstraktionsgrad in welchem Bereich der TAS gearbeitet werden soll. Ein Beispiel: Welche Schnittstellen der Testadaptierungs- und/oder -ausführungsschicht müssen über die gesamte Lebensdauer der TAA hinweg externalisiert, formal definiert und stabil gehalten werden? Zudem muss besprochen werden, ob eine abstrakte Testdefinition verwendet wird oder die TAA nur eine Testausführungsschicht mit Testskripten nutzt. Analog dazu muss man wissen, ob die Testgenerierung durch Verwendung von Testmodellen und modellbasierten Testansätzen abstrahiert wird. Der TAE muss sich darüber im Klaren sein, dass es Kompromisse zwischen komplexen und einfachen Implementierungen einer TAA gibt, was den Gesamtfunktionsumfang, die Wartbarkeit und die Erweiterbarkeit angeht. Bei der Entscheidung darüber, welche Abstraktion in einer TAA genutzt werden soll, müssen diese Kompromisse berücksichtigt werden.

Je mehr Abstraktion für eine TAA genutzt wird, desto flexibler ist sie im Hinblick auf die weitere Entwicklung oder Überführung in neue Ansätze oder Technologien. Das erfordert eine größere Anfangsinvestition (z. B. komplexere Testautomatisierungsarchitektur und -werkzeuge, höhere Kompetenzanforderungen, steilere Lernkurven), was die Amortisierung verzögert, sich aber langfristig bezahlt machen kann. Zudem kann es die Leistung der TAS beeinträchtigen.

Gründliche Überlegungen zum ROI (Return on Investment) sind Aufgabe des TAM. Der TAE muss Informationen für die ROI-Analyse liefern, indem er technische Evaluierungen und Vergleiche verschiedener Testautomatisierungsarchitekturen und -ansätze im Hinblick auf Zeitrahmen, Kosten, Aufwand und Vorteile beisteuert.

Verstehen von SUT-Technologien und deren Vernetzung mit der TAS
Der Zugriff auf die Testschnittstellen des SUT ist entscheidend für jede automatisierte Testausführung. Der Zugriff

kann über folgende Ebenen gegeben sein:

- Softwareebene, z. B. SUT und Testsoftware sind miteinander verknüpft
- API-Ebene, z. B. : die TAS ruft die Funktionen/ Operationen/ Methoden auf, die an einer (entfernten) API bereitgestellt werden
- Protokollebene, z. B. : die TAS interagiert über HTTP, TCP usw. mit dem SUT
- Dienstebene, z. B. : die TAS interagiert mit den SUT-Diensten über Webdienste, RESTful-Dienste usw.

Darüber hinaus muss der TAE entscheiden, welches TAA-Interaktionsparadigma für die Interaktion zwischen TAS und SUT verwendet werden soll, wenn TAS und SUT durch APIs, Protokolle oder Dienste getrennt sind. Dabei kann es sich um folgende Paradigmen handeln:

- Ereignisgetriebenes Paradigma, das die Interaktion über Ereignisse vorantreibt, die über einen Ereignisbus ausgetauscht werden
- Client-Server-Paradigma, das die Interaktion über den Aufruf von Diensten von Service-Requestern an den Service-Provider vorantreibt
- Peer-to-Peer-Paradigma, das die Interaktion über den Aufruf von Diensten von einem Peer vorantreibt

Die Wahl des Paradigmas hängt oft von der SUT-Architektur ab und kann wiederum Auswirkungen auf die SUT-Architektur haben. Die Vernetzung zwischen SUT und TAA muss sorgfältig analysiert und entworfen werden, damit eine zukunftssichere Architektur zwischen den beiden Systemen gewählt wird.

Verstehen der SUT-Umgebung
Ein SUT kann Folgendes sein: eigenständige Software oder Software, die im Verbund mit anderer Software arbeitet (z. B. System von Systemen), Hardware (z. B. eingebettete Systeme) oder Umgebungskomponenten (z. B. cyber-physische Systeme). Eine TAS simuliert oder emuliert die SUT-Umgebung im Rahmen eines automatisierten Testaufbaus.

Hier einige Beispiele für Testumgebungen und Verwendungsbeispiele:

- Ein Computer mit dem SUT und der TAS – hilfreich für das Testen einer Softwareanwendung
- Einzelne über Netzwerk miteinander verbundene Computer jeweils für das SUT und die TAS – hilfreich für das Testen von Serversoftware
- Zusätzliche Testgeräte zur Simulation und Beobachtung der technischen Schnittstellen eines SUT – hilfreich für das Testen der Software auf z. B. einer Set-Top-Box
- Über Netzwerk miteinander verbundene Testgeräte zur Emulation der Betriebsumgebung des SUT – hilfreich für das Testen der Software auf einem Netzwerkrouter
- Simulatoren zur Simulation der physischen Umgebung des SUT – hilfreich für das Testen der Software eines eingebetteten Steuergeräts

Zeitaufwand und Komplexität der Implementierung einer Testautomatisierungsarchitektur
Die Abschätzung des Aufwands für ein TAS-Projekt ist Aufgabe des TAM. Der TAE muss den TAM dabei unterstützen, indem er gute Schätzwerte für den Zeitaufwand und die Komplexität eines TAA-Entwurfs liefert. Für das Schätzen gibt es folgende Methoden und Beispiele:

- analogiebasiertes Schätzen wie Funktionspunkte, Dreipunktschätzung, Breitband-Delphi und Expertenschätzung
- Schätzung durch Rückgriff auf Projektstrukturplänen wie sie Managementsoftware oder Projektvorlagen bieten
- parametrische Schätzung wie das Constructive Cost Model (COCOMO)
- größenbasierte Schätzungen wie die Funktionspunktanalyse, Story-Point-Analyse oder Anwendungsfallanalyse
- Gruppenschätzungen wie Planungspoker

Einfachheit der Verwendung der Implementierung einer Testautomatisierungsarchitektur
Neben dem Funktionsumfang der TAS, ihrer Kompatibilität mit dem SUT, ihrer langfristigen Stabilität und Möglichkeit zur Weiterentwicklung, ihrem erforderlichen Aufwand und den ROI-Überlegungen muss der TAE speziell auch den Benutzbarkeitsfragen einer TAS Rechnung tragen. Das schließt u. a. Folgendes ein:

- Tester-orientierter Entwurf
- Einfachheit der Verwendung der TAS
- TAS-Unterstützung für andere Rollen in der Softwareentwicklung, Qualitätssicherung und im Projektmanagement
- effektive Organisation, Navigation und Suche in der TAS
- hilfreiche Dokumentation, Handbücher und Hilfetext für die TAS
- praktische Berichterstattung durch und über die TAS
- iterative Entwürfe, um Feedback zur TAS und empirischen Erkenntnissen Rechnung zu tragen

3.2.2 Ansätze zur Automatisierung von Testfällen

Testfälle müssen in Handlungsabfolgen umgesetzt werden, die an einem SUT ausgeführt werden. Diese Handlungsabfolge kann in einem Testablauf dokumentiert und/oder in einem Testskript implementiert werden. Neben den Handlungen/Aktionen müssen die automatisierten Testfälle auch Testdaten für die Interaktion mit dem SUT definieren und Verifizierungsschritte umfassen, mit denen sich prüfen lässt, ob vom SUT das erwartete Ergebnis erzielt wurde. Für die Erstellung der Handlungsabfolge kann eine Reihe von Ansätzen genutzt werden:

1. Der TAE implementiert Testfälle direkt in automatisierten Testskripts. Dazu wird am wenigsten geraten, weil es an Abstraktion mangelt und ein hoher Wartungsaufwand besteht.
2. Der TAE entwirft Testabläufe und wandelt diese in automatisierte Testskripte um. Diese Option verfügt über Abstraktion, es mangelt ihr aber an Automatisierung zur Erzeugung der Testskripts.
3. Der TAE nutzt ein Werkzeug zur Umsetzung der Testabläufe in automatisierte Testskripts. Diese Option vereint Abstraktion und automatisierte Skripterzeugung.
4. Der TAE nutzt ein Werkzeug, das automatisierte Testabläufe generiert und/oder die Testskripte direkt aus den Modellen ableitet. Diese Option hat den höchsten Automatisierungsgrad.

Beachten Sie, dass die Optionen stark vom Kontext des Projekts abhängen. Effizient kann es auch sein, die Testautomatisierung durch Anwendung einer der weniger komplexen Optionen zu starten, weil diese im Regelfall einfacher zu implementieren sind. Das kann kurzfristig Mehrwert bieten, beinträchtig allerdings die Wartbarkeit der Lösung.

Zu den gut etablierten Ansätzen für die Automatisierung von Testfällen zählen:

- Der Mitschnitt-Ansatz (Capture/Playback), der für Option 1 genutzt werden kann.
- Die strukturierte Skripterstellung und der schlüsselwortgetriebene Ansatz, die für Option 2 oder 3 genutzt werden können.
- Das modellbasierte Testen (einschließlich des prozessgetriebenen Ansatzes), das für Option 4 verwendet werden kann.

Die wichtigsten Konzepte sowie die Pros und Kontras dieser Ansätze werden nachstehend erläutert.

Mitschnitt-Ansatz (Capture/Replay)

Grundkonzept

Bei Mitschnitt-Ansätzen erfassen Werkzeuge die Interaktionen mit dem SUT während der Ausführung der durch einen Testablauf vorgegebenen Handlungsabfolge. Dabei werden die Eingaben und ggf. auch die Ausgaben für spätere Prüfungen aufgezeichnet. Während der Wiedergabe der Ereignisse (Replay) gibt es verschiedene Möglichkeiten der manuellen und automatisierten Prüfung der Ausgaben:

- Manuell: Der Tester muss die SUT-Ausgaben auf Anomalien beobachten.
- Vollständig: Alle Systemausgaben, die bei der Erfassung aufgezeichnet wurden, müssen vom SUT reproduziert werden.
- Exakt: Alle Systemausgaben, die bei der Erfassung aufgezeichnet wurden, müssen vom SUT bis auf die Detailebene der Aufzeichnung reproduziert werden.
- Checkpoints: Es werden nur ausgewählte Systemausgaben an bestimmten Punkten für festgelegte Werte geprüft.

Pros

Der Mitschnitt-Ansatz kann für SUTs auf GUI- und/oder API-Ebene genutzt werden. Er ist anfänglich einfach einzurichten und zu verwenden.

Kontras

Mitschnitt-Skripte sind schwierig zu warten und weiterzuentwickeln, weil die mitgeschnittene SUT-Ausführung stark von der SUT-Version abhängt, von der der Mitschnitt angefertigt wurde. Ein Beispiel: Beim Aufzeichnen auf GUI-Ebene können Änderungen im GUI-Layout Auswirkungen auf das Testskript haben, auch wenn es sich lediglich um eine Änderung in der Positionierung eines GUI-Elements handelt. Das macht Mitschnitt-Ansätze anfällig für Änderungen.

Die Implementierung der Testfälle (Skripts) kann erst beginnen, wenn das SUT verfügbar ist.

Lineare Skripterstellung

Grundkonzept

Wie bei allen Techniken der Skripterstellung bilden auch bei der linearen Skripterstellung einige manuelle Testabläufe den Anfang. Hierbei muss es sich nicht um schriftliche Dokumente handeln. Das Wissen darüber, welche Tests wie ausgeführt werden sollen, sollte aber einem oder mehreren Testanalysten bekannt sein.

Jeder Test wird manuell ausgeführt. Das Testwerkzeug zeichnet dabei die Abfolge der Aktionen und in einigen Fällen die sichtbare Ausgabe des SUT auf dem Bildschirm auf. Das mündet in der Regel in einem (meist umfangreichen) Skript für jeden Testablauf. Aufgezeichnete Skripte können bearbeitet werden, um die Lesbarkeit zu verbessern (z. B. durch Hinzufügen von Kommentaren, um zu erläutern, was an den wichtigen Punkten passiert). Ferner können unter Verwendung der Skriptsprache des Werkzeugs weitere Prüfungen hinzugefügt werden.

Die Skripte können dann mit dem Werkzeug wiedergegeben werden (Replay). Dadurch wiederholt das Werkzeug genau die Aktionen, die der Tester beim Aufzeichnen der Skripte durchgeführt hat. Damit lassen sich zwar GUI-Tests automatisieren, die Technik bietet sich jedoch nicht an, wenn eine große Anzahl von Tests automatisiert werden soll, die für viele Releases der Software verwendet werden müssen. Das liegt am hohen Wartungsaufwand, der bei Änderungen am SUT entsteht. Jede Änderung am SUT kann viele Änderungen in den aufgezeichneten Skripten erforderlich machen.

Pros

Die Vorteile linearer Skripte sind im Umstand zu sehen, dass es keiner oder nur einer geringen Vorbereitung bedarf, bevor Sie mit dem Automatisieren beginnen können. Sobald Sie sich mit der Verwendung des Werkzeugs auskennen, ist es einfach eine Frage des Aufzeichnens eines manuellen Tests und dessen anschließender Wiedergabe (auch wenn im Aufzeichnungsteil zusätzliche Interaktionen mit dem Testwerkzeug erforderlich sein können, um Vergleiche von Ist-Ausgaben mit erwarteten Ausgaben anzufordern, um sich davon zu überzeugen, dass die Software wie vorgesehen funktioniert). Programmierkenntnisse sind dafür nicht erforderlich, aber in der Regel hilfreich.

Kontras

Lineare Skripte haben eine Vielzahl von Nachteilen. Der für die Automatisierung eines Testablaufs nötige Aufwand hängt meist von seinem Umfang (Anzahl der Schritte oder Aktionen) ab. Das heißt: Der

tausendste zu automatisierende Testablauf erfordert einen vergleichbaren anteiligen Aufwand wie der einhundertste Testablauf. Mit anderen Worten: Der Spielraum für die Reduzierung des Aufwands für die Entwicklung neuer automatisierter Tests ist klein.

Zudem gilt: Wenn es ein zweites Skript gibt, das einen ähnlichen Test, allerdings mit anderen Eingabewerten ausführt, enthält dieses Skript dieselbe Abfolge von Anweisungen wie das erste Skript; lediglich die in den Anweisungen enthalten Informationen (die Anweisungsargumente oder -parameter) sind andere. Gibt es mehrere Tests (und damit auch Skripts), enthalten diese alle dieselbe Abfolge von Anweisungen, die alle gewartet werden müssen, wenn sich Änderungen an der Software ergeben, die Einfluss auf die Skripte haben.

Weil die Skripte nicht in natürlicher Sprache, sondern in Programmiersprache vorliegen, sind sie für Nicht-Programmierer u. U. schwer zu verstehen. Einige Testwerkzeuge nutzen proprietäre (werkzeugspezifische) Sprachen. Dann kostet es Zeit, die jeweilige Sprache zu lernen und versiert einzusetzen.

Aufgezeichnete Skripte enthalten – wenn überhaupt – nur allgemeine Aussagen in den Kommentaren. Vor allem lange Skripte sollten mit Kommentaren versehen werden, in denen erläutert wird, was in jedem Schritt des Tests passiert. Das vereinfacht die Wartung. Skripte können schnell sehr umfangreich werden (d. h., viele Anweisungen enthalten), wenn der Test viele Schritte umfasst.

Die Skripte sind nicht modular und schwer zu warten. Die lineare Skripterstellung genügt den allgemeinen Grundsätzen der Wiederverwendbarkeit und Modularität von Software nicht, und die Skripte sind stark an das verwendete Werkzeug gebunden.

Strukturierte Skripterstellung

Grundkonzept

Der größte Unterschied zwischen der strukturierten und der linearen Skripterstellung ist die Skriptbibliothek, die bei der strukturierten Technik hinzukommt. Sie enthält wiederverwendbare Skripte, die Anweisungsabfolgen ausführen, wie sie bei einer Vielzahl von Tests in der Regel benötigt werden. Beispiele für Skripte dieser Art sind Skripte mit einer Schnittstelle, z. B. zu den Vorgängen auf SUT-Oberflächen.

Pros

Die Vorteile des Ansatzes liegen in der erheblichen Reduzierung der nötigen Wartung bei Änderungen und des Aufwands für die Automatisierung neuer Tests (weil bereits existierende Skripte genutzt werden können und so nicht alle Skripte neu erstellt werden müssen).

Die Vorteile der strukturierten Skripterstellung ergeben sich zum großen Teil aus der Wiederverwendung von Skripten. Mehr Tests können automatisiert werden, ohne dass dazu eine ähnlich große Menge an Skripten wie beim linearen Ansatz erstellt werden muss. Das wirkt sich direkt auf die Entwicklungs- und Wartungskosten aus. Der zweite und die nachfolgenden Tests erfordern weniger Automatisierungsaufwand, weil einige der für die Implementierung des ersten Tests entwickelten Skripte erneut verwendet werden können.

Kontras

Der Anfangsaufwand für die Erstellung der gemeinsam genutzten Skripte kann als Nachteil gesehen werden, bei der richtigen Vorgehensweise macht er sich jedoch mehr als bezahlt. Für die Erstellung aller Skripte können Programmierkenntnisse erforderlich sein, weil ein einfaches Aufzeichnen nicht reicht. Die Skriptbibliothek muss gut verwaltet werden, d. h., die Skripte müssen dokumentiert werden und die benötigten Skripte müssen für technische Testanalysten einfach zu finden sein (eine logische Benennungskonvention hilft dabei).

Datengetriebenes Testen

Grundkonzept

Die datengetriebene Skripterstellung baut auf der strukturierten Skripterstellung auf. Der wichtigste Unterschied ist die Handhabung der Testeingaben. Diese Eingaben werden aus den Skripten extrahiert und in einer oder mehreren Dateien abgelegt (die in der Regel „Datendateien“ heißen).

Das heißt, dass das Haupttestskript für die Implementierung weiterer Tests wiederverwendet werden kann (statt nur für einen Test). Das ‚wiederverwendbare‘ Haupttestskript wird in der Regel als ‚Steuerungsskript‘ bezeichnet. Das Steuerungsskript enthält die Abfolge der Anweisungen für die Durchführung der Tests, liest die Eingabedaten aber aus einer Datendatei. Ein Steuerungsskript kann für viele Tests verwendet werden, reicht aber in der Regel nicht, um eine Vielzahl von Tests zu automatisieren. Daher werden mehrere Steuerungsskripte benötigt. Sie bilden jedoch nur einen Bruchteil der Anzahl der automatisierten Tests.

Pros

Der Aufwand für das Hinzufügen neuer automatisierter Tests lässt sich mit dieser Technik der Skripterstellung erheblich reduzieren. Sie dient der Automatisierung vieler Variationen eines nützlichen Tests, ermöglicht so ein tieferes Testen in einem spezifischen Bereich und kann die Testüberdeckung erhöhen.

Weil die Tests von den Datendateien ‚beschrieben‘ werden, können Testanalysten ‚automatisierte‘ Tests spezifizieren, indem sie einfach eine oder mehrere Datendateien mit Daten füllen. Das gibt Testanalysten eine größere Freiheit bei der Spezifizierung von automatisierten Tests bei geringerer Abhängigkeit von technischen Testanalysten (die ein knappes Gut sein können).

Kontras

Die Notwendigkeit, Datendateien zu verwalten und sicherzustellen, dass diese von der TAS lesbar sind, ist ein Nachteil, der sich aber handhaben lässt.

Auch fehlen möglicherweise wichtige negative Testfälle. Negative Tests sind eine Kombination aus Testabläufen und Testdaten. Bei einem Ansatz, der vorrangig auf Testdaten abzielt, können „negative Testabläufe“ fehlen.

Schlüsselwortgetriebenes Testen

Grundkonzept

Die schlüsselwortgetriebene Skripterstellung baut auf der datengetriebenen Skripterstellung auf. Es gibt zwei entscheidende Unterschiede: (1) die Datendateien heißen hier ‚Testdefinitionsdateien‘ oder ähnlich (z. B. Aktionswortdateien); und (2) es gibt nur ein Steuerungsskript.

Eine Testdefinitionsdatei enthält eine Beschreibung der Tests, die für Testanalysten leichter verständlich ist (einfacher als die äquivalente Datendatei). In der Regel enthält sie genau wie die Datendateien Daten, aber Schlüsselwortdateien enthalten darüber hinaus abstrakte Anweisungen (die Schlüssel- oder Aktionswörter).

Die Schlüsselwörter müssen so gewählt werden, dass sie für den Testanalysten, die zu beschreibenden Tests und die zu testende Anwendung sinnvoll sind. Sie dienen meist, allerdings nicht ausschließlich, der Abbildung abstrakter Interaktionen mit einem System (z. B. „Bestellung aufgeben“). Jedes Schlüsselwort repräsentiert eine Reihe detaillierter Interaktionen mit dem zu testenden System. Abfolgen von Schlüsselwörtern (einschließlich der relevanten Testdaten) dienen der Spezifizierung von Testfällen. Spezielle Schlüsselwörter können für die verifizierenden Schritte verwendet werden oder sowohl die Aktionen als auch die Verifizierungsschritte enthalten.

In den Aufgabenbereich des Testanalysten fällt das Erstellen und Warten der Schlüsselwortdateien. Das heißt: Sobald die unterstützenden Skripte implementiert wurden, können Testanalysten ‚automatisierte‘ Tests hinzufügen, indem sie diese einfach in einer Schlüsselwortdatei spezifizieren (wie bei der datengetriebenen Skripterstellung).

Pros

Sobald das Steuerungsskript und die unterstützenden Skripte für die Schlüsselwörter geschrieben wurden, sinkt der Aufwand für das Hinzufügen neuer automatisierter Tests mittels dieser Technik der

Skripterstellung erheblich.

Weil die Tests von den Schlüsselwortdateien ‚beschrieben' werden, können Testanalysten ‚automatisierte' Tests spezifizieren, indem sie die Tests einfach mit Stichwörtern und den zugehörigen Daten beschreiben. Das gibt Testanalysten eine größere Freiheit bei der Spezifizierung von automatisierten Tests bei geringerer Abhängigkeit von technischen Testanalysten (die ein knappes Gut sein können). Der Vorteil des schlüsselwortgetriebenen Ansatzes gegenüber dem datengetriebenen Ansatz ist in dieser Hinsicht die Verwendung von Schlüsselwörtern. Jedes Schlüsselwort sollte eine Abfolge detaillierter Aktionen repräsentieren, die ein sinnvolles Ergebnis produzieren. Ein Beispiel: ‚Konto erstellen', ‚Bestellung aufgeben' und ‚Bestellungsstatus prüfen' sind mögliche Aktionen für eine Online-Shopping-Anwendung, die jeweils eine Reihe detaillierter Schritte einschließen. Wenn ein Testanalyst ein zu testendes System für einen anderen Testanalysten beschreibt, tauschen sie sich wahrscheinlich unter Bezugnahme auf diese abstrakten Aktionen und nicht auf Ebene der detaillierten Schritte aus. Ziel des schlüsselwortgetriebenen Ansatzes ist es dann, diese abstrakten Aktionen zu implementieren und das Definieren von Tests zu ermöglichen, die sich nicht auf die detaillierten Schritte, sondern auf die abstrakten Aktionen beziehen.

Diese Testfälle sind einfacher zu warten, zu lesen und zu schreiben, weil die Komplexität in den Schlüsselwörtern (oder bei der strukturierten Skripterstellung in den Bibliotheken) versteckt werden kann. Die Schlüsselwörter können eine Abstraktion der Komplexitäten der Schnittstellen des SUT bieten.

Kontras

Die Implementierung der Schlüsselwörter bleibt eine große Aufgabe für die TAEs, vor allem, wenn sie dafür ein Werkzeug nutzen, das keine Unterstützung für diese Technik der Skripterstellung bietet. Bei kleinen Systemen kann der Aufwand für die Implementierung zu groß sein, sodass die Nachteile die Vorteile überwiegen.

Es muss sorgfältig darauf geachtet werden, dass die richtigen Schlüsselwörter verwendet werden. Gute Schlüsselwörter werden häufig bei vielen verschiedenen Tests verwendet, während schlechte Schlüsselwörter nur einmal oder wenige Male verwendet werden.

Prozessgetriebene Skripterstellung

Grundkonzept

Der prozessgetriebene Ansatz baut auf der schlüsselwortgetriebenen Skripterstellung auf. Der Unterschied besteht darin, dass Szenarios – als Abbild der Anwendungsfälle des SUT und ihrer Varianten – die Skripte bilden. Diese werden mit Testdaten parametrisiert oder zu abstrakteren Testdefinitionen kombiniert.

Solche Testdefinitionen sind einfacher zu handhaben, weil sich die logische Beziehung zwischen Aktionen, z. B. ‚Bestellungsstatus prüfen' nach ‚Bestellung aufgeben' beim Testen von Features oder ‚Bestellungsstatus prüfen' ohne vorheriges ‚Bestellung aufgeben', ermitteln lässt.

Pros

Die Verwendung prozessartiger, szenariobasierter Definitionen von Testfällen ermöglicht die Definition von Testabläufen aus Workflow-Perspektive. Ziel des prozessgetriebenen Ansatzes ist die Implementierung dieser abstrakten Workflows unter Verwendung auf Testbibliotheken, die detaillierte Testschritte repräsentieren (siehe auch schlüsselwortgetriebener Ansatz).

Kontras

Die Prozesse eines SUT sind vom technischen Testanalysten u. U. nicht einfach zu verstehen – und das erschwert die Implementierung der prozessorientierten Skripts, vor allem, wenn vom Werkzeug keine Geschäftsprozesslogik unterstützt wird.

Es muss zudem sorgfältig darauf geachtet werden, dass die richtigen Prozesse unter Verwendung der richtigen Schlüsselwörter verwendet werden. Gute Prozesse werden von anderen Prozessen referenziert und münden in vielen relevanten Tests. Schlechte Prozesse hingegen haben keine Relevanz, eine schlechte Fehlererkennungsrate usw.

Modellbasiertes Testen

Grundkonzept

Das modellbasierte Testen bezeichnet die automatisierte Generierung von Testfällen (siehe auch den Lehrplan für den modellbasierten Tester vom ISTQB) – im Gegensatz zur automatisierten Ausführung von Testfällen – unter Verwendung von Mitschnitten sowie der linearen, strukturierten, datengetriebenen oder prozessgetriebenen Skripterstellung. Beim modellbasierten Testen werden (semi-)formale Modelle verwendet, die von den Skripterstellungstechniken der TAA abstrahieren. Verschiedene Methoden der Testgenerierung können für das Ableiten von Tests für die verschiedenen, eingangs erläuterten Skripterstellungs-Frameworks verwendet werden.

Pros

Das modellbasierte Testen ermöglicht durch Abstraktion die Konzentration auf die Quintessenz des Testens (in Bezug auf zu testende Geschäftslogik, Datenszenarien, Konfigurationen usw.). Zudem ermöglicht es das Generieren von Tests für verschiedene Zielsysteme und avisierte Technologien, so dass die für die Testgenerierung verwendeten Modelle eine zukunftssichere Repräsentation der Testmittel darstellen, die wiederverwendet und mit der sich weiterentwickelnden Technik entsprechend gewartet werden können.

Bei Änderung der Anforderungen muss nur das Testmodell angepasst werden; die Testfälle werden dann in Gänze automatisch erzeugt. Testfall-Entwurfstechniken sind Bestandteil der Testfall-Generatoren.

Kontras

Die wirksame Umsetzung eines modellbasierten Testansatzes setzt Modellierungskenntnisse voraus. Die Aufgabe der Modellierung durch Abstraktion der Schnittstellen eines SUT, von Daten und/oder Verhaltensmustern kann schwierig sein. Zudem sind Modellierungs- und modellbasierte Testwerkzeuge noch nicht im Mainstream angekommen, werden aber zunehmend ausgereifter. Modellbasierte Testansätze erfordern Anpassungen in den Testprozessen. So muss z. B. die Rolle des Testentwicklers etabliert werden. Darüber hinaus bilden die für die Testgenerierung verwendeten Modelle wichtige Artefakte für die Qualitätssicherung eines SUT und müssen selbst qualitätsgesichert und gewartet werden.

3.2.3 Technische Überlegungen zum SUT

Beim Entwurf einer TAA müssen auch die technischen Aspekte des jeweiligen SUT berücksichtigt werden. Einige dieser Aspekte werden nachstehend behandelt. Dies kann jedoch nicht erschöpfend erfolgen. Die behandelten Aspekte sollen daher als Beispiele für wichtige Aspekte dienen.

Schnittstellen des SUT

Ein SUT hat interne Schnittstellen (innerhalb des Systems) und externe Schnittstellen (zur Systemumgebung und ihren Benutzern oder durch exponierte Komponenten). Eine TAA muss in der Lage sein, all diese Schnittstellen des SUT, die von Testabläufen potentiell betroffen sind, zu steuern und/oder zu beobachten (d. h., Schnittstellen müssen testbar sein). Darüber hinaus kann es notwendig sein, die Interaktionen zwischen dem SUT und der TAS mit unterschiedlichen Detailgraden, in der Regel mit Zeitstempeln, aufzuzeichnen.

Testfokussierung (z. B. ein bestimmter Test) ist am Anfang des Projekts (oder fortwährend in agilen Umgebungen) während der Definition der Architektur notwendig, um die Verfügbarkeit der benötigten Testschnittstellen oder Testeinrichtungen zu verifizieren, die benötigt werden, damit das SUT testbar ist (Auslegung auf Testbarkeit).

SUT-Daten

Ein SUT nutzt Konfigurationsdaten, um seine Instanziierung, Konfiguration, Administration usw. zu steuern. Zudem verwendet es Benutzerdaten, die es verarbeitet. Ein SUT kann für die Ausführung seiner Aufgaben zudem externe Daten von anderen Systemen nutzen. In Abhängigkeit von den Testabläufen für ein SUT müssen all diese Datentypen definierbar, konfigurierbar und von der TAA instanziierbar sein. Wie mit den SUT-Daten konkret verfahren wird, legt man beim Entwurf der TAA fest. Je nach Ansatz können Daten als Parameter, Testdatenblätter, Testdatenbanken, reale Daten usw. gehandhabt werden.

SUT-Konfigurationen
Ein SUT kann in verschiedenen Konfigurationen bereitgestellt werden: z. B. auf verschiedenen Betriebssystemen, auf verschiedenen Zielgeräten oder mit unterschiedlichen Spracheinstellungen. Je nach den Testabläufen kann es sein, dass die TAA unterschiedlichen SUT-Konfigurationen Rechnung tragen muss. Die Testabläufe können verschiedene Testumgebungen (in einem Labor) oder virtuelle Testumgebungen (in der Cloud) der TAA in Kombination mit einer gegebenen SUT-Konfiguration erfordern. Das kann auch die Aufnahme von Simulatoren und/oder Emulatoren ausgewählter SUT-Komponenten für ausgewählte SUT-Aspekte erfordern.

SUT-Standards und rechtliche Vorgaben
Neben den technischen Aspekten eines SUT müssen beim TAA-Entwurf u. U. rechtliche und/oder standardbezogene Vorgaben berücksichtigt werden, um die Konformität der TAA zu garantieren. Dazu zählen z. B. Datenschutzvorgaben für die Testdaten oder Vertraulichkeitsvorgaben, die sich auf die Protokollierungs- und Berichtsfunktionen der TAA auswirken.

Werkzeuge und Werkzeugumgebungen für die Entwicklung des SUT
Bei der SUT-Entwicklung können verschiedene Werkzeuge für das Definieren von Anforderungen, Entwurf und Modellierung, Kodierung sowie die Integration und Verteilung des SUT verwendet werden. Die TAA mit ihren eigenen Werkzeugen muss der SUT-Werkzeuglandschaft Rechnung tragen, um die Kompatibilität, Rückverfolgbarkeit und/oder Wiederverwendung von Artefakten zu gewährleisten.

Testschnittstellen im Softwareprodukt
Es wird dringend empfohlen, vor der Freigabe des Produkts nicht alle Testschnittstellen zu entfernen. In den meisten Fällen verbleiben diese Schnittstellen im SUT, ohne beim Endprodukt Probleme zu bereiten. Werden die Schnittstellen belassen, können sie von Service- und Support-Technikern für die Problemdiagnose sowie das Testen von Wartungs-Releases genutzt werden. Es ist allerdings wichtig, sich davon zu überzeugen, dass die Schnittstellen keine Sicherheitsrisiken darstellen. Gegebenenfalls können Entwickler diese Testschnittstellen deaktivieren, damit sie außerhalb der Entwicklungsabteilung nicht genutzt werden können.

3.2.4 Überlegungen zu Entwicklungs-/QA-Prozessen

Beim Entwurf einer TAA müssen die Aspekte der Entwicklungs- und QA-Prozesse eines SUT berücksichtigt werden. Einige dieser Aspekte werden nachstehend behandelt. Dies kann jedoch nicht erschöpfend erfolgen. Die behandelten Aspekte sollen daher als Beispiele für wichtige Aspekte dienen.

Anforderungen an die Steuerung der Testausführung
Je nach Automatisierungsgrad, den die TAA erfordert, kann die TAA die interaktive Testausführung, die Testausführung im Batch-Modus oder die vollautomatisierte Testausführung unterstützen.

Anforderungen an die Berichterstattung
Je nach den Anforderungen an die Berichterstattung einschließlich der Berichtstypen und ihres Aufbaus muss die TAA feste, parametrisierte oder definierte Testberichte in unterschiedlichen Formaten und Layouts unterstützen können.

Rollen und Zugriffsrechte
Je nach den Sicherheitsanforderungen muss die TAA u. U. ein Rollen- und Zugriffsrechtesystem bieten.

Etablierte Werkzeuglandschaft
SUT-Projektmanagement, Testmanagement, Code- und Test-Repository, Fehlerverfolgung, Fehler- und Abweichungsmanagement, Risikoanalyse usw. können von Werkzeugen unterstützt werden, die zusammen die etablierte Werkzeuglandschaft bilden. Die TAA wird zudem von einem Werkzeug oder Werkzeugsatz unterstützt, das/der sich nahtlos mit den anderen Werkzeugen der Landschaft integrieren lassen muss. Zudem sollten Testskripte wie SUT-Code gespeichert und versioniert werden, damit Revisionen bei beiden nach demselben Prozess erfolgen.

3.3 TAS-Entwicklung

3.3.1 Einführung in die TAS-Entwicklung

Die Entwicklung einer TAS ist mit anderen Softwareentwicklungsprojekten vergleichbar. Sie kann nach denselben Abläufen und Prozessen erfolgen, einschließlich Peer Reviews durch Entwickler und Tester. Spezifisch für eine TAS sind ihre Kompatibilität und Synchronisierung mit dem SUT. Das muss beim Entwurf der TAA (siehe Abschnitt 3.2) und bei der Entwicklung der TAS berücksichtigt werden. Zudem wird das SUT von der Teststrategie beeinflusst, d. h., es ist dafür zu sorgen, dass Testschnittstellen für die TAS verfügbar sind.

In diesem Abschnitt werden anhand des Softwareentwicklungslebenszyklus (SDLC) der TAS-Entwicklungsprozess und die prozessbezogenen Aspekte der Kompatibilität und Synchronisierung mit dem SUT erläutert. Diese Aspekte sind genauso wichtig für jeden anderen Entwicklungsprozess, der für die SUT- und/oder TAS-Entwicklung gewählt wurde oder verwendet wird – sie müssen entsprechend adaptiert werden.

Der grundlegende SDLC für die TAS wird in Abbildung 2 veranschaulicht.

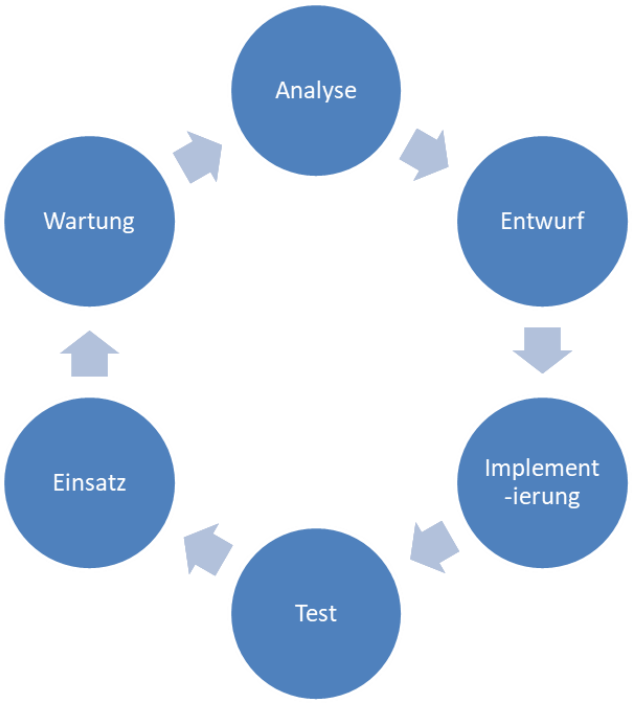

Abbildung 2: Grundlegender SDLC für die TAS

Die Anforderungen für eine TAS müssen analysiert und gesammelt werden (siehe Abb. 2). Die Anforderungen bilden den Leitfaden beim Entwurf der TAS anhand der Definition ihrer TAA (siehe Abschnitt 3.2). Aus dem Entwurf wird durch Umsetzung von Softwareentwicklungsansätzen Software. Beachten Sie, dass eine TAS auch eine dedizierte Testgerätehardware nutzen kann, die in diesem Lehrplan nicht behandelt wird. Wie jede andere Software muss eine TAS getestet werden. In der Regel erfolgt dies im Rahmen grundlegender Fähigkeitstests für die TAS, gefolgt von einem Zusammenspiel zwischen TAS und SUT. Nach Verteilung und Einsatz einer TAS muss die TAS häufig weiterentwickelt werden, um weitere Testfunktionen hinzuzufügen, Tests zu ändern oder die TAS zu aktualisieren, um dem sich ändernden SUT Rechnung zu tragen. Die Weiterentwicklung der TAS erfordert eine neue Runde der TAS-Entwicklung gemäß dem SDLC.

Beachten Sie auch, dass im SDLC Backup, Archivierung und Außerbetriebnahme einer TAS keine Berücksichtigung finden. Wie bei der TAS-Entwicklung müssen diese Prozesse den etablierten Methoden eines Unternehmens folgen.

3.3.2 Kompatibilität zwischen TAS und SUT

Prozesskompatibilität
Das Testen eines SUT muss synchron zu seiner Entwicklung erfolgen – und bei Automatisierung der Tests auch synchron zur TAS-Entwicklung. Daher ist es von Vorteil, die Prozesse für die SUT-Entwicklung, die TAS-Entwicklung und für das Testen zu koordinieren. Äußerst hilfreich ist es, wenn die SUT- und TAS-Entwicklung hinsichtlich Prozessstruktur, Prozessmanagement und Werkzeugunterstützung kompatibel sind.

Team-Kompatibilität
Die Team-Kompatibilität ist ein weiterer Aspekt der Kompatibilität zwischen TAS- und SUT-Entwicklung. Wenn die TAS- und SUT-Entwicklung mit denselben Denkmustern angegangen und gesteuert wird, profitieren beide Teams, weil sie die Anforderungen, Entwürfe und/oder Entwicklungsartefakte des jeweils anderen überprüfen, Probleme besprechen und kompatible Lösungen finden können. Team-Kompatibilität erleichtert auch die Kommunikation und Interaktion miteinander.

Technologie-Kompatibilität
Auch die technologische Kompatibilität zwischen TAS und SUT sollte Berücksichtigung finden. Es ist von Vorteil, von Anfang an ein nahtloses Zusammenspiel zwischen TAS und SUT anzustreben und umzusetzen. Selbst wenn das nicht möglich ist (z. B. weil technische Lösungen für die TAS oder das SUT nicht verfügbar sind), ist es u. U. mit Hilfe von Adaptern, Wrappern oder anderen intermediären Kommunikationswegen zu bewerkstelligen.

Werkzeugkompatibilität
Die Werkzeugkompatibilität zwischen TAS- und SUT-Management, -Entwicklung und -Qualitätssicherung muss berücksichtigt werden. Ein Beispiel: Wenn dieselben Tools für das Anforderungsmanagement und/oder das Problemmanagement genutzt werden, vereinfacht das den Austausch von Informationen sowie die Koordination von TAS- und SUT-Entwicklung.

3.3.3 Synchronisierung zwischen TAS und SUT

Synchronisierung von Anforderungen
Nach der Erhebung von Anforderungen sind die SUT- und TAS-Anforderungen zu entwickeln. TAS-Anforderungen lassen sich in zwei Hauptgruppen unterteilen: (1) Anforderungen in Bezug auf die Entwicklung der TAS als softwarebasiertes System, z. B. Anforderungen für die TAS-Features für den Entwurf, die Spezifikation und die Ergebnisanalyse von Tests usw., und (2) Anforderungen in Bezug auf das Testen des SUT mittels der TAS. Diese so genannten Testanforderungen entsprechen den SUT-Anforderungen und spiegeln alle SUT-Features und -Eigenschaften wider, die von der TAS zu testen sind. Wenn die SUT- oder TAS-Anforderungen aktualisiert werden, muss die Konsistenz zwischen beiden geprüft werden. Außerdem ist zu prüfen, ob es für alle SUT-Anforderungen, die von der TAS zu testen sind, definierte Testanforderungen gibt.

Synchronisierung der Entwicklungsphasen
Damit die TAS einsatzbereit ist, wenn sie für das Testen des SUT gebraucht wird, müssen die Entwicklungsphasen koordiniert werden. Am effizientesten ist es, wenn dazu Anforderungen, Entwurf, Spezifikationen und Implementierungen von SUT und TAS synchronisiert werden.

Synchronisierung der Fehlerverfolgung
Fehler können sich auf das SUT, die TAS oder die Anforderungen/ Entwürfe/ Spezifikationen beziehen. Aufgrund des Zusammenhangs zwischen beiden Projekten kann die Korrekturmaßnahme für einen Fehler in einem Projekt Auswirkungen auf das andere haben. In die Fehlerverfolgung und die Bestätigungs- bzw. Fehlernachtests müssen beide Systeme einbezogen werden: TAS und SUT.

Synchronisierung der Weiterentwicklung von SUT und TAS
Sowohl das SUT als auch die TAS können weiterentwickelt werden – um neue Funktionen hinzuzufügen, Funktionen zu deaktivieren, Fehler zu beheben oder Anpassungen an Änderungen in ihrer Umgebung vorzunehmen

(das schließt jeweils die Änderungen an SUT und TAS ein, weil beide eine Umgebungskomponente des jeweils anderen Systems sind). Jede Änderung an einem SUT oder einer TAS kann Auswirkungen auf das andere System haben. Das Management dieser Änderungen muss daher SUT und TAS einschließen.

Zwei Synchronisierungsansätze zwischen SUT- und TAS-Entwicklung werden in Abbildung 3 und 4 veranschaulicht.

Abbildung 3 zeigt einen Ansatz, bei dem zwei SDLC-Prozesse für das SUT und die TAS vorrangig in zwei Phasen synchronisiert werden: (1) die TAS-Analyse basiert auf dem SUT-Entwurf, der seinerseits auf der SUT-Analyse basiert, und (2) beim Testen des SUT wird die bereitgestellte TAS genutzt.

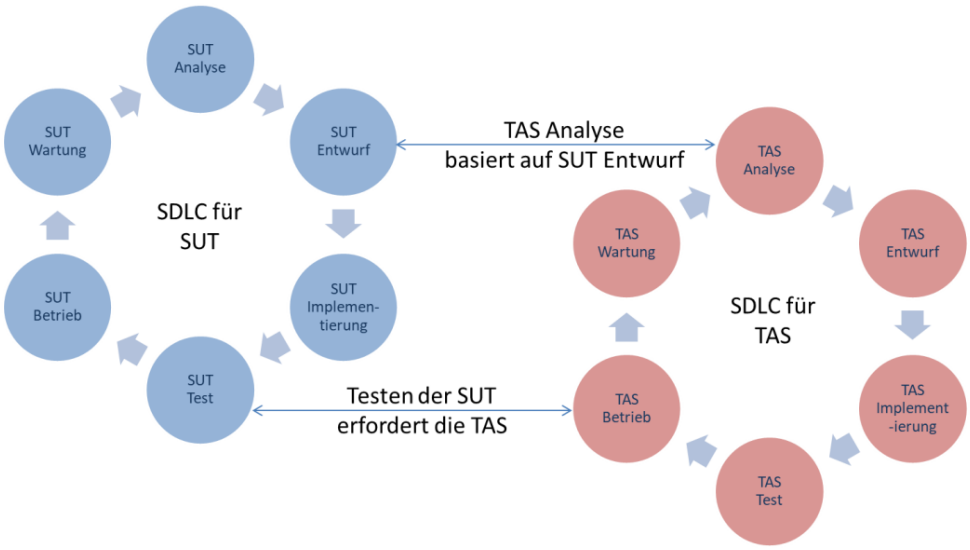

Abbildung 3: Beispiel 1 für die Synchronisierung von TAS- und SUT-Entwicklungsprozess

Abbildung 4 zeigt einen hybriden Ansatz, der manuelles und automatisiertes Testen kombiniert. Wenn manuelle Tests verwendet werden, bevor die Tests automatisiert werden, oder wenn manuelle und automatisierte Tests parallel verwendet werden, muss die TAS-Analyse auf dem SUT-Entwurf und den manuellen Tests basieren. Das garantiert die Synchronisierung der TAS mit beiden. Der zweite wichtige Synchronisierungsaspekt bei diesem Ansatz bleibt derselbe: Das SUT-Testen erfordert bereitgestellte Tests, die bei manuellen Tests einfach in der Befolgung der manuellen Testabläufe bestehen können.

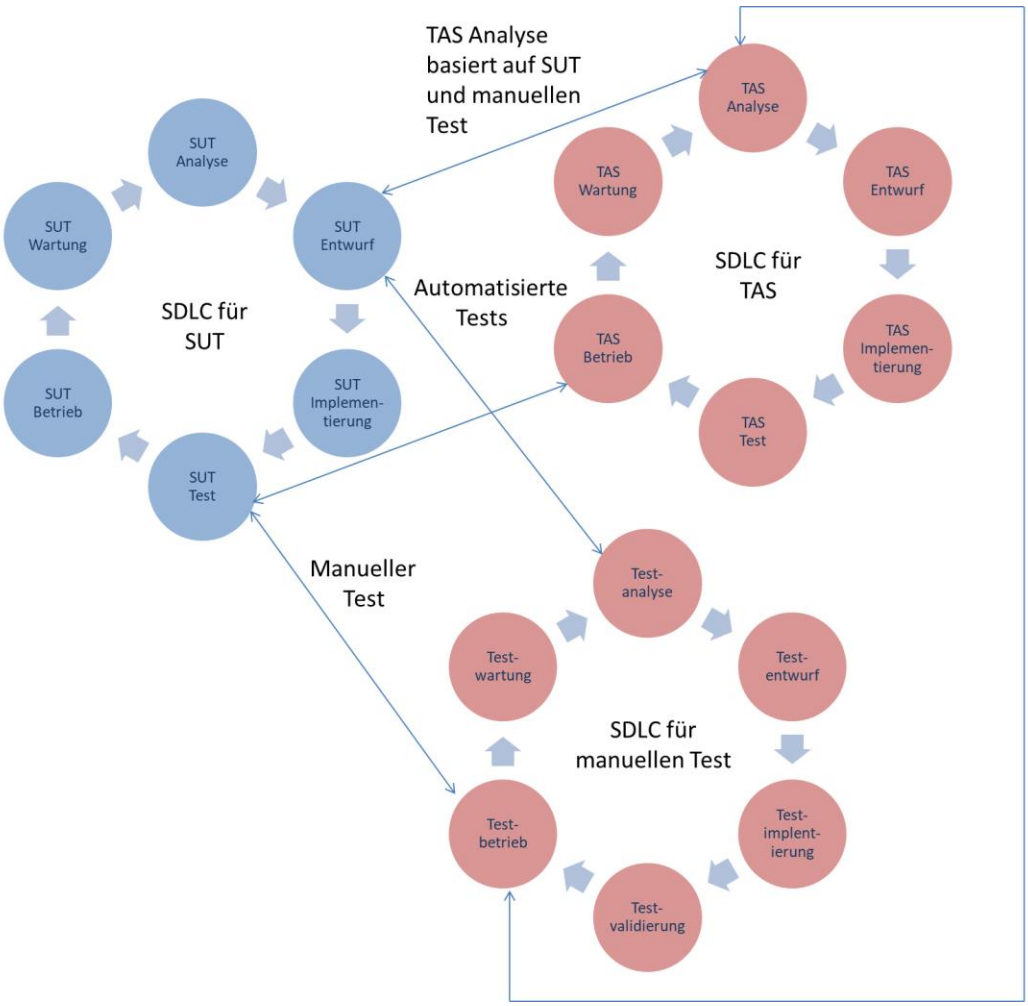

Abbildung 4: Beispiel 2 für die Synchronisierung von TAS- und SUT-Entwicklungsprozess

3.3.4 Einbau von Wiederverwendbarkeit in die TAS

Die Wiederverwendung einer TAS meint die Wiederverwendung von TAS-Artefakten (von jeder Ebene ihrer Architektur) über Produktreihen, Produkt-Frameworks, Produktdomänen und/oder Projektfamilien hinweg. Anforderungen für die Wiederverwendung ergeben sich aus der Relevanz von TAS-Artefakten für die anderen Produktvarianten, Produkte und/oder Projekte. Wiederverwendbare TAS-Artefakte können sein:

- Testmodelle von Testzielen, Testszenarios, Testkomponenten oder Testdaten (oder Teile von ihnen)
- Testfälle, Testdaten, Testabläufe oder Testbibliotheken (oder Teile von ihnen)
- die Test-Engine und/oder das Testberichts-Framework
- die Adapter für die SUT-Komponenten und/oder Schnittstellen

Während Wiederverwendungsaspekte bereits beim Definieren der TAA festgelegt werden, kann die TAS helfen, die Wiederverwendbarkeit zu erhöhen – durch folgende Maßnahmen:

- Befolgung der TAA bzw. deren Überarbeitung und Aktualisierung im Bedarfsfall
- Dokumentieren der TAS-Artefakte, damit diese einfach verständlich sind und in neue Kontexte eingebunden werden können
- Gewährleisten der Richtigkeit jedes TAS-Artefakts, damit die Verwendung in neuen Kontexten durch ihre

hohe Qualität erleichtert wird

Dabei ist unbedingt Folgendes zu beachten: Die Auslegung auf Wiederverwendbarkeit ist vorrangig eine Angelegenheit für die TAA; die Wartung und Verbesserungsmaßnahmen zur Wiederverwendung fallen hingegen in den TAS-Lebenszyklus. Damit es tatsächlich zur Wiederverwendung kommt, bedarf es einer kontinuierlichen Befassung mit der Thematik und entsprechender Bemühungen. Der Mehrwert der Wiederverwendung muss ermittelt und demonstriert werden, um andere von der Wiederverwendung bestehender TASs zu überzeugen.

3.3.5 Unterstützung verschiedener Zielsysteme

Wenn die TAS verschiedene Zielsysteme unterstützen soll, heißt das, dass sie in der Lage sein muss, verschiedene Konfigurationen eines Softwareprodukts zu testen. Verschiedene Konfigurationen können Folgendes sein:

- Anzahl und Vernetzung von SUT-Komponenten
- Umgebungen (Software und Hardware), in denen SUT-Komponenten laufen
- Technologien, Programmiersprachen oder Betriebssysteme, die für die Implementierung der SUT-Komponenten verwendet werden
- Bibliotheken und Pakete, die von SUT-Komponenten genutzt werden
- Werkzeuge zur Implementierung der SUT-Komponenten

Während die ersten vier Aspekte die TAS auf jeder Teststufe betreffen, kommt der letzte Aspekt in erster Linie für das Testen auf Komponenten- und Integrationsteststufe zum Tragen.

Das Vermögen einer TAS, verschiedene Softwareprodukt-Konfigurationen zu testen, wird bei der Definition der TAA festgelegt. Die TAS muss jedoch die Fähigkeit implementieren, die technische Varianz zu handhaben. Zudem muss sie das Management der TAS-Funktionen und -Komponenten ermöglichen, die für verschiedene Konfigurationen eines Softwareprodukts benötigt werden.

Wie die TAS mit den vielfältigen Ausprägungen des Softwareprodukts umgeht, kann unterschiedlich gehandhabt werden:

- Mittels Versions-/Konfigurationsmanagement für die TAS und das SUT können die jeweils zueinander passenden Versionen und Konfigurationen von TAS und SUT bereitgestellt werden.
- Mittels TAS-Parametrisierung kann eine TAS an eine SUT-Konfiguration angepasst werden.

Dabei ist unbedingt Folgendes zu beachten: Die Auslegung der TAS auf Variabilität ist vorrangig eine Angelegenheit für die TAA; die Wartung und Verbesserungsmaßnahmen im Hinblick auf die Variabilität fallen hingegen in den TAS-Lebenszyklus. Es bedarf einer kontinuierlichen Befassung mit der Thematik und entsprechender Bemühungen, um Möglichkeiten und Formen der Variabilität zu prüfen, hinzuzufügen und sogar zu verwerfen.

4. Risiken und Eventualitäten bei der Softwareverteilung – 150min

Begriffe
Produktrisiko, Risiko, Risikobewertung, Risikominderung

Lernziele für „Risiken und Eventualitäten bei der Softwareverteilung“

4.1 Auswahl des Testautomatisierungsansatzes und Planung von Verteilung/Rollout

ALTA-E-4.1.1 (K3) Anwendung von Richtlinien, die effiziente Erprobungs- und Verteilungsaktivitäten von Testwerkzeugen unterstützen

4.2 Strategien für die Bewertung und Begrenzung von Risiken

ALTA-E-4.2.1 (K4) Analysieren von Verteilungsrisiken und Ermitteln technischer Probleme, die zum Scheitern des Testautomatisierungsprojekts führen könnten, und Planen von Risikominderungsstrategien

4.3 Wartung der Testautomatisierung

ALTA-E-4.3.1 (K2) Verstehen, welche Faktoren die Wartbarkeit der TAS unterstützen und beeinflussen

4.1 Auswahl des Testautomatisierungsansatzes und Planung von Softwareverteilung/Rollout

Bei Implementierung und Rollout einer TAS gibt es zwei Hauptaktivitäten: Erprobung (Pilotversuch) und Verteilung (engl. Deployment). Welche konkreten Schritte diese beiden Aktivitäten umfassen, hängt von der Art der TAS und der jeweiligen Situation ab.

Für den Pilotversuch sollten mindestens die folgenden Schritte vorgesehen werden:

- Ermittlung eines geeigneten Projekts
- Planen des Pilotversuchs
- Durchführen des Pilotversuchs
- Evaluieren des Pilotversuchs

Für die Verteilungsphase sollten mindestens die folgenden Schritte vorgesehen werden:

- Ermitteln der ersten Zielprojekte
- Verteilung der TAS in den ausgewählten Projekten
- Überwachung und Evaluierung der TAS in Projekten nach einem vorher festgelegten Zeitraum
- Rollout im übrigen Unternehmen bzw. in den restlichen Projekten

4.1.1 Pilotprojekt

Die Werkzeugimplementierung beginnt in der Regel mit einem Pilotprojekt. Ziel des Pilotprojekts ist es, sicherzustellen, dass die TAS genutzt werden kann, um den geplanten Nutzwert zu realisieren. Ziele des Pilotprojekts:

- Gewinnen von Erkenntnissen über die TAS.
- Sehen, wie sich die TAS in bestehende Prozesse, Abläufe und Werkzeuge einfügt; Ermitteln, wie diese u. U. geändert werden müssen. (In der Regel wird die TAS in Abstimmung auf bestehende Prozesse/Abläufe geändert. Wenn diese verändert werden müssen, um die „TAS zu unterstützen“, sollte das

mindestens eine Verbesserung der Prozesse selbst darstellen.)
- Entwurf der Automatisierungsschnittstelle in Abstimmung auf die Erfordernisse der Tester.
- Wahl standardmäßiger Verfahren der Verwendung, Verwaltung, Speicherung und Wartung der TAS und der Testmittel einschließlich Verzahnung mit dem Konfigurations- und Änderungsmanagement (z. B. Benennungskonventionen für Dateien und Tests festlegen, Bibliotheken erstellen und die Modularität der Testsuites definieren).
- Ermitteln von Metriken und Messverfahren zur Überwachung der Testautomatisierung im Betrieb, einschließlich Nutzbarkeit, Wartbarkeit und Erweiterbarkeit.
- Beurteilen, ob sich der Nutzwert zu vertretbaren Kosten realisieren lässt. Das bietet die Möglichkeit, Erwartungen neu zu formulieren, sobald die TAS genutzt wurden.
- Ermitteln, welche Kompetenzen benötigt werden und welche davon verfügbar sind bzw. fehlen.

Ermittlung eines geeigneten Projekts
Das Pilotprojekt muss unter Berücksichtigung von folgenden Richtlinien sorgfältig ausgewählt werden:

- Es darf kein kritisches Projekt sein. Wenn die Verteilung der TAS zu Verzögerungen führt, darf dies keine ernsten Folgen für kritische Projekte haben. Die Verteilung der TAS ist am Anfang mit hohem Zeitaufwand verbunden. Daran sollte das Projektteam immer denken.
- Es darf kein triviales Projekt sein. Ein triviales Projekt ist keine gute Wahl, weil ein Erfolg bei dessen Bereitstellung nicht automatisch bedeutet, dass auch die Bereitstellung komplexerer Projekte gelingt, und daher der Erkenntnisgewinn gering ist.
- Alle Beteiligten (einschließlich Management) müssen in den Auswahlprozess einbezogen werden.
- Das SUT des Pilotprojekts muss eine gute Referenz für die anderen Projekte des Unternehmens bilden. So sollte das SUT z. B. repräsentative GUI-Komponenten enthalten, die automatisiert werden sollen.

Planen des Pilotversuchs
Das Pilotprojekt sollte als reguläres Entwicklungsprojekt behandelt werden: einen Plan machen, Budget und Ressourcen reservieren, über den Fortschritt informieren, Meilensteine definieren usw. Als weiterer wichtiger Punkt muss gewährleistet werden, dass die Personen, die an der TAS-Bereitstellung arbeiten, ausreichend Zeit für die Bereitstellung aufwenden können – auch wenn andere Projekte die Ressourcen für ihre Aktivitäten fordern. Es ist wichtig, die Unterstützung des Managements zu haben, vor allem wenn es um geteilte Ressourcen geht. Die betreffenden Mitarbeiter werden wahrscheinlich nicht in Vollzeit an der Bereitstellung arbeiten können.

Wenn die TAS nicht von einem externen Anbieter geliefert, sondern im eigenen Haus entwickelt wird, müssen die entsprechenden Entwickler in die Bereitstellungsaktivitäten einbezogen werden.

Durchführen des Pilotversuchs
Bei der Durchführung des Pilotversuchs sollten auf folgende Punkte beachtet werden:

- Bietet die TAS die erwarteten (und vom Anbieter versprochenen) Funktionen? Falls nicht, muss das so schnell wie möglich adressiert werden. Wenn die TAS selbst entwickelt wird, müssen die betreffenden Entwickler die Bereitstellung unterstützen, indem sie fehlende Funktionen nachliefern.
- Unterstützen die TAS und der bestehende Prozess einander? Falls nicht, muss eine Abstimmung erfolgen.

Evaluieren des Pilotversuchs
Es sollten alle Beteiligten in die Evaluierung einbezogen werden.

4.1.2 Bereitstellung

Erst wenn das Pilotprojekt bei der Auswertung als Erfolg gewertet wurde, sollte die TAS in der übrigen Abteilung bzw. im Unternehmen bereitgestellt werden. Das Rollout muss schrittweise und gut gesteuert erfolgen. Erfolgsfaktoren für die Bereitstellung:

- Ein schrittweiser Rollout: Führen Sie den Rollout für das übrige Unternehmen in Schritten (Inkrementen) durch. So erreicht die Unterstützung die neuen Benutzer in „Wellen“ statt auf einmal. So lässt sich die Nutzung der TAS schrittweise ausdehnen. Mögliche Engpässe können ermittelt und beseitigt werden, bevor sie zu echten Problemen werden. Wenn erforderlich, können Lizenzen ergänzt werden.
- Anpassung und Optimierung von Prozessen in Abstimmung auf die Nutzung der TAS: Wenn unterschiedliche Benutzer die TAS nutzen, kommen verschiedene Prozesse mit der TAS in Kontakt und müssen auf die TAS feinabgestimmt werden – oder die TAS bedarf (kleinerer) Abstimmungen auf die Prozesse.
- Bereitstellung von Schulungen und Coaching/Mentoring für neue Benutzer: Neue Benutzer benötigen Schulung und Unterweisung in der Nutzung der neuen TAS. Das muss gewährleistet sein. Schulungen/Workshops müssen den Benutzern angeboten werden, bevor sie tatsächlich mit der TAS arbeiten.
- Definieren der Nutzungsrichtlinien: Es können Richtlinien, Checklisten und FAQs für die Nutzung der TAS verfasst werden. Das kann eine Welle von Anfragen an den Support verhindern.
- Implementierung eines Instruments zur Erfassung von Informationen zur Nutzung in der Praxis: Es sollte eine automatisierte Möglichkeit zur Erfassung von Angaben über die tatsächliche Nutzung der TAS geben. Idealerweise umfasst das nicht nur die Nutzung selbst, sondern auch, welche Teile der TAS (bestimmte Funktionen) genutzt werden. Auf diese Weise lässt sich die Nutzung der TAS einfach überwachen.
- Überwachung der Nutzung, des Nutzwertes und der Kosten der TAS: Die Überwachung der Nutzung der TAS über einen bestimmten Zeitraum hinweg gibt Aufschluss darüber, ob die TAS tatsächlich genutzt wird. Diese Informationen können auch für eine erneute Wirtschaftlichkeitsberechnung genutzt werden (z. B. wie viel Zeit wurde gespart, wie viele Probleme wurden verhindert).
- Leisten von Unterstützung für das Test- und Entwicklungsteam für eine TAS.
- Sammeln von gewonnenen Erkenntnissen von allen Teams: Durchführung von Evaluierungs-/Bewertungssitzungen mit den verschiedenen Teams, die die TAS nutzen. So lassen sich gewonnene Erkenntnisse ermitteln. Die Teams spüren, dass ihr Input gebraucht wird, um die Nutzung der TAS zu verbessern.
- Ermitteln und Umsetzen von Verbesserungen: Gestützt auf das Feedback des Teams und die Überwachung der TAS ermitteln Sie Verbesserungsmaßnahmen und setzen diese um. Das ist auch den Beteiligten unmissverständlich zu vermitteln.

4.1.3 Verteilung der TAS innerhalb des Softwarelebenszyklus

Die Verteilung einer TAS hängt in großem Maß von der Entwicklungsphase des Softwareprojekts ab, das von der TAS getestet wird.

Eine neue TAS oder eine neue Version von ihr wird in der Regel entweder zu Projektbeginn oder bei Erreichen eines Meilensteins (z. B. Code-Freeze oder Ende eines Sprints) bereitgestellt. Grund dafür ist, dass die Verteilungsaktivitäten mit allen damit einhergehenden Tests und Modifikationen Zeit und Aufwand erfordern. Zudem ist das eine gute Möglichkeit, das Risiko zu begrenzen, dass die TAS nicht funktioniert und Unterbrechungen im Testautomatisierungsprozess bewirkt. Wenn es bei der TAS kritische Probleme gibt, die behoben werden müssen oder wenn eine Komponente in der Umgebung, in der sie läuft, ausgetauscht werden muss, dann erfolgt die Verteilung jedoch unabhängig von der Entwicklungsphase des SUT.

4.2 Strategien für die Bewertung und Begrenzung von Risiken

Technische Probleme können zu Produkt- oder Projektrisiken führen. Typische technische Probleme sind:

- Zu starke Abstraktion kann es erschweren, die tatsächlichen Abläufe zu verstehen (z. B. bei Schlüsselwörtern).
- Datengetrieben: Datentabellen können zu groß/ komplex/ unübersichtlich werden.

- Abhängigkeit der TAS, bestimmte Betriebssystem-Bibliotheken oder andere Komponenten verwenden zu müssen, die nicht in allen Zielumgebungen des SUT verfügbar sind

Typische Risiken eines Softwareverteilungsprojekts:

- Probleme mit der personellen Ausstattung: Es kann schwierig sein, die richtigen Leute für die Wartung der Codebasis zu bekommen.
- Neue SUT-Arbeitsergebnisse können den fehlerhaften Betrieb der TAS bewirken.
- Verzögerungen bei der Einführung der Automatisierung
- Verzögerungen bei der Aktualisierung der TAS auf Basis der Änderungen am SUT
- Die TAS kann die (nicht standardmäßigen) Objekte nicht erfassen, die sie verfolgen soll.

Potentielle Fehlerpunkte des TAS-Projekts:

- Migration in eine andere Umgebung
- Verteilung in der Zielumgebung
- neue Lieferung aus der Entwicklung

Diesen Risiken kann mit einer Reihe von Risikominderungsstrategien begegnet werden. Diese werden nachstehend besprochen.

Die TAS hat einen eigenen Softwarelebenszyklus, unabhängig davon, ob sie im eigenen Haus entwickelt oder fremdbeschafft wird. Eines darf man nicht vergessen: Die TAS muss wie jede andere Software auch unter Versionskontrolle stehen und ihre Funktionen müssen dokumentiert werden. Andernfalls wird es sehr schwierig, verschiedene Teile von ihr bereitzustellen und zusammenarbeiten zu lassen oder in bestimmten Umgebungen arbeiten zu lassen.

Zudem muss es ein dokumentiertes, klares und einfach zu befolgendes Verteilungsverfahren geben. Dieses Verfahren ist versionsabhängig; daher muss es ebenfalls unter Versionskontrolle gestellt werden.

Es gibt zwei prägnante Fälle der Verteilung einer TAS:

1. Erstverteilung
2. Wartungsverteilung: die TAS existiert bereits und muss gewartet werden

Vor dem Start der Erstverteilung einer TAS muss sichergestellt sein, dass sie in ihrer eigenen Umgebung ausgeführt werden kann, isoliert von zufälligen Änderungen ist und Testfälle aktualisiert und verwaltet werden können. Sowohl die TAS als auch ihre Infrastruktur müssen gewartet werden.

Bei der Erstbereitstellung müssen folgende grundlegende Schritte ausgeführt werden:

- Definieren der Infrastruktur, in der die TAS ausgeführt wird
- Erstellen der Infrastruktur für die TAS
- Erstellen eines Verfahrens für die Wartung der TAS und ihrer Infrastruktur
- Erstellen eines Verfahrens für die Wartung der Testsuite, die die TAS ausführen wird

Folgende Risiken gehen mit der Erstbereitstellung einher:

- Die Gesamtausführungszeit der Testsuite kann größer als die geplante Ausführungszeit für den Testzyklus sein. In diesem Fall muss sichergestellt werden, dass die Testsuite genug Zeit bekommt, um komplett ausgeführt zu werden, bevor der nächste geplante Testzyklus beginnt.
- Es gibt Installations- und Konfigurationsprobleme mit der Testumgebung (z. B. Datenbank-Setup und Anfangslast, Start/Stopp von Diensten). Im Allgemeinen benötigt die TAS einen effektiven Weg, die nötigen Vorbedingungen für die automatisierten Testfälle innerhalb der Testumgebung zu erfüllen.

Bei Wartungsverteilungen sind zusätzliche Punkte zu berücksichtigen. Die TAS selbst muss sich weiterentwickeln, und die Updates für sie müssen in der Produktion bereitgestellt werden. Vor der Verteilung einer aktualisierten Version der TAS in der Produktion muss diese wie jede andere Software getestet werden. Daher müssen die neuen Funktionen geprüft werden, um sicherzustellen, dass die Testsuite in der aktualisierten TAS

ausgeführt werden kann, dass Berichte gesendet werden können und dass es keine Leistungsprobleme oder andere funktionale Regressionen gibt. In einigen Fällen muss u. U. die gesamte Testsuite in Abstimmung auf die neue Version der TAS geändert werden.

Bei der Wartungsverteilung müssen folgende Schritte ausgeführt werden:

- Durchführung einer Bewertung der Änderungen in der neuen Version der TAS im Vergleich zur alten Version
- Testen der TAS auf neue Funktionen und Regressionen
- Prüfen, ob die Testsuite in Abstimmung auf die neue Version der TAS geändert werden muss

Ein Update bringt folgende Risiken und entsprechende Maßnahmen zu deren Begrenzung mit sich:

- Die Testsuite muss geändert werden, um in der aktualisierten TAS zu laufen: Nehmen Sie die erforderlichen Änderungen an der Testsuite vor, und testen Sie diese vor der Verteilung in der TAS.
- Beim Testen verwendete Platzhalter, Treiber und Schnittstellen müssen an die aktualisierte TAS angepasst werden: Nehmen Sie die nötigen Änderungen am Testrahmen vor, und testen Sie ihn vor seiner Verteilung in der TAS.
- Die Infrastruktur muss in Abstimmung auf die aktualisierte TAS geändert werden: Nehmen Sie eine Bewertung der Infrastrukturkomponenten vor, die geändert werden müssen, nehmen Sie die Änderungen vor, und testen Sie sie an der aktualisierten TAS.
- Die aktualisierte TAS hat weitere Defekte oder Performanzprobleme: Führen Sie eine Risiko-Nutzen-Analyse durch. Wenn die entdeckten Probleme die Aktualisierung der TAS unmöglich machen, ist es u. U. besser, nicht mit der Aktualisierung fortzufahren oder auf die nächste Version der TAS zu warten. Wenn die Probleme im Vergleich zum Nutzen vernachlässigbar sind, kann die TAS aktualisiert werden. Sie müssen unbedingt Release Notes mit bekannten Problemen verfassen, um die TAEs und andere Beteiligte zu informieren. Versuchen Sie außerdem abzuschätzen, wann die Probleme behoben werden.

4.3 Wartung der Testautomatisierung

Die Entwicklung von Testautomatisierungslösungen ist kein triviales Unterfangen. Die Lösungen müssen modular, skalierbar, verständlich, zuverlässig und testbar sein. Noch komplexer wird die Sache, weil sich Testautomatisierungslösungen – wie andere Softwaresysteme auch – weiterentwickeln müssen. Ob aufgrund interner Änderungen oder Änderungen an der Umgebung, in der sie operieren – Wartung ist ein wichtiger Aspekt der Entwicklung der Architektur einer TAS. Die Wartung der TAS durch ihre Anpassung an neue Arten von zu testenden Systemen, durch Erweiterung der Unterstützung auf neue Softwareumgebungen oder durch Schaffung der Konformität mit neuen Gesetzen und Vorschriften trägt dazu bei, einen zuverlässigen und sicheren Betrieb der TAS zu gewährleisten. Zudem optimiert sie die Lebensdauer und die Leistungsfähigkeit der TAS.

4.3.1 Arten der Wartung

Die Wartung erfolgt an einer bestehenden, in Betrieb befindlichen TAS und wird durch Modifikationen, Migration oder Außerbetriebnahme des Systems ausgelöst. Dieser Prozess lässt sich in folgende Kategorien gliedern:

- Präventive Wartung: Es werden Änderungen vorgenommen, damit die TAS mehr Testarten unterstützt, an mehreren Schnittstellen testet, mehrere Versionen des SUT testet oder die Testautomatisierung für ein neues SUT unterstützt.
- Korrektive Wartung: Es werden Änderungen vorgenommen, um Fehler der TAS zu beheben. Am besten und mit dem geringsten Risiko lässt sich eine in Betrieb befindliche TAS warten, indem man regelmäßige Wartungstests durchführt.
- Verbessernde Wartung: Die TAS wird optimiert und nichtfunktionale Fehler werden behoben. Sie kann die Leistungsfähigkeit der TAS, ihre Benutzbarkeit, ihre Robustheit oder Zuverlässigkeit verbessern.
- Adaptive Wartung: Wenn neue Softwaresysteme auf den Markt kommen (Betriebssysteme,

Datenbankverwaltungsprogramme, Webbrowsers usw.), muss die TAS sie u. U. unterstützen können. Es kann auch der Fall sein, dass die TAS neue Gesetze, Vorschriften oder branchenspezifische Anforderungen erfüllen muss. In diesem Fall werden Änderungen an der TAS vorgenommen, um sie entsprechend anzupassen. Hinweis: Konformität mit Gesetzen und Vorschriften bedingt in der Regel eine verpflichtende Wartung mit speziellen Regeln, Anforderungen und mitunter vorgeschriebenen Prüfungen (Audits). Zudem gilt: Wenn integrierende Werkzeuge aktualisiert und neue Versionen erstellt werden, müssen die Endpunkte der Werkzeugintegration gewartet und funktional gehalten werden.

4.3.2 Umfang und Ansatz

Wartung ist ein Prozess, der alle Schichten und Komponenten einer TAS betreffen kann. Ihr Umfang hängt von folgenden Faktoren ab:

- Größe und Komplexität der TAS
- Größe der Änderung
- Risiko der Änderung

Angesichts des Umstands, dass die Wartung die in Betrieb befindliche TAS betrifft, muss im Rahmen einer Auswirkungsanalyse ermittelt werden, wie sich die Änderungen auf das System auswirken können. Je nach den Auswirkungen müssen schrittweise Änderungen vorgenommen und nach jedem Schritt Tests durchgeführt werden, um sicherzustellen, dass die TAS unterbrechungsfrei funktioniert. Hinweis: Die Wartung der TAS kann schwierig sein, wenn ihre Spezifikationen und ihre Dokumentation veraltet sind.

Weil zeitliche Effizienz der wichtigste Faktor für den Erfolg der Testautomatisierung ist, bedarf es bewährter Vorgehensweisen für die Wartung der TAS. Dazu zählen:

- Die Verteilungsverfahren und die Nutzung der TAS müssen klar und dokumentiert sein.
- Die Abhängigkeiten von Dritten müssen dokumentiert werden – zusammen mit Nachteilen und bekannten Problemen.
- Die TAS muss modular sein, damit sich Teile von ihr einfach ersetzen lassen.
- Die TAS muss in einer Umgebung ausgeführt werden, die austauschbar ist oder austauschbare Komponenten hat.
- Die TAS muss die Testskripte vom TAF selbst trennen.
- Die TAS muss isoliert von der Entwicklungsumgebung ausgeführt werden, damit Änderungen an der TAS keine Beeinträchtigung der Testumgebung zur Folge haben.
- Die TAS muss zusammen mit Testumgebung, Testsuite und Testmittel unter Konfigurationsmanagement stehen.

Zudem ist die Wartung von Fremdkomponenten und -bibliotheken zu berücksichtigen:

- Häufig ist es der Fall, dass eine TAS für die Ausführung der Tests auf Fremdkomponenten zugreift. Zudem kann es der Fall sein, dass die TAS von Fremdbibliotheken abhängt (z. B. den UI-Automatisierungsbibliotheken). Alle Fremdkomponententeile der TAS müssen dokumentiert werden und unter Konfigurationsmanagement stehen.
- Es muss einen Plan für den Fall geben, dass diese externen Komponenten modifiziert oder von Fehlern bereinigt werden müssen. Der Verantwortliche für die Wartung der TAS muss wissen, an wen er sich wendet oder wo er Fehler melden muss.
- Es muss eine Dokumentation bezüglich der Lizenz geben, unter der die Fremdkomponenten genutzt werden, damit Informationen dazu vorliegen, ob, zu welchem Grad und von wem sie modifiziert werden können.
- Für jede der Fremdkomponenten müssen Informationen über Updates und neue Versionen eingeholt werden. Die Fremdkomponenten und -bibliotheken aktuell zu halten, ist eine präventive Maßnahme, deren Aufwand sich langfristig bezahlt macht.

Folgende Überlegungen zu Benennungsstandards und anderen Konventionen müssen angestellt werden:

- Dass Benennungsstandards und andere Konventionen anzuwenden sind, hat einen einfachen Grund: Die Testsuite und die TAS selbst müssen einfach lesbar, verständlich, sowie zu ändern und zu warten sein. Das spart Zeit im Wartungsprozess und minimiert zudem das Risiko des Einschleusens von Regressionen oder falschen Fixes, die sich einfach hätten vermeiden lassen.
- Wenn Standard-Benennungskonventionen genutzt werden, ist es einfacher, neue Mitarbeiter an ein Testautomatisierungsprojekt heranzuführen.
- Die Benennungsstandards können sich auf Variablen und Dateien, Testszenarios, Schlüsselwörter und Schlüsselwortparameter beziehen. Sonstige Konventionen beziehen sich auf Voraussetzungen und Anschlussmaßnahmen der Testausführung, den Inhalt der Testdaten, die Testumgebung, den Status der Testausführung sowie Ausführungsprotokolle und -berichte.
- Alle Standards und Konventionen müssen bei Beginn eines Testautomatisierungsprojekts vereinbart und dokumentiert werden.

Überlegungen zur Dokumentation:

- Die Notwendigkeit einer guten und aktuellen Dokumentation für die Testszenarios und die TAS liegt auf der Hand. Dabei gibt es jedoch zwei Probleme: Einer muss sie schreiben, und einer muss sie pflegen.
- Während der Code des Testwerkzeugs selbstdokumentierend sein kann oder halbautomatisch dokumentiert werden kann, müssen alle Entwürfe, Komponenten, Integrationen mit Dritten, Abhängigkeiten und Verteilungsverfahren von jemandem dokumentiert werden.
- Es ist gute Praxis, das Schreiben der Dokumentation zum Bestandteil des Entwicklungsprozesses zu machen. Eine Aufgabe darf erst dann als erledigt gelten, wenn sie dokumentiert wurde oder die zugehörige Dokumentation aktualisiert wurde.

Überlegungen zum Schulungsmaterial:

- Wenn die Dokumentation der TAS entsprechend geschrieben ist, kann sie als Basis für das Schulungsmaterial der TAS dienen.
- Das Schulungsmaterial ist eine Kombination aus funktionalen Spezifikationen der TAS, Entwurf und Architektur der TAS, Verteilung und Wartung der TAS, Nutzung der TAS (Benutzerhandbuch), praktischen Beispielen und Übungen sowie Tipps und Tricks.
- Die Wartung des Schulungsmaterials besteht aus dem Verfassen des Materials und seiner regelmäßigen Überprüfung und Aktualisierung. In der Praxis wird das von Teammitgliedern erledigt, die zu Ausbildern für die TAS ernannt wurden. In der Regel passiert das gegen Ende einer Lebenszyklus-Iteration des SUT (z. B. am Ende von Sprints).

5. Berichte und Metriken bei der Testautomatisierung – 165 min

Begriffe
Äquivalenter manueller Testaufwand (EMTE: Equivalent Manual Test Effort), Fehlerdichte des Automatisierungscodes, Metrik, Rückverfolgbarkeitsmatrix, Testprotokollierung, Testberichterstattung, Überdeckungsgrad

Lernziele für „Berichte und Metriken bei der Testautomatisierung“

5.1 Auswahl von TAS-Metriken
ALTA-E-5.1.1 (K2) Klassifizierung von Metriken, mit denen sich die Testautomatisierungsstrategie und deren Wirksamkeit überwachen lassen

5.2 Implementierung der Messwerterfassung
ALTA-E-5.2.1 (K3) Implementieren von Erfassungsmethoden für Metriken zur Unterstützung von technischen Anforderungen und Managementanforderungen. Erläutern, wie sich die Messung für die Testautomatisierung implementieren lässt.

5.3 Protokollierung von TAS und SUT
ALTA-E-5.3.1 (K4) Analysieren der Testprotokollierung von TAS- und SUT-Daten

5.4 Erstellung von Berichten zur Testautomatisierung
ALTA-E-5.4.1 (K2) Erläutern, wie ein Testausführungsbericht aufgebaut ist und veröffentlicht wird

5.1 Auswahl von TAS-Metriken

Im Mittelpunkt dieses Abschnitts stehen Metriken, mit denen sich die Testautomatisierungsstrategie sowie die Wirksamkeit der TAS überwachen lassen. Diese TAS-Metriken werden getrennt von den SUT-bezogenen Metriken erfasst, die zur Überwachung des SUT sowie des (funktionalen und nicht-funktionalen) Testens des SUT verwendet werden. TAS-Metriken werden vom Testmanager für das gesamte Projekt ausgewählt. Sie ermöglichen dem TAM und dem TAE die Fortschrittsverfolgung in Bezug auf die Ziele der Testautomatisierung und die Überwachung der Auswirkungen von Änderungen an der Testautomatisierungslösung.

Die TAS-Metriken lassen sich in zwei Gruppen untergliedern: externe und interne Metriken. Mit externen Metriken werden die Auswirkungen der TAS auf andere Aktivitäten gemessen (besonders die Testaktivitäten). Interne Metriken werden für die Messung der Wirksamkeit und Effizienz der TAS im Hinblick auf die Erfüllung ihrer Ziele benutzt.

Folgende TAS-Metriken werden in der Regel ermittelt:

- externe TAS-Metriken
  - Nutzwert der Automatisierung
  - Aufwand für die Erstellung automatisierter Tests
  - Aufwand für die Analyse der bei automatisierten Tests ermittelten Abweichungen
  - Aufwand für die Wartung automatisierter Tests
  - Verhältnis von fehlgeschlagenen Tests zu Fehlern im SUT
  - Ausführungszeit automatisierter Tests
  - Anzahl der automatisierten Testfälle
  - Anzahl der positiven und negativen Ergebnisse
  - Anzahl der falsch-negativen und falsch-positiven Ergebnisse
  - Überdeckungsgrad des Quellcodes
- interne TAS-Metriken
  - Skriptmetriken

- Fehlerdichte des Automatisierungscodes
- Geschwindigkeit und Effizienz der TAS-Komponenten

Diese Metriken werden nachstehend beschrieben.

Nutzwert der Automatisierung
Da der Aufwand einer TAS / die Anzahl der involvierten Mitarbeiter über einen bestimmten Zeitraum gut ersichtlich ist, ist es wichtig, den Nutzwert einer TAS zu ermitteln und zu dokumentieren. So sehen auch Personen außerhalb des Testprojekts den erzielten Nutzwert und haben nicht nur einen Eindruck vom Gesamtaufwand.

Jede Nutzen-Messung hängt vom Ziel der TAS ab. In der Regel sind es Einsparungen im Hinblick auf Zeit und Aufwand, eine Erhöhung der Anzahl der Tests (Breite oder Tiefe des Überdeckungsgrades bzw. Ausführungshäufigkeit) oder ein anderer Vorteil wie eine bessere Wiederholbarkeit, eine stärkere Nutzung von Ressourcen oder weniger manuelle Fehler. Mögliche Kennziffern sind:

- Anzahl der Stunden, die für manuelle Tests gespart wurden
- Verkürzung der Zeit für die Durchführung von Regressionstests
- Anzahl der erzielten zusätzlichen Testausführungszyklen
- Anzahl zusätzlich ausgeführter Tests in Prozent
- Anteil der automatisierten Testfälle in Relation zur Gesamtanzahl der Testfälle (auch wenn sich automatisierte nicht einfach mit manuellen Testfällen vergleichen lassen)
- Erhöhung des Überdeckungsgrades (Anforderungen, Funktionsumfang, strukturell)
- Anzahl der dank der TAS vorzeitig gefundenen Fehler (wenn der mittlere Nutzwert vorzeitig gefundener Fehler bekannt ist, lässt sich daraus eine Summe der gesparten Kosten „berechnen“)
- Anzahl der Fehler, die dank der TAS gefunden wurden und durch manuelles Testen nicht gefunden worden wären (z. B. Zuverlässigkeitsfehler)

Zu beachten ist, dass durch die Testautomatisierung generell manueller Testaufwand eingespart wird. Dieser Aufwand kann in andere (manuelle) Tests fließen (z. B. explorative Tests). Fehler, die durch diese zusätzlichen Tests gefunden werden, können auch als indirekter Nutzen der TAS gesehen werden, weil die Durchführung dieser manuellen Tests erst durch die Automatisierung anderer Tests möglich wurde. Ohne die TAS wären diese Tests nicht durchgeführt und folglich auch die zusätzlichen Fehler nicht gefunden worden.

Aufwand für die Erstellung automatisierter Tests
Der Aufwand für die Automatisierung von Tests ist ein wesentlicher Kostenpunkt bei der Testautomatisierung. Häufig übersteigt er die Kosten für die manuelle Ausführung desselben Tests und kann daher ein Argument gegen eine Ausweitung der Testautomatisierung sein. Die Kosten für die Implementierung eines bestimmten automatisierten Tests hängen weitgehend vom Test selbst ab. Andere Faktoren wie der verwendete Automatisierungsansatz, die Vertrautheit mit dem Testwerkzeug, die Umgebung und die Fähigkeiten des TAE haben jedoch ebenfalls einen Einfluss.

Bei umfangreicheren oder komplexeren Tests dauert die Automatisierung in der Regel länger als bei kurzen oder einfachen Tests. Die Berechnung der Entwicklungskosten für die Testautomatisierung sollte daher anhand der durchschnittlichen Entwicklungszeit vorgenommen werden. Das lässt sich noch verfeinern, indem man die durchschnittlichen Kosten für eine bestimmte Gruppe von Tests (wie die Tests für eine bestimmte Funktion oder auf einer bestimmten Ebene) berücksichtigt. Ein weiterer Ansatz ist die Darstellung der Entwicklungskosten als Faktor des Aufwands für die manuelle Ausführung des Tests (EMTE). Die Automatisierung eines Testfalls kann z. B. den doppelten manuellen Testaufwand, also zweimal den EMTE erfordern.

Aufwand für die Analyse von SUT-Fehlern
Die Analyse von Fehlern im SUT, die bei automatisierten Tests entdeckt wurden, kann deutlich komplexer sein als bei einem manuell ausgeführten Test, da der Tester die Ereignisse, die zu einem Fehler führen, häufig kennt. Das Problem lässt sich auf Entwurfsebene (siehe Kapitel 3.1.4) sowie auf Berichterstattungsebene (siehe Kapitel 5.3 und 5.4) entschärfen. Eine Kennziffer hierzu kann als Mittelwert pro fehlgeschlagenen Testfall oder als ein

Faktor von EMTE dargestellt werden. Letzteres eignet sich besonders dann, wenn automatisierte Tests in ihrer Komplexität und Ausführungszeit erheblich variieren.

Die Verfügbarkeit der Protokollierung des SUT und der TAS spielen eine entscheidende Rolle bei der Analyse von Fehlern. Die Protokollierung muss genügend Informationen für eine effiziente Durchführung dieser Analyse liefern. Wichtige Anforderungen an die Protokollierung:

- SUT- und TAS-Protokollierung müssen synchron erfolgen
- die TAS muss das erwartete und das tatsächliche Verhalten protokollieren
- die TAS muss die auszuführenden Aktionen protokollieren

Das SUT muss alle von ihm ausgeführten Aktionen protokollieren. Das ist unabhängig davon, ob eine Aktion das Ergebnis des manuellen oder automatisierten Testens ist. Interne Fehler müssen protokolliert werden. Es müssen Absturz-Speicherabbilder (Crash Dumps) und Stack Traces verfügbar sein.

Aufwand für die Wartung automatisierter Tests
Der benötigte Wartungsaufwand, um automatisierte Tests synchron zum SUT zu halten, kann erheblich schwanken und letztlich den Nutzwert der TAS übersteigen. Es ist ein wesentlicher Grund dafür, dass viele Automatisierungsprojekte fehlschlagen. Um rechtzeitig Schritte zur Reduzierung des Wartungsaufwands zu ergreifen oder zumindest das unkontrollierte Anwachsen zu verhindern, ist es daher wichtig, den Wartungsaufwand zu überwachen.

Kennziffern für den Wartungsaufwand können als Summe aller automatisierten Tests mit Wartungsbedarf bei jedem neuen Release des SUT dargestellt werden. Sie lassen sich aber auch als Mittelwert pro aktualisiertem, automatisiertem Test oder als Faktor von EMTE darstellen.

Eine zugehörige Metrik ist die Anzahl oder der Anteil von Tests, die Wartungsarbeiten benötigen.

Wenn der Wartungsaufwand für automatisierte Tests bekannt ist bzw. sich ermitteln lässt, können diese Informationen eine wichtige Rolle bei der Entscheidung spielen, ob bestimmte Funktionen implementiert oder bestimmte Fehlerzustände behoben werden sollen. Bei Änderung des SUT sollte auch der damit daraus folgende Aufwand für die Wartung des Testfalls berücksichtigt werden.

Verhältnis von fehlgeschlagenen Tests zu Fehlern im SUT
Ein häufiges Problem bei automatisierten Tests ist, dass viele von ihnen aufgrund desselben Fehlers in der Software fehlschlagen können. Tests haben den Zweck, Fehler in der Software zu entdecken. Wenn mehrere Tests denselben Fehler entdecken, ist das Ressourcenverschwendung. Das gilt besonders für das automatisierte Testen, weil der Aufwand für die Analyse jedes fehlgeschlagenen Tests erheblich sein kann. Das Messen der Anzahl der automatisierten Tests, die durch einen bestimmten Fehler im getesteten System fehlschlagen, kann helfen, Fälle zu ermitteln, wo dies ein Problem sein kann. Die Lösung liegt im Entwurf der automatisierten Tests und ihrer Auswahl zur Ausführung.

Ausführungszeit automatisierter Tests
Eine einfach zu ermittelnde Metrik ist die Zeit, die für die Ausführung der automatisierten Tests benötigt wird. Am Anfang mag das noch nebensächlich sein, mit steigender Anzahl der automatisierten Testfälle in der TAS kann diese Metrik jedoch wichtig werden.

Anzahl der automatisierten Testfälle
Mit dieser Metrik kann gezeigt werden, welchen Fortschritt das Testautomatisierungsprojekt nimmt. Dabei ist jedoch auch zu bedenken, dass die bloße Anzahl der automatisierten Testfälle noch nicht viel besagt, z. B. geht aus ihr nicht hervor, ob die Testüberdeckung gestiegen ist.

Anzahl der positiven und negativen Ergebnisse
Das ist eine allgemeine Metrik, die angibt, wie viele automatisierte Tests auf dem Weg zum Erreichen des erwarteten Ziels bestanden wurden und wie viele fehlgeschlagen sind. Fehlschläge müssen analysiert werden, um zu ermitteln, ob die Ursache ein Fehler im SUT oder ein externes Problem wie ein Problem mit der Umgebung oder der TAS selbst war.

Anzahl der falsch-negativen und falsch-positiven Ergebnisse
Wie man bei einigen Metriken zuvor bereits gesehen hat, kann die Analyse fehlgeschlagener Tests zeitaufwändig sein. Stellt sich ein Fehler als Fehlalarm (falsch-positives Ergebnis) heraus, ist das natürlich umso frustrierender. Das passiert, wenn das Problem in der TAS oder im Testfall statt im SUT liegt. Die Zahl der Fehlalarme (und damit der unnütze Aufwand) muss unbedingt klein gehalten werden. Falsch-positive Ergebnisse können das Vertrauen in die TAS schmälern. Umgekehrt gilt: Falsch-negative Ergebnisse können gefährlicher sein. Bei einem falsch-negativen Ergebnis lag ein Fehler im SUT vor, den die Testautomatisierung nicht erkannte und daraufhin ein Bestanden-Ergebnis meldete. Ein möglicher Fehler bleibt dadurch u. U. unerkannt. Das kann der Fall sein, wenn die Verifizierung des Ergebnisses nicht richtig erfolgte, ein ungültiges Testorakel verwendet wurde oder der Testfall das falsche Ergebnis erwartete.

Beachten Sie, dass Fehlalarme durch Fehler im Testquellcode (siehe Metrik „Fehlerdichte des Automatisierungscodes“), aber auch durch ein instabiles SUT, das sich unvorhersehbar verhält (z. B. die Ausführungszeit überschreitet), ausgelöst werden können. Aufgrund des Intrusionsgrads, den sie bewirken, können auch Test Hooks Fehlalarme auslösen.

Codeüberdeckung (Code Coverage)
Die Ermittlung der SUT-Codeüberdeckung durch die verschiedenen Testfälle kann hilfreiche Informationen liefern. Dies lässt sich auch auf abstrakter Ebene ermitteln, z. B. die Codeüberdeckung der Regressionstestsuite. Es gibt keinen absoluten Prozentwert, der einen angemessenen Überdeckungsgrad signalisiert. Eine Codeüberdeckung von 100 % ist lediglich in sehr einfachen Softwareanwendungen zu erzielen. Man ist sich jedoch einig, dass eine höhere Überdeckung generell besser ist, weil sie das Gesamtrisiko einer gelieferten Software senkt. Diese Metrik kann auch ein Zeichen für Aktivitäten im SUT sein. Ein Beispiel: Wenn die Codeüberdeckung sinkt, heißt das wahrscheinlich, dass dem SUT Funktionen hinzugefügt wurden, die automatisierte Testsuite jedoch nicht um einen entsprechenden Testfall erweitert wurde.

Skriptmetriken
Es gibt viele Metriken, mit denen sich die die Entwicklung von Automatisierungsskripten überwachen lässt. Die meisten von ihnen ähneln Quellcode-Metriken für das SUT. Mit LOC (Lines of Code) und zyklomatischer Komplexität lassen sich zu umfangreiche oder zu komplexe Skripte ermitteln, die überarbeitet werden sollten.

Das Verhältnis von Kommentaren zu ausführbaren Anweisungen liefert einen Hinweis auf den Umfang der Dokumentation und Annotation eines Skripts. Die Zahl der Konformitätsverstöße gegen Skripterstellungsstandards kann einen Hinweis darauf geben, inwieweit die betreffenden Standards eingehalten werden.

Fehlerdichte des Automatisierungscodes
Automatisierungscode unterscheidet sich dahingehend nicht vom Code des SUT, als dass es sich um Software handelt, die Fehler enthält. Automatisierungscode darf nicht als weniger wichtig als der SUT-Code betrachtet werden. Es müssen gute Kodierungspraktiken und -standards angewendet werden, und das Ergebnis muss mit Metriken wie der Codefehlerdichte überwacht werden. Einfacher sind die dafür nötigen Daten mit Hilfe eines Konfigurationsmanagementsystems zu erfassen.

Geschwindigkeit und Effizienz der TAS-Komponenten
Wenn die Ausführung derselben Testschritte in derselben Umgebung unterschiedliche lange dauert, kann dies ein Hinweis auf ein Problem im SUT sein. Wenn das SUT dieselben Funktionen nicht immer in derselben Zeitspanne ausführt, muss das untersucht werden. Das kann Zeichen für eine Variabilität im System sein, die nicht akzeptabel ist und sich mit steigender Last verschlimmern könnte. Die TAS muss stabil genug laufen, um die Leistungsfähigkeit des SUT nicht zu beeinträchtigen. Wenn die Leistungsfähigkeit eine kritische Anforderung für das SUT ist, muss die TAS so ausgelegt werden, dass diese Berücksichtigung findet.

Trend-Metriken
Bei vielen der genannten Metriken sind die Trends (d.h., wie sich die Messwerte über die Zeit verändern) häufig aufschlussreicher als die reinen Messwerte zu einem bestimmten Zeitpunkt. Ein Beispiel: Wenn die mittleren Wartungskosten pro automatisiertem Test mit Wartungsbedarf über denen der beiden Vorgängerversionen des SUT liegen, kann es ratsam sein, die Ursache für dieses Problem zu ermitteln und Maßnahmen zur Umkehr

dieses Trends einzuleiten.

Die Kosten für das Messen sollten so niedrig wie möglich sein. Häufig lässt sich das durch die Automatisierung der Erfassung und Berichterstattung erreichen.

5.2 Implementierung der Messwerterfassung

Da automatisierte Testmittel das Herzstück einer Testautomatisierungsstrategie bilden, können sie so erweitert werden, dass sie Informationen zu ihrer Nutzung aufzeichnen. Bei der Kombination von Abstraktion mit strukturierten Testmitteln lassen sich Erweiterungen der zugrunde liegenden Testmittel auch von automatisierten Testskripten höherer Ebenen nutzen. Ein Beispiel: Die Erweiterung der zugrunde liegenden Testmittel um die Aufzeichnung der Start- und Endzeit der Testausführung lässt sich gut auf alle Tests anwenden.

Funktionen der Automatisierung, die das Messen und die Berichterstattung unterstützen
Die Skriptsprachen vieler Testwerkzeuge unterstützen das Messen und Erstellen von Berichten über Mechanismen, mit denen sich Informationen vor, während und nach der Ausführung einzelner Tests sowie Gruppen von Tests und einer kompletten Testsuite aufzeichnen und protokollieren lassen.

Die Erstellung von Berichten zu jedem Testdurchlauf einer Testreihe muss über eine Analysefunktion verfügen, um Ergebnisse früherer Testläufe berücksichtigen und Trends ermitteln zu können (z. B. Änderungen in der Erfolgsquote der Tests).

Die Automatisierung des Testens erfordert typischerweise eine Automatisierung sowohl der Testausführung als auch der Testverifizierung. Letzteres wird durch den Vergleich bestimmter Elemente des Testergebnisses mit einem vordefinierten erwarteten Ergebnis erreicht. Dieser Vergleich wird im Allgemeinen am besten von einem Testwerkzeug vorgenommen. Dabei ist die Informationsebene zu berücksichtigen, die im Ergebnis dieses Vergleichs protokolliert wird. Wichtig ist, dass der Status des Tests richtig ermittelt wird (z. B. bestanden, fehlgeschlagen). Beim Status „Fehlgeschlagen“ werden mehr Informationen über die Ursache des Fehlschlagens benötigt (z. B. Screenshots).

Die Unterscheidung zwischen erwarteten Abweichungen im tatsächlichen und dem erwarteten Ergebnis eines Tests ist nicht immer trivial. Werkzeuge können dabei helfen, Vergleiche zu definieren, bei denen erwartete Abweichungen (z. B. Datumsangaben und Uhrzeiten) ignoriert, nicht erwartete Abweichungen hingegen gemeldet werden.

Integration mit Werkzeugen von Drittanbietern (Kalkulationstabellen, XML, Dokumenten, Datenbanken, Berichtswerkzeugen usw.)
Wenn Informationen aus der Ausführung automatisierter Testfälle in anderen Werkzeugen für die Rückverfolgung und Berichterstattung verwendet werden (z. B. durch die Aktualisierung einer Rückverfolgbarkeitsmatrix), können diese Informationen in einem Format bereitgestellt werden, das auch von Werkzeugen von Drittanbietern genutzt werden kann. Das kann über vorhandene Funktionen der Testwerkzeuge (wie z.B. durch Exportformate für die Berichterstattung) oder über eine eigens definierte Berichterstattung erfolgen, die in einem Format ausgegeben wird, das mit anderen Programmen kompatibel ist (z. B. „.xls“ für Excel, „.doc“ für Word, „.html“ für Browser usw.).

Visualisierung von Ergebnissen (Dashboards, Diagramme, Graphen usw.)
Testergebnisse sollten grafisch aufbereitet werden. Man sollte beispielsweise überlegen, Probleme in der Testausführung farblich zu signalisieren. So ließe sich der Fortschritt der Testausführung/-automatisierung z. B. in Form einer Ampel veranschaulichen. Auf diese Weise können Entscheidungen auf Basis gemeldeter Informationen getroffen werden. Vor allem das Management ist an grafischen Zusammenfassungen interessiert, um das Testergebnis auf einen Blick erfassen zu können. Falls weiterführende Informationen benötigt werden, können sie immer noch tiefer in die Details eintauchen.

5.3 Protokollierung von TAS und SUT

Protokollierung ist in der TAS sehr wichtig: die Protokollierung der Testautomatisierung selbst und die Protokollierung des SUT. Testprotokolle sind eine Quelle, die häufig zur Analyse potentieller Probleme herangezogen wird. Der folgende Abschnitt enthält nach TAS und SUT kategorisierte Beispiele für die Testprotokollierung.

Die TAS-Protokollierung sollte Folgendes umfassen (ob das TAF oder der Testfall die Informationen protokolliert, spielt dabei eine untergeordnete Rolle und hängt vom Kontext ab):

- Welcher Testfall gerade ausgeführt wird, einschließlich Start- und Endzeit.
- Der Status der Testfallausführung, weil fehlgeschlagene Tests sich zwar einfach in Protokolldateien ermitteln lassen, das Framework selbst aber auch über diese Informationen verfügen und diese über ein Dashboard melden sollte. Der Ausführungsstatus des Testfalls kann „bestanden“, „fehlgeschlagen“ oder „TAS-Fehler“ lauten. Das Ergebnis eines TAS-Fehlers wird in Situationen verwendet, in denen das Problem nicht im SUT liegt.
- Details des Testprotokolls auf hoher Ebene (Protokollierung wichtiger Schritte) einschließlich Zeitangaben.
- Dynamische Informationen über das SUT (z. B. Speicherlecks), die der Testfall mit Hilfe von Fremdwerkzeugen ermitteln konnte. Die tatsächlichen Ergebnisse bzw. das Fehlschlagen dieser dynamischen Messungen sollten zusammen mit dem Testfall protokolliert werden, der bei Entdeckung der Abweichung ausgeführt wurde.
- Im Falle von Zuverlässigkeitstests / Stresstests (bei denen zahlreiche Zyklen durchlaufen werden) sollte ein Wert erfasst werden, mit dem sich einfach ermitteln lässt, wie oft die Testfälle ausgeführt wurden.
- Wenn Testfälle zufällige Elemente aufweisen (z. B. Zufallsparameter oder Zufallsschritte bei Tests mit Zustandsautomaten), müssen die Zufallswerte/-entscheidungen protokolliert werden.
- Alle Aktionen, die ein Testfall ausführt, müssen so protokolliert werden, dass die Protokolldatei (oder Teile von ihr) wiedergegeben werden kann, damit die Tests mit exakt denselben Schritten und demselben Timing erneut ausgeführt werden können. Damit lassen sich die Reproduzierbarkeit eines ermittelten Fehlers prüfen und Zusatzinformationen erfassen. Die Informationen zu den Aktionen des Testfalls können auch im SUT selbst protokolliert werden. Diese Protokolle können zum Reproduzieren von Problemen, die vom Kunden ermittelt wurden, verwendet werden. Ein typisches Szenario wäre, dass ein Kunde einen Fehler meldet. Die dabei erfassten Protokollinformationen können dann vom Entwicklungsteam bei der Untersuchung des Problems analysiert werden.
- Screenshots und andere visuelle Dokumentationsarten können während der Testausführung zur späteren Verwendung bei der Fehleranalyse gespeichert werden.
- Wenn ein Testfall auf einen Fehler stößt, muss die TAS sicherstellen, dass alle Informationen, die für die Analyse des Problems benötigt werden, gespeichert wurden und verfügbar sind. Das schließt ggf. auch alle Informationen zur Fortsetzung der Tests ein. Zugehörige Crash Dumps und Stack Traces sollten von der TAS an einem sicheren Ort gespeichert werden. An diesen Ort sollten auch alle weiteren Protokolldateien kopiert werden, da sie überschrieben werden könnten, wenn auf dem SUT zyklische Puffer verwendet werden. Dort stehen sie dann für die spätere Analyse zur Verfügung.
- Die Verwendung von Farben kann bei der Unterscheidung der verschiedenen Arten von protokollierten Informationen helfen (z. B. Fehler in Rot, Fortschrittsangaben in Grün).

SUT-Protokollierung:

- Wenn das SUT ein Problem hat, müssen alle für die Analyse dieses Problems benötigten Informationen protokolliert werden, darunter die Datums- und Uhrzeitstempel, der Quellstandort des Problems, Fehlermeldungen usw.
- Das SUT kann sämtliche Benutzerinteraktionen direkt über die verfügbare UI, aber auch über Netzwerkschnittstellen usw. protokollieren. Auf diese Weise können vom Kunden ermittelte Probleme richtig

analysiert werden und die Entwicklung das Problem möglicherweise reproduzieren.

- Bei Inbetriebnahme des Systems sollten Konfigurationsdaten in einer Datei protokolliert werden: die verschiedenen Software-/Firmwareversionen, die Konfiguration des SUT, die Konfiguration des Betriebssystems usw.

Sämtliche Protokolldaten müssen einfach durchsuchbar sein. Ein von der TAS in der Protokolldatei erkanntes Problem sollte in der Protokolldatei des SUT leicht identifiziert werden können und umgekehrt. Dies sollte unabhängig davon möglich sein, ob zusätzliche Werkzeuge verwendet werden. Die Synchronisierung verschiedener Protokolle mit einem Zeitstempel erleichtert die Zuordnung der Ereignisse bei Meldung eines Fehlers.

5.4 Erstellung von Berichten zur Testautomatisierung

Die Testprotokolle enthalten detaillierte Informationen zu den Ausführungsschritten, Aktionen und Reaktionen eines Testfalls bzw. einer Testsuite. Sie allein liefern jedoch keinen guten Überblick über das gesamte Ausführungsergebnis. Dafür ist es eine Berichterstellungsfunktion erforderlich. Nach jeder Ausführung der Testsuite muss ein präziser Bericht erstellt und veröffentlicht werden. Zur Erzeugung dieser Berichte kann ein wiederverwendbarer Generator verwendet werden.

Inhalt der Berichte
Der Testausführungsbericht muss eine Zusammenfassung enthalten, die einen Überblick über die Ergebnisse der Ausführung, das getestete System und die Umgebung gibt, in der die Tests ausgeführt wurden. Mit dieser Zusammenfassung müssen alle Beteiligten etwas anfangen können.

Im Bericht muss angegeben sein, welche Tests fehlgeschlagen sind und warum. Um die Fehlersuche und Fehlerbeseitigung zu vereinfachen, müssen der Verlauf der Testausführung und die verantwortliche Person für den Test bekannt sein. Die verantwortliche Person ist in der Regel die Person, die ihn erstellt oder zuletzt aktualisiert hat. Sie muss die Ursache des Fehlschlagens untersuchen, die zugehörigen Probleme melden, deren Behebung nachverfolgen und prüfen, ob die Behebung richtig umgesetzt wurde.

Die Berichterstattung dient auch der Diagnose von Fehlern der TAF-Komponenten (siehe dazu Kap. 7).

Veröffentlichen der Berichte
Der Bericht sollte für jeden verfügbar sein, der Interesse an den Ausführungsergebnissen hat. Dazu kann er auf eine Website hochgeladen, über eine Mailingliste verschickt oder in ein anderes Werkzeug (z. B. ein Testmanagementwerkzeug) exportiert werden. Aus praktischer Sicht ist es am wahrscheinlichsten, dass die am Ausführungsergebnis interessierten Personen den Bericht lesen und analysieren, wenn sie einen Bezugsmechanismus dafür haben oder den Bericht per E-Mail beziehen können.

Es besteht die Möglichkeit, zur Ermittlung problematischer Teile des SUT eine Berichtshistorie zu führen, um so statistische Zahlen über Testfälle oder Testsuites mit häufigen Regressionen zu erfassen.

6. Überführung des manuellen Testens in eine automatisierte Umgebung – 120 min

Begriffe
Fehlernachtest, Regressionstest

Lernziele für „Überführung des manuellen Testens in eine automatisierte Umgebung“

6.1 Kriterien für die Automatisierung
ALTA-E-6.1.1 (K3) Ermitteln von Kriterien bezüglich der Eignung von Tests für die Automatisierung
ALTA-E-6.1.2 (K2) Verstehen der Faktoren für die Umstellung vom manuellen auf das automatisierte Testen

6.2 Erforderliche Schritte zur Automatisierung von Regressionstests
ALTA-E-6.2.1 (K2) Erläutern der Faktoren, die bei der Implementierung automatisierter Regressionstests zu berücksichtigen sind

6.3 Faktoren bei der Automatisierung des Testens neuer Funktionen
ALTA-E-6.3.1 (K2) Erläutern der Faktoren, die beim Automatisieren von Tests für neue Funktionen zu berücksichtigen sind

6.4 Bei der Automatisierung von Fehlernachtests zu berücksichtigende Faktoren
ALTA-E-6.4.1 (K2) Erläutern der Faktoren, die bei der Implementierung automatisierter Fehlernachtests zu berücksichtigen sind

6.1 Kriterien für die Automatisierung

Traditionell haben Unternehmen meist nur manuelle Testfälle entwickelt. Entschließt man sich, auf eine automatisierte Testumgebung umzustellen, muss man den aktuellen Zustand der manuellen Tests bewerten und den effektivsten Ansatz für die Automatisierung dieser Testressourcen ermitteln. Die vorhandene Struktur eines manuellen Tests kann für die Automatisierung geeignet sein oder nicht. In diesem Fall kann ein vollständiges Neuschreiben des Tests zur Unterstützung der Automatisierung erforderlich sein. Eventuell können relevante Komponenten vorhandener manueller Tests (wie z. B. Eingabewerte, erwartete Ergebnisse, Navigationspfade) Tests extrahiert und für die Automatisierung wiederverwendet werden. Um die Umstellung auf eine Automatisierung zu erleichtern, sollte bereits die manuelle Teststrategie so umgesetzt werden, dass sie durch ihre Struktur eine spätere Automatisierung berücksichtigt.

Nicht alle Tests können oder müssen automatisiert werden. Mitunter ist auch die erste Iteration eines Tests manuell. Daher gibt es zwei Aspekte, die bei der Umstellung zu berücksichtigen sind: die Erstumwandlung manueller Tests in automatisierte Tests und die anschließende Umstellung neuer manueller Tests auf Automatisierung.

Man sollte auch beachten, dass bestimmte Testarten nur in automatisierter Form (wirkungsvoll) ausgeführt werden können. Das gilt z. B. für Zuverlässigkeitstests, Stresstests und Performanztests.

Durch Testautomatisierung ist es möglich, Anwendungen und Systeme ohne UI zu testen. In diesem Fall kann das Testen über Schnittstellen in der Software auf der Integrationsebene erfolgen. Diese Arten von Testfällen können zwar auch manuell ausgeführt werden (mittels von Hand eingegebener Befehle zur Ansteuerung der Schnittstellen), dies ist in der Regel aber unpraktisch. Die Automatisierung ermöglicht es, Nachrichten und Anweisungen in einer Abarbeitungsreihenfolge zu organisieren. Dadurch kann das Testen eher beginnen und Fehler können bereits entdeckt werden, wenn ein manuelles Testen noch nicht möglich ist.

Vor dem Starten der Testautomatisierung ist zu prüfen, ob das Erstellen automatisierter Tests im Vergleich zu manuellen Tests sinnvoll und durchführbar ist. Zu den Eignungskriterien zählen u. a.:

- Verwendungshäufigkeit
- Komplexität der Automatisierung
- Werkzeugunterstützung
- Reifegrad des Testprozesses
- Eignung der Automatisierung für die Phase des Softwareprodukt-Lebenszyklus
- Nachhaltigkeit der automatisierten Umgebung
- Steuerbarkeit des SUT

Jeder dieser Punkte wird nachstehend ausführlich erläutert.

Verwendungshäufigkeit
Wie oft ein Test ausgeführt werden muss, ist eine Überlegung im Zusammenhang mit der Durchführbarkeit einer Automatisierung. Tests, die im Rahmen eines Major- oder Minor-Release-Zyklus regelmäßiger durchgeführt werden, sind bessere Automatisierungskandidaten, weil sie häufiger verwendet werden. Als allgemeine Regel gilt: Je größer die Anzahl der Anwendungs-Releases und der damit verbundenen Testzyklen, desto größer ist der Nutzwert automatisierter Tests.

Wenn funktionale Tests automatisiert werden, können sie in späteren Releases für die Regressionstests verwendet werden. Automatisierte Tests, die für Regressionstests verwendet werden, bieten eine hohe Investitionsrendite (ROI) und Risikominderung für die bestehende Codebasis.

Wenn ein Testskript einmal im Jahr ausgeführt wird und sich das SUT innerhalb dieses Jahres ändert, ist die Erstellung eines automatisierten Tests gegebenenfalls nicht praktikabel oder effizient. Die Zeit, die für die jährliche Anpassung des Tests an das SUT benötigt wird, sollte dann besser in das manuelle Testen fließen.

Komplexität der Automatisierung
Wenn ein komplexes System getestet werden muss, kann die Automatisierung von enormem Vorteil sein, um dem manuellen Tester die schwierige Aufgabe zu ersparen, komplexe Schritte wiederholen zu müssen, die langwierig, zeitaufwendig und fehleranfällig sind.

Bestimmte Testskripte lassen sich gegebenenfalls nur schwer oder zu hohen Kosten automatisieren. Es gibt eine Reihe von Faktoren, die Einfluss darauf haben:

- ein SUT, das nicht kompatibel mit bestehenden, verfügbaren, automatisierten Testlösungen ist
- die Anforderung, umfangreichen Programmcode zu erstellen und Aufrufe an APIs für die Automatisierung zu entwickeln
- die Vielfalt von Systemen, die im Rahmen der Ausführung eines Tests adressiert werden müssen
- die Interaktion mit externen Schnittstellen und/oder proprietären Systemen
- bestimmte Aspekte von Benutzbarkeitstests
- der Zeitaufwand für das Testen der Automatisierungsskripte usw.

Werkzeugunterstützung
Es gibt eine Vielzahl von Plattformen für die Entwicklung von Anwendungen. Der Tester muss wissen, welche Werkzeuge verfügbar sind, um alle möglichen Plattformen zu unterstützen und in welchem Umfang die verwendete Plattform unterstützt wird. Unternehmen nutzen eine Vielzahl von Testwerkzeugen, darunter z. B. Werkzeuge von kommerziellen Anbietern, Open-Source-Werkzeuge und selbst entwickelte Werkzeuge. Jedes Unternehmen hat andere Erfordernisse, was die Ressourcen zur Unterstützung von Testwerkzeugen angeht. Kommerzielle Anbieter bieten in der Regel bezahlten Support an. Die Marktführer haben meist ein Netzwerk mit Experten, die Unterstützung bei der Implementierung von Testwerkzeugen leisten können. Für Open-Source-Werkzeuge erhält man in der Regel Support in Online-Foren, wo Informationen zu finden sind und Fragen gestellt werden können. Für im eigenen Haus entwickelte Testwerkzeuge übernimmt das vorhandene Personal den Support.

Das Thema Werkzeugunterstützung ist nicht zu unterschätzen. Startet man ein Testautomatisierungsprojekt, ohne zu wissen, inwieweit die einzelnen Testwerkzeuge und das SUT miteinander kompatibel sind, kann das

Scheitern vorprogrammiert sein. Selbst wenn sich die meisten Tests für das SUT automatisieren lassen, kann es passieren, dass dies gerade für die kritischen Tests nicht gilt.

Reifegrad des Testprozesses
Damit ein Testprozess wirkungsvoll automatisiert werden kann, muss er strukturiert, regelkonform und wiederholbar sein. Mit der Automatisierung hält ein kompletter Entwicklungsprozess Einzug in den bestehenden Testprozess. Das erfordert das Management des Automatisierungscodes und der zugehörigen Komponenten.

Eignung der Automatisierung für die Phase des Softwareprodukt-Lebenszyklus
Ein SUT hat einen Produktlebenszyklus, der sich über Jahre bis Jahrzehnte erstrecken kann. Mit Beginn der Entwicklung eines Systems wird das System geändert und erweitert, um Fehler auszumerzen und es im Sinne der Erfordernisse seiner Endnutzer zu optimieren. In den frühen Phasen seiner Entwicklung ändert sich das System u. U. so schnell, dass die Implementierung einer automatisierten Testlösung nicht möglich ist. Bildschirm-Layouts und Bedienelemente werden fortwährend optimiert. In einer sich derart dynamisch wandelnden Umgebung kann die Automatisierung eine kontinuierliche Nachbesserung erfordern, was weder effizient noch effektiv ist. Das wäre so, als ob man bei einem fahrenden Auto einen Reifen wechseln wollte. Bei großen Systemen in einer sequenziellen Entwicklungsumgebung ist der beste Zeitpunkt für den Start der Implementierung automatisierter Tests gekommen, wenn sich das System stabilisiert hat und einen Kern an wichtigen Funktionen enthält.

Mit der Zeit erreichen Systeme das Ende ihres Produktlebenszyklus und werden außer Betrieb genommen oder komplett überarbeitet und mit neuerer und effizienterer Technologie ausgerüstet. Bei Systemen mit einem nahenden Ende ihres Lebenszyklus ist die Automatisierung nicht ratsam, da so kurzlebige Unterfangen den Aufwand nicht rechtfertigen. Für Systeme, die unter Verwendung einer anderen Architektur umgestaltet werden, während die vorhandene Funktionalität jedoch erhalten bleibt, ist eine automatisierte Testumgebung, die Datenelemente definiert, in alten und neuen Systemen gleichermaßen nützlich. In diesem Fall wäre die Wiederverwendung von Testdaten möglich und ein Umbau der automatisierten Umgebung in Abstimmung auf die neue Architektur notwendig.

Nachhaltigkeit der Umgebung
Eine Testumgebung für die Automatisierung muss flexibel und an Änderungen anpassbar sein, die mit der Zeit am SUT auftreten. Dafür muss sie Folgendes können: Probleme müssen sich durch Automatisierung schnell diagnostizieren und beheben lassen, Automatisierungskomponenten müssen einfach zu warten sein und es muss die Möglichkeit bestehen, neue Funktionen und Unterstützung in die automatisierte Umgebung einzubinden. Diese Attribute sind integraler Bestandteil des Gesamtentwurfs und der Implementierung der gTAA.

Steuerbarkeit des SUT (Vorbedingungen, Setup und Stabilität)
Der TAE muss Steuerungs- und Visibilitätsmerkmale des SUT ermitteln, die bei der Entwicklung wirkungsvoller, automatisierter Tests helfen. Andernfalls basiert die Testautomatisierungslösung nur auf Interaktionen mit dem UI und ist daher weniger gut wartbar. Siehe dazu Abschnitt 2.3 zur Auslegung auf Testbarkeit und Automatisierung.

Technische Planung zur Unterstützung der ROI-Analyse
Die Testautomatisierung kann dem Testteam ein unterschiedliches Maß an Nutzen bieten. Mit der Implementierung einer wirkungsvollen automatisierten Testlösung geht jedoch stets ein erheblicher (finanzieller und zeitlicher) Aufwand einher. Bevor man diesen Aufwand für die Entwicklung automatisierter Tests auf sich nimmt, muss analysiert und bewertet werden, welchen potentiellen Gesamtnutzen die Implementierung der Testautomatisierung haben kann. Wenn dieser ermittelt wurde, müssen Maßnahmen zur Umsetzung eines solchen Plans definiert und die damit einhergehenden Kosten kalkuliert werden, um die Investitionsrendite (ROI) zu berechnen.

Zur angemessenen Vorbereitung der Umstellung auf eine automatisierte Umgebung müssen folgende Punkte berücksichtigt werden:

- Verfügbarkeit von Werkzeugen in der Testumgebung für die Testautomatisierung
- Genauigkeit der Testdaten und Testfälle

- Umfang der Arbeiten zur Testautomatisierung
- Schulung des Testteams im Hinblick auf den Paradigmenwechsel
- Rollen und Zuständigkeiten
- Kooperation zwischen den Entwicklungsteams und den Testautomatisierungsteams
- paralleles Arbeiten
- Erstellung von Berichten zur Testautomatisierung

Verfügbarkeit von Werkzeugen in der Testumgebung für die Testautomatisierung
Es müssen ausgewählte Testwerkzeuge installiert und auf Funktionsfähigkeit in der Testlaborumgebung geprüft werden. Das kann das Herunterladen von Service Packs oder Release Updates, die Auswahl der richtigen Installationskonfiguration – einschließlich Add-Ins – für die Unterstützung des SUTs und die Gewährleistung der ordnungsgemäßen Funktion der TAS in der Testlaborumgebung, sowie der Umgebung für die Automatisierungsentwicklung einschließen.

Genauigkeit der Testdaten und Testfälle
Die Genauigkeit und Vollständigkeit manueller Testdaten und Testfälle ist Voraussetzung dafür, dass deren Nutzung in der automatisierten Umgebung vorhersagbare Ergebnisse liefert. Automatisierte Tests benötigen explizite Daten zur Eingabe, Navigation, Synchronisierung und Validierung.

Umfang der Arbeiten zur Testautomatisierung
Damit sich frühe Erfolge zeigen und Feedback zu technischen Fragen mit Auswirkungen auf den Fortschritt gesammelt werden kann, sollte die Automatisierung mit einem begrenzten Umfang begonnen werden. Das erleichtert zukünftige Automatisierungsaufgaben. Ein Pilotprojekt kann auf einen Bereich des Funktionsumfangs des Systems abzielen, der repräsentativ für die gesamte Interoperabilität des Systems ist. Die Erkenntnisse aus dem Pilotprojekt helfen bei der genaueren Schätzung des künftigen, zeitlichen Aufwands und der entsprechenden terminlichen Planung sowie der Ermittlung von Bereichen, für die spezialisierte, technische Ressourcen benötigt werden. Ein Pilotprojekt bietet einen schnellen Weg, den raschen Erfolg der Automatisierung zu demonstrieren und sichert die weitere Unterstützung durch das Management.

Dazu sollten die zu automatisierenden Testfälle jedoch mit Umsicht ausgewählt werden. Wählen Sie Fälle mit kleinem Automatisierungsaufwand aber hohem Mehrwert. So bietet es sich beispielsweise an, automatische Regressions- oder Smoke-Tests zu implementieren. Sie liefern einen spürbaren Mehrwert, weil diese Tests in der Regel häufig, mitunter täglich ausgeführt werden. Ein weiterer guter Kandidat, um mit der Automatisierung zu beginnen, sind Zuverlässigkeitstests. Diese Tests bestehen häufig aus mehreren Schritten, werden immer wieder ausgeführt und spüren Probleme auf, die sich mit manuellen Tests nur schwer finden lassen. Zuverlässigkeitstests sind einfach zu implementieren, machen sich aber rasch bezahlt.

Diese Pilotprojekte rücken die Automatisierung dadurch ins Rampenlicht, dass manueller Testaufwand gespart oder ernste Probleme entdeckt werden und ebnen so den Weg für eine künftige Ausweitung, sei es durch die Investition von Aufwand oder Geld.

Darüber hinaus sollten Tests Vorrang erhalten, die anfänglich den größten Nutzen haben und daher für das Unternehmen von entscheidender Bedeutung sind. In diesem Zusammenhang ist es wichtig, die technisch komplexesten Tests aus dem Pilotprojekt auszuklammern, da bei ihnen der Automatisierungsaufwand in Bezug auf die vorzeigbaren Ergebnisse meist sehr hoch ist. In der Regel wird die Identifizierung von Tests, deren Merkmale große Teile der Anwendung abdecken, den erforderlichen Impuls liefern, um den Automatisierungsaufwand zu rechtfertigen.

Schulung des Testteams im Hinblick auf den Paradigmenwechsel
Es gibt unterschiedliche Arten von Testern: Manche können Fachexperten aus der Endbenutzer-Community oder Wirtschaftsanalytiker sein, andere wiederum haben ausgeprägte technische Kompetenzen, dank derer sie die zugrunde liegende Systemarchitektur besser verstehen. Für ein wirkungsvolles Testen sollte ein breites Kompetenzspektrum vertreten sein. Wenn das Testteam auf Automatisierung umstellt, werden die Rollen spezialisierter. Damit die Automatisierung ein Erfolg wird, muss die Zusammenstellung des Testteams geändert werden. Das Team ist frühzeitig über die beabsichtigte Umstellung aufzuklären, um Befürchtungen zu

zerstreuen, die sich im Hinblick auf Rollen oder die Gefahr, überflüssig zu werden, ergeben. Wenn dieser Prozess richtig gesteuert wird, werden alle Mitglieder des Testteams positiv an die Umstellung auf Automatisierung herangehen und sich konstruktiv in den damit einhergehenden organisatorischen und technischen Wandel einbringen.

Rollen und Zuständigkeiten
Die Testautomatisierung muss eine Aktivität sein, bei der jeder einen Beitrag leisten kann. Das heißt jedoch nicht, dass jeder dieselbe Rolle hat. Die Entwicklung, Implementierung und Wartung einer automatisierten Testumgebung ist technisch geprägt und sollte daher Mitarbeitern mit guten Programmierkenntnissen und technischem Hintergrund vorbehalten sein. Das Ergebnis der Entwicklung einer automatisierten Testumgebung muss eine Umgebung sein, die von technisch versierten Personen und technischen Laien gleichermaßen genutzt werden kann. Zur Maximierung des Werts einer automatisierten Testumgebung werden Personen mit Fachwissen und Kompetenzen im Bereich des Testens benötigt. Das ist die Voraussetzung für die Entwicklung geeigneter Testskripte (einschließlich der entsprechenden Testdaten). Diese werden genutzt, um die Testautomatisierungsumgebung weiterzuentwickeln und die angestrebte Testüberdeckung zu erreichen. Experten für den Geschäftsbereich prüfen Berichte, um den Funktionsumfang der Anwendung zu bestätigen. Technische Experten stellen ihrerseits sicher, dass die automatisierte Umgebung richtig und effizient funktioniert. Diese technischen Experten können auch Entwickler mit Interesse am Testen sein. Erfahrung in der Softwareentwicklung ist Voraussetzung für das Konzipieren von wartbarer Software. Wartbarkeit ist bei der Testautomatisierung von zentraler Bedeutung. Entwickler können sich auf das Testautomatisierungsframework oder die Testbibliotheken konzentrieren. Die Implementierung der Testfälle sollte den Testern vorbehalten sein.

Kooperation zwischen den Entwicklungsteams und den Testautomatisierungsteams
Eine erfolgreiche Testautomatisierung setzt sowohl die Einbindung des Softwareentwicklungsteams als auch der Tester voraus. Entwickler und Tester müssen bei der Testautomatisierung eng kooperieren, damit die Entwickler Support und technische Informationen zu ihren Entwicklungsmethoden und -werkzeugen bereitstellen können. Die TAEs können Bedenken im Hinblick auf die Testbarkeit von Systementwürfen und Entwicklercode aufwerfen – vor allem dann, wenn Standards nicht eingehalten werden oder Entwickler ausgefallene, selbst entwickelte oder sogar neue Bibliotheken/Objekte verwenden. Denkbar wäre z. B., dass Entwickler eine GUI-Steuerung von einem Fremdanbieter verwenden, die nicht mit dem gewählten Automatisierungswerkzeug kompatibel ist.

Letztlich muss das Projektmanagementteam eines Unternehmens ein klares Verständnis haben, welche Rollen und Zuständigkeiten für eine erfolgreiche Automatisierung benötigt werden.

Paralleles Arbeiten
Im Rahmen der Umstellungsaktivitäten bilden viele Unternehmen ein weiteres Team, um den Prozess, manuelle Tests in automatisierte Tests zu überführen, parallel zu beginnen. Die neuen, automatisierten Skripte werden dann in die bestehenden Tests eingebunden und ersetzen die manuellen Skripts. Zuvor ist es jedoch häufig ratsam, sich davon zu überzeugen, dass ein automatisiertes Skript denselben Test wie das ersetzte manuelle Skript durchführt.

In vielen Fällen erfolgt vor der Umstellung auf Automatisierung eine Bewertung der manuellen Skripts. Bei dieser Bewertung kann festgestellt werden, dass bestehende manuelle Testskripte umstrukturiert werden müssen, damit die automatisierte Variante effizienter und wirkungsvoller arbeitet.

Berichterstattung über die Automatisierung
Es gibt verschiedene Berichte, die von einer TAS automatisch erzeugt werden können: Bestanden-/Fehlgeschlagen-Status einzelner Skripte oder Schritte innerhalb eines Skripts, Statistiken zur Testausführung und Gesamt-Performanz der TAS. Ebenfalls wichtig ist es, Einblick in den ordnungsgemäßen Betrieb der TAS zu haben, damit sicher ist, dass gemeldete, anwendungsspezifische Ergebnisse präzise und vollständig sind (siehe Kapitel 7: Verifizieren der TAS).

6.2 Erforderliche Schritte zur Automatisierung von Regressionstests

Regressionstests bieten sich grundsätzlich für die Automatisierung an. Der Bestand an Regressionstests wächst, weil die funktionalen Tests von heute die Regressionstests von morgen werden. Es ist also nur eine Frage der Zeit, wann die Zahl der Regressionstests den verfügbaren Zeit- und Ressourcenrahmen eines traditionellen, manuellen Testteams sprengt.

Bei der Entwicklung von Schritten zur Vorbereitung der Automatisierung von Regressionstests muss eine Reihe von Fragen gestellt werden:

- Wie oft sollen die Tests ausgeführt werden?
- Welche Ausführungszeit haben die einzelnen Regressionstests und die gesamte Testsuite?
- Gibt es funktionale Überschneidungen zwischen den Tests?
- Nutzen Tests dieselben Daten?
- Hängen die Tests voneinander ab?
- Welche Vorbedingungen müssen vor der Testausführung erfüllt sein?
- Wie hoch ist die Testüberdeckung des SUT in Prozent?
- Werden die Tests gegenwärtig fehlerfrei ausgeführt?
- Was muss passieren, wenn Regressionstests zu lange dauern?

Jeder dieser Punkte wird nachstehend ausführlich erläutert.

Häufigkeit der Testausführung
Tests, die häufig im Rahmen von Regressionstests ausgeführt werden, sind die besten Kandidaten für eine Automatisierung. Diese Tests wurden bereits entwickelt, führen bekannte SUT-Funktionen aus und profitieren von der Automatisierung mit einer deutlich geringeren Ausführungszeit.

Testausführungszeit
Die Ausführungszeit, um einen Test oder eine komplette Testsuite auszuführen, ist ein wichtiger Parameter für die Bewertung des Nutzens der Automatisierung von Regressionstests. Eine Herangehensweise besteht darin, mit der Automatisierung zeitaufwändiger Tests zu beginnen. Das ermöglicht eine schnellere und effizientere Ausführung von Tests und gleichzeitig die Erhöhung der Anzahl von Ausführungszyklen automatisierter Regressionstests. Der Vorteil liegt in zusätzlichem und häufigerem Feedback zur SUT-Qualität und einem geringeren Verteilungsrisiko.

Funktionsüberschneidung
Bei der Automatisierung bestehender Regressionstests ist es eine bewährte Praxis, funktionale Überschneidungen zwischen Testfällen zu ermitteln und in den äquivalenten, automatisierten Tests nach Möglichkeit zu verringern. Das verkürzt die Ausführungsdauer automatisierter Tests, was bei wachsender Menge ausgeführter Tests von Bedeutung ist. Tests, die unter Verwendung von Automatisierung entwickelt wurden, nehmen häufig eine neue Struktur an, weil sie auf wiederverwendbare Komponenten und gemeinsam genutzte Daten-Repositories zugreifen. Es ist nicht ungewöhnlich, bestehende manuelle Tests in mehrere kleine automatisierte Tests zu zerlegen. Auch die Zusammenführung mehrerer manueller Tests zu einem größeren automatisierten Test kann sich anbieten. Manuelle Tests müssen einzeln und als Gruppe bewertet werden, damit eine wirkungsvolle Umsetzungsstrategie entwickelt werden kann.

Gemeinsame Datennutzung
Tests nutzen häufig dieselben Daten. Das kann der Fall sein, wenn Tests denselben Datensatz für die Ausführung verschiedener SUT-Funktionen nutzen. Ein Beispiel: Testfall „A“ überprüft die verfügbare Urlaubszeit eines Mitarbeiters, während Testfall „B“ überprüfen könnte, an welchen Weiterbildungsmaßnahmen der Mitarbeiter im Rahmen seiner beruflichen Entwicklungsziele teilgenommen hat. Beide Testfälle nutzen dieselben Mitarbeiterdaten, überprüfen aber unterschiedliche Parameter. In einer manuellen Testumgebung würden die Mitarbeiterdaten in der Regel bei jedem manuellen Testfall, der diese Mitarbeiterdaten prüft, viele Male dupliziert werden. Bei einem automatisierten Test sollten die gemeinsam genutzten Daten, sofern das möglich und durchführbar

ist, an einem Ort gespeichert und von dort abgerufen werden, um Duplikation oder das Einschleusen von Fehlern zu vermeiden.

Gegenseitige Abhängigkeit von Tests
Bei der Ausführung komplexer Regressionstestszenarios kann ein Test von einem anderen oder mehreren anderen Tests abhängig sein. Das tritt relativ häufig auf und kann z. B. das Ergebnis einer neuen „Bestell-ID“ sein, die als Ergebnis eines Testschritts erzeugt wird. Bei nachfolgenden Tests wird geprüft, ob: a) die neue Bestellung richtig im System angezeigt wird, b) Änderungen an der Bestellung möglich sind, bzw. c) das Löschen der Bestellung erfolgreich ist. In jedem Fall muss der Wert „Bestell-ID“, der im ersten Test dynamisch erstellt wird, zur Wiederverwendung durch spätere Tests erfasst werden. In Abhängigkeit vom Aufbau der TAS kann dies angesprochen werden.

Vorbedingungen für Tests
Häufig kann ein Test erst ausgeführt werden, wenn bestimmte Ausgangsbedingungen gegeben sind. Das kann die Wahl der richtigen Datenbank oder der Testdaten, auf deren Basis getestet werden soll oder das Festlegen von Anfangswerten bzw. -parametern einschließen. Viele dieser Initialisierungsschritte, die für die Herstellung der Vorbedingungen für einen Test nötig sind, lassen sich automatisieren. Das ermöglicht eine zuverlässigere und belastbarere Lösung, wenn diese Schritte vor Ausführung der Tests nicht übersprungen werden dürfen. Wenn Regressionstests automatisiert werden, müssen diese Vorbedingungen Bestandteil des Automatisierungsprozesses sein.

SUT-Überdeckung
Bei jeder Ausführung von Tests wird ein Teil der Funktionen eines SUT ausgeführt. Zur Gewährleistung der allgemeinen Qualität des SUT müssen Tests entworfen werden, die das SUT in möglichst großer Breite und Tiefe abdecken. Darüber hinaus kann mit Codeüberdeckungs-Werkzeugen die Ausführung automatisierter Tests überwacht werden, um deren Wirksamkeit zu quantifizieren. Bei automatisierten Regressionstests können wir erwarten, dass zusätzliche Tests mit der Zeit für eine zusätzliche Überdeckung sorgen. Dies zu messen, bildet ein wirksames Mittel, den Wert der Tests selbst zu quantifizieren.

Ausführbare Tests
Vor der Umwandlung eines manuellen Regressionstests in einen automatisierten Test muss überprüft werden, ob der manuelle Test richtig funktioniert. Das bildet die Voraussetzung für eine erfolgreiche Umwandlung in einen automatisierten Regressionstest. Wird ein manueller Test nicht richtig ausgeführt, weil er z. B. schlecht geschrieben ist, ungültige Daten verwendet, veraltet oder nicht mehr synchron mit dem aktuellen SUT ist oder weil das SUT einen Fehler aufweist und dennoch in einen automatisierten Test umgewandelt, ohne vorher die Fehlerursache zu kennen bzw. zu beheben, kommt ggf. ein nicht funktionierender automatisierter Test heraus. Das ist nutzlos und eine Verschwendung von Ressourcen.

Große Regressionstestsuites
Die Menge an Regressionstests für ein SUT kann ziemlich groß werden – so groß, dass die Tests über Nacht oder das Wochenende nicht vollständig ausgeführt werden können. In diesem Fall ist die parallele Ausführung von Testfällen eine Option, sofern mehrere SUTs verfügbar sind (bei PC-Anwendungen dürfte das wahrscheinlich kein Problem sein, wenn das SUT ein Flugzeug oder eine Weltraumrakete ist, sieht die Sache hingegen anders aus). SUTs können knapp und/oder teuer sein, was die parallele Ausführung unrealistisch macht. In diesem Fall wäre es eine Option, nur Teile des Regressionstests auszuführen. Nach einiger Zeit (Wochen) ist die Ausführung aller Tests dann abgeschlossen. Die Entscheidung, welcher Teil der Regressionstestsuite ausgeführt werden soll, kann auch auf eine Risikoanalyse gestützt werden (welche Teile des SUT wurden in jüngster Zeit geändert?).

6.3 Faktoren bei der Automatisierung des Testens neuer Funktionen

Oftmals ist es einfacher, Testfälle für neue Funktionen zu automatisieren, deren Implementierung noch nicht abgeschlossen ist (oder besser: noch gar nicht begonnen hat). Der Testentwickler kann den Entwicklern und Architekten somit erläutern, was in den neuen Funktionen benötigt wird, damit sie von der

Testautomatisierungslösung wirksam und effizient getestet werden können.

Wenn ein SUT um neue Funktionen erweitert wird, müssen Tester neue Tests für diese neuen Funktionen und die entsprechenden Anforderungen entwickeln. Der TAE muss Feedback von den Testentwicklern mit Spezialwissen einholen und ermitteln, ob die aktuelle TAS den Erfordernissen der neuen Funktionen genügt. Diese Analyse schließt u. a. den bislang genutzten Ansatz, Entwicklungswerkzeuge von Drittanbietern, verwendete Testwerkzeuge usw. ein.

Änderungen an der TAS müssen mit den vorhandenen automatisierten Testmitteln geprüft werden, damit Änderungen oder Ergänzungen vollständig dokumentiert sind und das Verhalten (oder die Leistung) bestehender TAS-Funktionen nicht beeinträchtigen.

Wird eine neue Funktion mit beispielsweise einer neuen Objektklasse implementiert, kann es erforderlich sein, Aktualisierungen oder Ergänzungen an den Testmitteln vorzunehmen. Darüber hinaus müssen die Kompatibilität mit bestehenden Testwerkzeugen und ggf. alternative Lösungen ermittelt werden. Bei Verwendung eines schlüsselwortgetriebenen Ansatzes kann es erforderlich sein, zusätzliche Schlüsselwörter zu entwickeln oder bestehende Schlüsselwörter zu modifizieren bzw. zu erweitern, um die neuen Funktionen zu unterstützen.

Es kann beispielsweise die Anforderung geben, zusätzliche Testwerkzeuge zu evaluieren, um die neue Umgebung zu unterstützen, in der sich die neuen Funktionen bewegen. So kann ein neues Testwerkzeug erforderlich sein, wenn das bestehende Testwerkzeug lediglich HTML unterstützt.

Neue Testanforderungen können Auswirkungen auf bestehende automatisierte Tests und Testmittel-Komponenten haben. Bevor Änderungen an den automatisierten Test durchgeführt werden, sollten daher die vorhandenen automatisierten Tests am neuen/aktualisierten SUT ausgeführt werden, um Änderungen gegenüber dem früheren Betrieb der bestehenden automatisierten Tests zu ermitteln oder aufzuzeichnen. Das sollte eine Ermittlung und Zuordnung der wechselseitigen Abhängigkeiten mit anderen Tests einschließen. Alle neuen Änderungen an Technologie erfordern eine Evaluierung der aktuellen Testmittel-Komponenten (einschließlich der Testwerkzeuge, Funktionsbibliotheken, APIs usw.) und der Kompatibilität mit der bestehenden TAS.

Wenn sich bestehende Anforderungen ändern, muss der Aufwand für die Aktualisierung von Testfällen, die diese Anforderungen verifizieren, Teil des Projektplans (Projektstrukturplan) sein. Die Rückverfolgbarkeit von Anforderungen zu den Testfällen gibt an, welche Testfälle aktualisiert werden müssen. Diese Aktualisierungen müssen Teil eines Gesamtkonzepts sein.

Zudem muss ermittelt werden, ob die bestehende TAS den Erfordernissen des aktuellen SUT weiterhin genügt. Sind die Implementierungstechniken noch gültig, wird eine neue Architektur benötigt oder lässt sich dies durch Erweiterung der gegenwärtigen Fähigkeiten realisieren?

Bei der Einführung neuer Funktionen haben Testentwickler die Chance, sich davon zu überzeugen, dass die neu definierten Funktionen auch testbar sind. Bereits während der Entwurfsphase sollte das Testen berücksichtigt werden, indem beispielsweise geplant wird, Testschnittstellen bereitzustellen, die von Skriptsprachen oder dem Testautomatisierungswerkzeug zum Überprüfen der neuen Funktionalität verwendet werden können. Siehe dazu Abschnitt 2.3 zur Auslegung auf Testbarkeit und Automatisierung.

6.4 Faktoren bei der Automatisierung von Fehlernachtests

Fehlernachtests erfolgen nach der Behebung eines gemeldeten Fehlers im Quellcode. In der Regel führt ein Tester dazu die erforderlichen Schritte für die Reproduzierung des Fehlers aus, um sich davon zu überzeugen, dass der Fehler nicht mehr vorliegt.

Fehler können sich von Release zu Release fortpflanzen, was möglichweise ein Zeichen für ein Problem mit dem Konfigurationsmanagement ist. Fehlernachtests zählen daher zu den bevorzugten Kandidaten für eine Automatisierung. Durch die Automatisierung kann die Ausführungszeit für Fehlernachtests verkürzt werden. Der Fehlernachtest kann den bestehenden automatisierten Regressionstests hinzugefügt werden und sie ergänzen.

Der automatisierte Fehlernachtest hat in der Regel einen eng begrenzten Funktionsumfang. Die Implementierung kann zu jedem Zeitpunkt erfolgen, sobald ein Fehler gemeldet und die erforderlichen Schritte für seine Reproduzierung ermittelt wurden. Automatisierte Fehlernachtests lassen sich in eine standardmäßige

automatisierte Regressionssuite einbinden oder (wenn praktikabel) unter bestehenden automatisierten Tests zusammenfassen. Bei beiden Ansätzen bleibt der Wert der Automatisierung von Fehlernachtests erhalten.

Die Nachverfolgung automatisierter Fehlernachtests ermöglicht die zusätzliche Erfassung der Zeit und der Anzahl der Zyklen, die für die Behebung von Fehlern aufgewendet wurden.

Neben Fehlernachtests sind Regressionstests erforderlich, um sicherzustellen, dass als Nebenwirkung der Fehlerbehebung keine neuen Fehler eingeschleust wurden. Zur Ermittlung des angemessenen Umfangs der Regressionstests kann eine Auswirkungsanalyse erforderlich sein.

7. Verifizieren der TAS – 120 min

Begriffe
Verifizierung

Lernziele für „Verifizieren der TAS“

7.1 Verifizieren der Komponenten der automatisierten Testumgebung
ALTA-E-7.1.1 (K3) Verifizieren der korrekten Funktion einer automatisierten Testumgebung einschließlich der verwendeten Testwerkzeuge

7.2 Verifizieren der automatisierten Testsuite
ALTA-E-7.2.1 (K3) Verifizieren des richtigen Verhaltens eines automatisierten Testskripts und/oder einer Testsuite

7.1 Verifizieren der Komponenten der automatisierten Testumgebung

Das Testautomatisierungsteam muss prüfen, ob die automatisierte Testumgebung wie vorgesehen arbeitet. Diese Prüfungen erfolgen beispielsweise vor Beginn des automatisierten Testens.

Die Komponenten der automatisierten Testumgebung lassen sich mit einer Reihe von Schritten überprüfen. Jeder dieser Schritte wird nachstehend ausführlich erläutert:

Installation, Einrichtung, Konfiguration und Anpassung von Testwerkzeugen
Die TAS besteht aus vielen Komponenten. Jede Komponente leistet ihren Beitrag zur zuverlässigen und wiederholbaren Ausführung der TAS. Herzstück der TAS sind die ausführbaren Komponenten, die zugehörigen funktionalen Bibliotheken sowie die unterstützenden Daten und Konfigurationsdateien. Das Konfigurieren einer TAS kann von der Verwendung automatisierter Installationsskripte bis zum manuellen Ablegen von Dateien in den entsprechenden Ordnern reichen. Für Testwerkzeuge gibt es genau wie für Betriebssysteme und andere Anwendungen in regelmäßigen Abständen Service Packs bzw. optionale oder obligatorische Add-Ins, um die Kompatibilität mit einer SUT-Umgebung zu gewährleisten.

Die automatisierte Installation (oder das Kopieren) von einem zentralen Repository hat Vorteile. Es kann garantiert werden, dass Tests an unterschiedlichen SUTs mit derselben Version der TAS und ggf. derselben Konfiguration der TAS durchgeführt wurden. Upgrades der TAS können über das Repository erfolgen. Die Nutzung des Repositorys und der Upgrade-Prozess (auf eine neue Version der TAS) sollten identisch mit den standardmäßigen Entwicklungswerkzeugen sein.

Testskripte mit bekannten Bestanden- und Fehlgeschlagen-Ergebnissen
Wenn Testfälle, die eigentlich bestanden sein müssten, fehlschlagen, ist sofort klar, dass ein grundlegendes Problem vorliegt, das so schnell wie möglich behoben werden muss. Umgekehrt gilt: Wenn Testfälle bestanden werden, die eigentlich hätten fehlschlagen müssen, müssen die Komponenten ermittelt werden, die nicht richtig funktionieren. Dazu muss überprüft werden, ob die Protokolldateien und Performanzmetriken richtig erzeugt sowie der Testfall bzw. das Testskript automatisch eingerichtet und außer Betrieb genommen werden. Hilfreich ist auch, einige Tests anderer Typen und Ebenen durchzuführen (funktionale Tests, Performanztests, Komponententests usw.). Das sollte auch auf Framework-Ebene erfolgen.

Wiederholbarkeit der Einrichtung/Außerbetriebnahme der Testumgebung
Eine TAS wird auf unterschiedlichen Systemen und Servern implementiert. Um sicherzustellen, dass die TAS in jeder Umgebung richtig funktioniert, wird ein systematischer Ansatz für das Laden der TAS in eine Umgebung bzw. die Umkehr dieses Vorgangs benötigt. Gelungen ist das, wenn es bei Einrichtung und Neueinrichtung der TAS keinen erkennbaren Unterschied gibt, was ihre umgebungsinternen und -übergreifenden Betriebsmerkmale angeht. Das Konfigurationsmanagement der TAS-Komponenten stellt sicher, dass sich eine gegebene

Konfiguration verlässlich erstellen lässt.

Konfiguration von Testumgebung und -komponenten
Um zu wissen, welche Aspekte der TAS betroffen sind oder Änderungen erfordern, wenn sich die SUT-Umgebung ändert, müssen die verschiedenen Komponenten, aus denen die TAS besteht, bekannt und dokumentiert sein.

Konnektivität gegenüber internen und externen Systemen/Schnittstellen
Wenn eine TAS in einer SUT-Umgebung installiert wurde, muss eine Reihe von Kontrollen erfolgen und Vorbedingungen erfüllt werden, um sicherzustellen, dass die Konnektivität mit internen und externen Systemen, Schnittstellen usw. gegeben ist. Erst dann kann die TAS am SUT ausgeführt werden. Die erfolgreiche Schaffung der Vorbedingungen für die Automatisierung ist ein wichtiger Nachweis dafür, dass die TAS richtig installiert und konfiguriert wurde.

Grad der Intrusion automatisierter Testwerkzeuge
Häufig ist die TAS eng mit dem SUT verzahnt. Konstruktionsbedingt herrscht also eine hohe Kompatibilität, vor allem im Hinblick auf die Interaktionen auf GUI-Ebene. Diese enge Verzahnung kann aber auch negative Effekte haben. Zu nennen wären hier u. a.: Ein SUT verhält sich anders, wenn die TAS innerhalb der SUT-Umgebung residiert; das SUT weist ein anderes Verhalten auf, wenn es manuell genutzt wird; die SUT-Performanz wird beeinträchtigt, wenn sich die TAS in seiner Umgebung befindet oder gegen das SUT ausgeführt wird.

Der Grad der Intrusion variiert mit dem gewählten Ansatz des automatisierten Testens. Zum Beispiel:

- Bei Interaktionen mit dem SUT über externe Schnittstellen, ist der Intrusionsgrad sehr niedrig. Externe Schnittstellen können elektronische Signale (für physische Schalter), USB-Signal für USB-Geräte (wie Tastaturen) usw. sein. Dieser Ansatz bietet die beste Simulation der Handlungen des Endbenutzers. Hier wird die Software des SUT für das Testen überhaupt nicht geändert. Der Testansatz hat keinen Einfluss auf Verhalten und Timing des SUT. Die Interaktion mit dem SUT über derartige Schnittstellen kann sehr komplex sein. Möglicherweise wird dedizierte Hardware benötigt oder für die Erzeugung der Schnittstelle mit dem SUT bedarf es Hardwarebeschreibungssprachen usw. Für reine Softwaresysteme ist das kein typischer Ansatz; bei Produkten mit eingebetteter Software ist dieser Ansatz hingegen durchaus verbreitet.
- Bei Interaktion mit dem SUT über Schnittstellen auf GUI-Ebene wird die SUT-Umgebung angepasst, um UI-Befehle eingeben und die Informationen auslesen zu können, die von den Testfällen benötigt werden. Das Verhalten des SUT wird zwar nicht direkt geändert, aber das Timing wird verändert, was zur Beeinflussung des Verhaltens führen kann. Der Intrusionsgrad ist höher als beim vorherigen Punkt, aber diese Art der schnittstellenbasierten Interaktion mit dem SUT ist weniger komplex. Für diese Form der Automatisierung können handelsübliche Werkzeuge von der Stange genutzt werden.
- Die Interaktion mit dem SUT kann über Testschnittstellen in der Software oder über bestehende Schnittstellen erfolgen, die die Software von Haus aus bietet. Die Verfügbarkeit dieser Schnittstellen (APIs) ist ein wichtiger Aspekt der Auslegung auf Testbarkeit. Der Intrusionsgrad kann in diesem Fall ziemlich hoch sein. Automatisierte Tests nutzen Schnittstellen, die von den Endbenutzern des Systems u. U. gar nicht genutzt werden (Testschnittstellen) oder Schnittstellen, die in einem anderen Kontext als in der Praxis genutzt werden. Andererseits ist es äußerst einfach und kostengünstig, automatisierte Tests über Schnittstellen (API) durchzuführen. Das Testen des SUT über Testschnittstellen kann ein solider Ansatz sein, solange man sich der potentiellen Risiken bewusst ist.

Ein hoher Intrusionsgrad kann beim Testen zu Fehlern führen, die sich unter den Nutzungsbedingungen der Praxis nicht zeigen würden. Wenn dies zum Fehlschlagen automatisierter Tests führt, kann das Vertrauen in die Testautomatisierungslösung drastisch sinken. Entwickler können vorgeben, dass Fehler, die durch automatisiertes Testen ermittelt wurden, zunächst nach Möglichkeit manuell reproduziert werden müssen, um die Analyse zu unterstützen.

Testen der Framework-Komponenten
Wie bei jedem anderen Softwareentwicklungsprojekt müssen die automatisierten Framework-Komponenten einzeln getestet und verifiziert werden. Das kann funktionale und nichtfunktionale Tests (Performanz, Ressourcennutzung, Benutzbarkeit usw.) einschließen.

So müssen z. B. Komponenten, die eine Objektverifizierung auf GUI-Systemen ermöglichen, auf ein breites Spektrum von Objektklassen getestet werden, um sicherzustellen, dass die Objektverifizierung richtig funktioniert. Analog dazu müssen Fehlerprotokolle und -berichte präzise Angaben zum Status der Automatisierung und des SUT-Verhaltens liefern.

Bei nichtfunktionalen Tests können beispielsweise die Verschlechterungen in der Leistung des Frameworks sowie die Auslastung der Systemressourcen ermittelt werden, die Anzeichen für Probleme wie Speicherlecks sein können. Ein weiteres Beispiel ist die Interoperabilität von Komponenten inner- und/oder außerhalb des Frameworks.

7.2 Verifizieren der automatisierten Testsuite

Automatisierte Testsuiten müssen auf Vollständigkeit, Konsistenz und richtiges Verhalten getestet werden. Mit verschiedenen Arten von Verifizierungsprüfungen lässt sich sicherstellen, dass die automatisierte Testsuite zu jedem Zeitpunkt läuft bzw. einsatzfähig ist.

Eine automatisierte Testsuite lässt sich mit einer Reihe von Schritten überprüfen. Dazu zählen:

- Ausführung von Testskripten mit bekannten Bestanden- und Fehlgeschlagen-Ergebnissen
- Prüfen der Testsuite
- Verifizieren neuer Tests mit Fokus auf neuen Funktionen des Frameworks
- Gewährleisten der Wiederholbarkeit von Tests
- Prüfen, ob die automatisierte Testsuite genug Verifizierungspunkte bietet

Jeder dieser Punkte wird nachstehend ausführlich erläutert.

Ausführung von Testskripten mit bekannten Bestanden- und Fehlgeschlagen-Ergebnissen
Wenn Testfälle fehlschlagen, die sonst bekanntlich bestehen, ist sofort klar, dass ein grundlegendes Problem vorliegt, das so schnell wie möglich behoben werden muss. Umgekehrt gilt: Wenn eine Testsuite erfolgreich durchlaufen wird, die eigentlich fehlschlagen müsste, muss der Testfall ermittelt werden, der nicht richtig funktionierte. Es ist wichtig, die korrekte Generierung von Protokolldateien, Ausführungsdaten, das Initialisieren (Setup) und das Aufräumen (Teardown) des Testfalls / Skripts zu überprüfen. Hilfreich ist auch, einige Tests anderer Typen und Teststufen durchzuführen (funktionale Tests, Performanztests, Komponententests usw.).

Prüfen der Testsuite
Überprüfung der Testsuite auf ihre Vollständigkeit (alle Testfälle haben erwartete Ergebnisse, Testdaten sind vorhanden) und die korrekte Version im Hinblick auf das Framework und das SUT.

Verifizieren neuer Tests mit Fokus auf neuen Funktionen des Frameworks
Wenn eine neue Funktion der TAS in Testfällen erstmalig verwendet wird, sollte es genau überprüft und überwacht werden, um sicherzustellen, dass die Funktionen ordnungsgemäß funktionieren.

Gewährleisten der Wiederholbarkeit von Tests
Wenn Tests wiederholt werden, muss das Ergebnis/Urteil des Tests immer dasselbe sein. Wenn die Testsuite Testfälle enthält, die kein zuverlässiges Ergebnis liefern (z. B. durch eine Wettlaufsituation), können diese aus der aktiven automatisierten Testsuite entfernt und gesondert analysiert werden, um die Ursache für das Problem zu ermitteln. Andernfalls wird für diese Testläufe immer wieder Zeit für die Ursachenforschung aufgewendet.

Sporadisch auftretende Fehler müssen ebenfalls analysiert werden. Das Problem kann im Testfall selbst oder im Framework liegen (oder sogar ein Problem des SUT sein). Mittels Analyse der Protokolldateien (des Testfalls, Frameworks und SUTs) lässt sich die Problemursache ermitteln. Auch ein Debugging kann notwendig sein. Für

das Finden der Ursache kann Unterstützung vom Testanalysten, Softwareentwickler und Bereichsexperten benötigt werden.

Prüfen, ob die automatisierte Testsuite und/oder die Testfälle genug Verifizierungspunkte bieten
Es muss möglich sein, zu verifizieren, dass die automatisierte Testsuite ausgeführt wurde und die erwarteten Ergebnisse geliefert hat. Es muss nachgewiesen werden, dass die Testsuite und/oder die Testfälle wie vorgesehen ausgeführt wurden. Dieser Nachweis besteht u.a. durch die Protokollierung der Start- und Endzeit jedes Testfalls, der Aufzeichnung des Testausführungsstatus nach Abschluss des jeweiligen Testfalls sowie ggf. der Verifizierung der Nachbedingungen.

8. Fortlaufende Optimierung – 150 min

Begriffe
Wartung

Lernziele für „Fortlaufende Optimierung“

8.1 Möglichkeiten der Optimierung der Testautomatisierung
ALTA-E-8.1.1 (K4) Analyse der technischen Aspekte einer bereitgestellten Testautomatisierungslösung und erfassen von Optimierungsempfehlungen

8.2 Planung der Realisierung der Testautomatisierungsverbesserung
ALTA-E-8.2.1 (K4) Analyse der automatisierten Testmittel einschließlich der Komponenten, Werkzeuge und unterstützenden Funktionsbibliotheken der Testumgebung, um zu ermitteln, an welchen Stellen nach Änderungen an Teilen der Testumgebung oder des SUT Konsolidierungen und Aktualisierungen erforderlich sind

8.1 Möglichkeiten der Optimierung der Testautomatisierung

Neben den nötigen Aufgaben der laufenden Wartung, um die TAS synchron zum SUT zu halten, gibt es in der Regel viele Möglichkeiten der Optimierung der TAS. Die Optimierung der TAS kann beispielsweise in der Erhöhung der Effizienz (weitere Reduzierung der Eingriffe durch den Menschen), der Erhöhung der Bedienfreundlichkeit, der Erweiterung des Funktionsumfangs und der Verbesserung des Supports für die Testaktivitäten bestehen. Die Entscheidung, wie die TAS verbessert wird, wird von den Vorteilen beeinflusst, die dem Projekt den größtmöglichen Nutzen bringen.

Bereiche der TAS, für die eine Optimierung in Betracht gezogen werden sollte, sind die Skripterstellung, die Verifizierung, die Architektur, die Vor- und Nachbearbeitung, die Dokumentation und die Werkzeugunterstützung. Diese werden nachstehend ausführlich erläutert.

Skripterstellung
Ansätze der Skripterstellung reichen vom einfach strukturierten Ansatz über datengetriebene Ansätze bis hin zu den komplexeren schlüsselwortgetriebenen Ansätzen, wie sie in Abschnitt 3.2.2 beschrieben wurden. Es kann ratsam sein, den bislang genutzten Automatisierungsansatz der TAS für alle neuen, automatisierten Tests auf ein komplexeres Verfahren umzustellen. Der Ansatz kann dann für alle bestehenden, automatisierten Tests nachgerüstet werden oder zumindest für diejenigen mit dem größten Wartungsaufwand.

Statt den Skripterstellungsumsatz komplett umzustellen, können sich die TAS-Verbesserungen auch auf die Implementierung von Skripten konzentrieren. Zum Beispiel:

- Prüfung, ob sich Testfälle/-schritte/-abläufe überschneiden, um die automatisierten Tests ggf. konsolidieren zu können.
  Testfälle die ähnliche Handlungsabfolgen enthalten, sollten diese Schritte nicht mehrmals implementieren. Vielmehr sollten diese Schritte zu einer Funktion zusammengefasst und in eine Bibliothek aufgenommen werden, damit sie wiederverwendet werden können. Diese Bibliotheksfunktionen können dann von verschiedenen Testfällen verwendet werden. Das verbessert die Wartungsfähigkeit der Testmittel.
  Wenn Testschritte nicht identisch, aber ähnlich sind, kann eine Parametrisierung erforderlich sein.
  Hinweis: Das ist ein typischer Ansatz beim schlüsselwortgetriebenen Testen.
- Etablierung eines Fehlerwiederherstellungsprozesses für TAS und SUT.
  Wenn während der Ausführung von Testfällen ein Fehler auftritt, muss die TAS in der Lage sein, eine Wiederherstellung aus diesem Fehlerzustand durchzuführen, um mit dem nächsten Testfall fortfahren zu können. Wenn ein Fehler im SUT auftritt, muss die TAS in der Lage sein, die erforderlichen

Wiederherstellungsschritte am SUT durchzuführen (z. B. Neustart des kompletten SUT).

- Evaluierung der Wartemechanismen, um sicherzustellen, dass die beste Option verwendet wird. Es gibt drei gängige Wartemechanismen:
  1. Festprogrammierte Wartezeiten (eine bestimmte Anzahl von Millisekunden) können eine Ursache für viele Probleme der Testautomatisierung sein.
  2. Das dynamische Warten durch Abfrage (z. B. durch Prüfen, ob eine bestimmte Zustandsänderung oder Aktion stattgefunden hat) ist flexibler und effizienter:
     - Es wird nur die nötige Zeit gewartet und keine Testausführungszeit verschwendet.
     - Wenn der Prozess aus irgendeinem Grund länger dauert, wartet die Abfrage genau bis zu dem Zeitpunkt, an dem der Zustand wahr wird. Hierbei kann der Einbau eines Zeitüberschreitungsmechanismus hilfreich sein, damit der Test bei einem Problem nicht unendlich lange wartet.
  3. Noch besser ist es, den Ereignismechanismus des SUT zu abonnieren. Das ist viel zuverlässiger als die beiden anderen Varianten. Die Testskriptsprache muss jedoch das Abonnieren von Ereignissen (Events) unterstützen und das SUT muss der Testanwendung diese Ereignisse zur Verfügung stellen. Dabei sollte nicht vergessen werden, einen Zeitüberschreitungsmechanismus einzubauen, um zu verhindern, dass der Test bei einem Problem unbegrenzt lang wartet.
- Behandlung der Testmittel als Software.
  Die Entwicklung und Wartung von Testmitteln ist lediglich eine Form der Softwareentwicklung. Daher sind gute Programmierpraktiken anzuwenden (z. B. Anwendung von Programmierrichtlinien, statischer Analyse, Code-Reviews). Es kann sogar ratsam sein, Softwareentwickler hinzuzuziehen (statt Testentwicklern), um bestimmte Teile der Testmittel zu programmieren (z. B. Bibliotheken).
- Evaluierung bestehender Skripte auf Überarbeitung/Löschung.
  Wenn Skripte problematisch sind (z. B. hin und wieder versagen oder hohe Wartungskosten verursachen) kann es ratsam sein, sie zu überarbeiten. Andere Testskripte können vielleicht ganz aus der Suite genommen werden, weil sie keinen konkreten Nutzen mehr bieten.

Testausführung
Die Ausführungszeit einer automatisierten Regressionstestsuite kann so stark variieren, dass sie evtl. nicht über Nacht abgeschlossen wird. Wenn das Testen zu lange dauert, kann es notwendig sein, auf mehreren Systemen parallel zu testen. Das ist aber nicht immer möglich. Wenn teure Systeme (Ziele) für das Testen verwendet werden, kann es sein, dass alle Tests auf einem Ziel durchgeführt werden müssen. Dann kann die Regressionstestsuite auf mehrere Teile aufgeteilt werden, die jeweils in einer vordefinierten Zeitdauer ausgeführt werden (z. B. in einer Nacht). Eine weitere Analyse zur Überdeckung automatisierter Tests kann unnötige Doppelungen in Form von Duplikaten offenbaren. Das Beseitigen dieser Duplikate kann die Ausführungszeit verkürzen und weitere Effizienzsteigerungen bringen.

Verifizierung
Bevor neue Verifizierungsfunktionen entwickelt werden, sollten standardmäßige Verifizierungsverfahren für alle automatisierten Tests genutzt werden. Damit verhindern Sie, dass Verifizierungsschritte bei mehreren Tests stets aufs Neue implementiert werden müssen. Wenn Verifizierungsverfahren nicht identisch, aber ähnlich sind, hilft eine Parametrisierung dabei, eine Funktion für mehrere Objekttypen einzusetzen.

Architektur
Es kann erforderlich sein, die Architektur zu ändern, um Verbesserungen der Testbarkeit des SUT zu ermöglichen. Diese Änderungen können in der Architektur des SUT und/oder in der Architektur der Automatisierungslösung vorgenommen werden. Das kann die Testautomatisierung erheblich optimieren, erfordert jedoch unter Umständen starke Eingriffe und Investitionen in SUT/TAS. Ein Beispiel: Wenn das SUT um APIs für das Testen erweitert werden soll, muss auch die TAS entsprechend refaktorisiert werden. Funktionen dieser Art nachträglich hinzuzufügen, kann ziemlich teuer sein; viel besser ist es, dies bereits bei Beginn der Automatisierung vorzusehen (und in den frühen Phasen der Entwicklung des SUT, siehe Abschnitt 2.3, Auslegung auf Testbarkeit und

Automatisierung).

Vor- und Nachbearbeitung
Stellen Sie standardmäßige Aufgaben für die Einrichtung und Außerbetriebnahme bereit. Diese sind auch als Vorbearbeitung (Setup) und Nachbearbeitung (Teardown) geläufig. Dadurch müssen die Aufgaben nicht für jeden automatisierten Test aufs Neue implementiert werden. Das senkt nicht nur die Wartungskosten, sondern verringert auch den Implementierungsaufwand für neue, automatisierte Tests.

Dokumentation
Das schließt alle Formen der Dokumentation ein: von der Dokumentation von Skripten (was Skripte tun, wie sie verwendet werden sollten usw.), über die Benutzerdokumentation für die TAS bis hin zu den von der TAS erzeugten Berichten und Protokollen.

TAS-Funktionen
Man kann die TAS um Features und Funktionen wie die detaillierte Berichterstattung, Protokolle, Integration mit anderen Systemen usw. erweitern. Das sollte man aber nur tun, wenn diese neuen Funktionen auch wirklich genutzt werden. Andernfalls erhöhen sie nur die Komplexität zu Lasten von Zuverlässigkeit und Wartbarkeit.

TAS-Updates und -Upgrades
Durch Updating oder Upgrading der TAS auf neue Versionen lassen sich neue Funktionen einführen, die von Testfällen genutzt werden können (oder Fehler beheben). Es besteht dabei allerdings das Risiko, dass sich die Aktualisierung des Frameworks (durch Upgraden der bestehenden oder Einführung neuer Testwerkzeuge) negativ auf bestehende Testfälle auswirkt. Vor dem Rollout einer neuen Version des Testwerkzeugs sollten daher Probetests durchgeführt werden. Die Probetests müssen repräsentativ für die automatisierten Tests der verschiedenen Anwendungen, die unterschiedlichen Testtypen und ggf. die unterschiedlichen Umgebungen sein.

8.2 Planung der Realisierung der Testautomatisierungsverbesserung

Änderungen an der bestehenden TAS erfordern eine sorgsame Planung und Analyse. Es wurde schließlich viel Mühe aufgewendet, um eine robuste TAS bestehend aus einer TAF und den Komponentenbibliotheken zu schaffen. Jede Änderung, ganz gleich wie trivial, kann weitreichende Folgen für die Zuverlässigkeit und Performanz der TAS haben.

Ermitteln von Änderungen an Komponenten der Testumgebung
Es muss evaluiert werden, welche Änderungen und Verbesserungen vorgenommen werden müssen. Welche Änderungen an der Testsoftware, an systemspezifischen Funktionsbibliotheken oder am OS sind erforderlich? All dies hat Einfluss auf den Betrieb der TAS. Oberstes Ziel ist es, sicherzustellen, dass die automatisierten Tests weiterhin effizient ausgeführt werden. Änderungen sollten schrittweise erfolgen, damit sich ihre Auswirkung auf die TAS durch eine limitierte Ausführung von Testskripten messen lassen. Wenn nachgewiesen wurde, dass keine nachteiligen Folgen bestehen, können die Änderungen vollständig implementiert werden. Als letzter Schritt sollte dann ein vollständiger Regressionslauf sicherstellen, dass die Änderungen an der TAS die automatisierten Testskripte nicht beeinträchtigt haben. Während der Ausführung dieses Regressionstests können Fehler gefunden werden. Durch Ermittlung der Ursache dieser Fehler (durch Berichte, Protokolle, Datenanalyse usw.) lässt sich sicherstellen, dass diese nicht die Folge der Optimierung der Automatisierungslösung sind.

Erhöhung der Effizienz und Wirksamkeit der zentralen TAS-Funktionsbibliotheken
Mit zunehmender Reife der TAS ergeben sich neue Wege, Aufgaben effizienter auszuführen. Diese neuen Techniken (einschließlich der Optimierung von Code in Funktionen, der Verwendung neuerer Betriebssystembibliotheken usw.) müssen in die zentralen Funktionsbibliotheken eingebunden werden, die vom aktuellen Projekt und anderen Projekten verwendet werden.

Konsolidieren mehrerer Funktionen, die auf denselben Steuerungstyp wirken
Ein Großteil dessen, was während eines automatisierten Testlaufs passiert, besteht im Aufrufen von Steuerelementen der GUI. Diese Aufrufe dienen dem Erhalt von Informationen über das Steuerelement (z. B.

sichtbar/nicht sichtbar, aktiviert/nicht aktiviert, Größe und Abmessungen, Daten usw.). Mit diesen Informationen kann ein automatisierter Test ein Element aus einer Dropdown-Liste wählen, Daten in ein Feld eingeben, einen Wert aus einem Feld auslesen usw. Es gibt mehrere Funktionen, die auf Steuerelemente wirken, um diese Informationen zu erlangen. Manche dieser Funktionen sind extrem spezialisiert, während andere dagegen eher allgemeinerer Natur sind. So kann es beispielsweise eine spezifische Funktion geben, die nur Dropdown-Listen ansteuert. Denkbar ist auch eine Funktion (die extra für die TAS erstellt und in ihr genutzt wird), die wiederum mit anderen Funktionen arbeitet, die sie selbst als einen ihrer Parameter spezifiziert. Daher kann ein TAE mehrere Funktionen verwenden, die sich zu Funktionen zusammenfassen lassen, um dieselben Ergebnisse bei geringerem Wartungsaufwand zu erzielen.

Refaktorisieren der TAA in Reaktion auf Änderungen im SUT
Im Verlauf der Existenz einer TAS müssen Modifikationen an ihr vorgenommen werden, um Änderungen im SUT Rechnung zu tragen. Mit der Entwicklung und Reifung des SUT muss sich auch die zugrunde liegende TAA entwickeln, um sicherzustellen, dass die Fähigkeit zur Unterstützung des SUT gegeben ist. Bei der Erweiterung der Funktionen muss darauf geachtet werden, dass diese nicht „aufgepfropft“, sondern auf Architekturebene der automatisierten Lösung analysiert und geändert werden. Dadurch wird sichergestellt, dass sofern neue SUT-Funktionen zusätzliche Testskripte erfordern, kompatible Komponenten für diese neu zu automatisierenden Tests vorhanden sind.

Benennungskonventionen und Standardisierung
Bei der Einführung von Änderungen müssen Benennungskonventionen für den neuen Automatisierungscode sowie für die neuen Funktionsbibliotheken mit den zuvor definierten Standards übereinstimmen (siehe Abschnitt 4.3.2, Umfang und Ansatz).

Prüfen bestehender Testskripte auf Überarbeitung/Löschung
Der Änderungs- und Optimierungsprozess schließt auch eine Bewertung bestehender Skripte, ihrer Verwendung und ihres aktuellen Nutzwerts mit ein. Wenn die Ausführung bestimmter Tests zu komplex und zeitaufwändig ist, kann es möglichweise praktischer und effizienter sein, sie in kleinere Tests zu zerlegen. Durch das Löschen von Tests, die selten oder gar nicht ausgeführt werden, sinkt die Komplexität der TAS und die Übersichtlichkeit über den Wartungsbedarf steigt.

9. Literaturhinweise

9.1 Standards

Zu den Standards für die Testautomatisierung zählen u. a.:

- die Testing and Test Control Notation (TTCN-3) von ETSI (European Telecommunication Standards Institute) und ITU (International Telecommunication Union), bestehend aus:
  - ES 201 873-1: TTCN-3 Core Language
  - ES 201 873-2: TTCN-3 Tabular Presentation Format (TFT)
  - ES 201 873-3: TTCN-3 Graphical Presentation Format (GFT)
  - ES 201 873-4: TTCN-3 Operational Semantics
  - ES 201 873-5: TTCN-3 Runtime Interface (TRI)
  - ES 201 873-6: TTCN-3 Control Interface (TCI)
  - ES 201 873-7: Using ASN.1 with TTCN-3
  - ES 201 873-8: Using IDL with TTCN-3
  - ES 201 873-9: Using XML with TTCN-3
  - ES 201 873-10: TTCN-3 Documentation
  - ES 202 781: Extensions: Configuration and Deployment Support
  - ES 202 782: Extensions: TTCN-3 Performance and Real-Time Testing
  - ES 202 784: Extensions: Advanced Parameterization
  - ES 202 785: Extensions: Behaviour Types
  - ES 202 786: Extensions: Support of interfaces with continuous signals
  - ES 202 789: Extensions: Extended TRI
- die Automatic Test Markup Language (ATML) des IEEE (Institute of Electrical and Electronics Engineers), bestehend aus:
  - IEEE Std 1671.1: Test Description
  - IEEE Std 1671.2: Instrument Description
  - IEEE Std 1671.3: UUT Description
  - IEEE Std 1671.4: Test Configuration Description
  - IEEE Std 1671.5: Test Adaptor Description
  - IEEE Std 1671.6: Test Station Description
  - IEEE Std 1641: Signal and Test Definition
  - IEEE Std 1636,1: Test Results
- die ISO/IEC/IEEE 29119-3: Software-und Systemengineering – Software-Test – Teil 3: Testdokumentation
- das UML Testing Profile (UTP) der OMG (Object Management Group) mit Vorgaben für Testspezifikationskonzepte für
  - Testarchitektur
  - Testdaten
  - Testverhalten
  - Testprotokollierung
  - Testmanagement

9.2 ISTQB-Dokumente

ID	Referenz
GTB-Glossar	ISTQB/GTB Standardglossar der Testbegriffe, Version 3.21, abrufbar von [GTB-Web]
ISTQB-AL-TM	ISTQB Certified Tester, Advanced Level Syllabus, Test Manager, Version 2012, abrufbar von [ISTQB-Web]
ISTQB-AL-TTA	ISTQB Certified Tester, Advanced Level Syllabus, Technical Test Analyst, Version 2012, abrufbar von [ISTQB-Web]
ISTQB-AL-Modules	https://www.istqb.org/]ISTQB-FLISTQB Foundation Level Syllabus, Version 2011, abrufbar von [ISTQB-Web]
ISTQB-FL	ISTQB Foundation Level Syllabus, Version 2011, abrufbar von [ISTQB-Web]
ISTQB-Glossary	ISTQB-Begriffsglossar, Version 2.4, 4. Juli 2014, abrufbar von [ISTQB-Web]

9.3 Marken

Die folgenden eingetragenen Marken und Dienstleistungsmarken werden in diesem Dokument verwendet:

ISTQB® ist eine eingetragene Marke des International Software Testing Qualifications Board.

9.4 Bücher

ID	Buch
[Baker08]	Paul Baker, Zhen Ru Dai, Jens Grabowski und Ina Schieferdecker, „Model-Driven Testing: Using the UML Testing Profile“, Springer, Auflage 2008, ISBN-10: 3540725628, ISBN-13: 978-3540725626
[Dustin09]	Efriede Dustin, Thom Garrett, Bernie Gauf, „Implementing Automated Software Testing: how to save time and lower costs while raising quality“, Addison-Wesley, 2009, ISBN 0-321-58051-6
[Dustin99]	Efriede Dustin, Jeff Rashka, John Paul, „Automated Software Testing: introduction, management, and performance“, Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879
[Fewster&Graham12]	Mark Fewster, Dorothy Graham, „Eperiences of Test Automation: Case Studies of Software Test Automation“, Addison-Wesley, 2012
[Fewster&Graham99]	Mark Fewster, Dorothy Graham, „Software Test Automation: Effective use of test execution tools“, ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400
[McCaffrey06]	James D. McCaffrey, „.NET Test Automation Recipes: A Problem-Solution Approach“, APRESS, 2006 ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3
[Mosley02]	Daniel J. Mosley, Bruce A. Posey, „Just Enough Software Test Automation“, Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682
[Willcock11]	Colin Willcock, Thomas Deiß, Stephan Tobies und Stefan Keil, „An Introduction to TTCN-3“ Wiley, 2nd Edition 2011, ISBN- 10: 0470663065, ISBN-13: 978-0470663066

9.5 Internetquellen

ID	Referenz
GTB-Web	Webseite des German Testing Board: www.german-testing-board.info
ISTQB-Web	Website des International Software Testing Qualifications Board: www.istqb.org

10. Hinweis für Ausbildungsanbieter

10.1 Kursdauer

Für jedes Kapitel im Lehrplan ist eine Zeitdauer in Minuten angegeben. Sie soll als Richtwert dafür dienen, wie viel Zeit auf die einzelnen Abschnitte eines akkreditierten Kurses entfallen und wie viel Zeit mindestens für die Behandlung jedes Abschnitts im Kurs einzuplanen ist.

Die Ausbildungsanbieter können mehr Zeit als angegeben aufwenden. Den Kursteilnehmern steht es ebenfalls frei sich länger mit einem Abschnitt zu beschäftigen. Die Kurslehrpläne müssen sich nicht an die Reihenfolge im vorliegenden Lehrplan halten. Der Kurs muss nicht als Blockkurs durchgeführt werden.

Die nachstehende Tabelle enthält Richtwerte für die Lehr- und Übungsdauer der einzelnen Kapitel (alle Zeiten sind in Minuten angegeben).

Kapitel	**Minuten**
0. Einleitung	0
1. Einführung in die Testautomatisierung und deren Ziele	30
2. Vorbereitungen für die Testautomatisierung	165
3. Die generische Testautomatisierungsarchitektur	270
4. Risiken und Eventualitäten bei der Softwareverteilung	150
5. Berichte und Metriken bei der Testautomatisierung	165
6. Überführung des manuellen Testens in eine automatisierte Umgebung	120
7. Verifizieren der TAS	120
8. Fortlaufende Optimierung	150
Gesamt:	1170

Die Gesamtkursdauer in Tagen beträgt basierend auf dem Mittelwert von sieben Stunden pro Arbeitstag: 2 Tage, 5 Stunden, 30 Minuten.

10.2 Praktische Übungen am Arbeitsplatz

In diesem Lehrplan gibt es keine Übungen, die am Arbeitsplatz durchzuführen sind.

10.3 Regeln für das e-Learning

Alle Teile dieses Lehrplans können auch als e-Learning-Kurs aufbereitet werden.

11. Index