ISTQB® Certified Tester

Lehrplan

Advanced Level

Technical Test Analyst

Version 4.0 DE

Deutschsprachige Ausgabe
Herausgegeben durch das German Testing Board e.V.

Übersetzung des englischsprachigen Lehrplans des International Software Testing Qualifications Board (ISTQB®), Originaltitel: Certified Tester, Advanced Level Syllabus, Technical Test Analyst, Version 4.0.

Urheberschutzvermerk

Copyright © German Testing Board (nachstehend als GTB® bezeichnet).

Urheberrecht © 2019 die Autoren der englischen Originalausgabe: ISTQB Advanced Level Working Group:

Mike Smith (chair), Graham Bath (vice-chair), Rex Black, Judy McKay, Kenji Onoshi, Erik van Veenendaal.

Urheberrecht © 2021 die Autoren der englischen Aktualisierung: ISTQB Advanced and Expert Level Working Group:

Stuart Reid (chair), Adam Roman, Armin Born, Christian Graf

Urheberrecht © an der Übersetzung in die deutsche Sprache 2021: GTB Arbeitsgruppe Advanced Level

Graham Bath (Leiter), Matthias Hamburg, Marc-Florian Wendland

Dieser ISTQB® Certified Tester Lehrplan Advanced Level – Technical Test Analyst, deutschsprachige Ausgabe, ist urheberrechtlich geschützt.

Inhaber der ausschließlichen Nutzungsrechte an dem Werk sind das German Testing Board (GTB.

Die Nutzung des Werks ist – soweit sie nicht nach den nachfolgenden Bestimmungen und dem Gesetz über Urheberrechte und verwandte Schutzrechte vom 9. September 1965 (UrhG) erlaubt ist – nur mit ausdrücklicher Zustimmung des GTB gestattet. Dies gilt insbesondere für die Vervielfältigung, Verbreitung, Bearbeitung, Veränderung, Übersetzung, Mikroverfilmung, Speicherung und Verarbeitung in elektronischen Systemen sowie die öffentliche Zugänglichmachung.

Dessen ungeachtet ist die Nutzung des Werks einschließlich der Übernahme des Wortlauts, der Reihenfolge sowie Nummerierung der in dem Werk enthaltenen Kapitelüberschriften für die Zwecke der Anfertigung von Veröffentlichungen gestattet. Die Verwendung der in diesem Werk enthaltenen Informationen erfolgt auf die alleinige Gefahr des Nutzers. GTB übernehmen insbesondere keine Gewähr für die Vollständigkeit, die technische Richtigkeit, die Konformität mit gesetzlichen Anforderungen oder Normen sowie die wirtschaftliche Verwertbarkeit der Informationen. Es werden durch dieses Dokument keinerlei Produktempfehlungen ausgesprochen.

Die Haftung von GTB gegenüber dem Nutzer des Werks ist im Übrigen auf Vorsatz und grobe Fahrlässigkeit beschränkt. Jede Nutzung des Werks oder von Teilen des Werks ist nur unter Nennung des GTB als Inhaber der ausschließlichen Nutzungsrechte sowie der oben genannten Autoren als Quelle gestattet.

© German Testing Board e.V.

Änderungsübersicht der deutschsprachigen Ausgabe

Version	Datum	Bemerkung
v4.0 DE	5.1.2022	Deutschsprachige Fassung freigegeben

© German Testing Board e.V.

Inhaltsverzeichnis

© German Testing Board e.V.

© German Testing Board e.V.

© German Testing Board e.V.

© German Testing Board e.V.

© German Testing Board e.V.

Danksagung

Das Autorenteam der englischen Originalausgabe dieses Dokuments dankt den Reviewern für ihre Kommentare und Beiträge:

Dani Almog	Andrew Archer	Rex Black
Armin Born	Sudeep Chatterjee	Tibor Csöndes
Wim Decoutere	Klaudia Dussa-Zieger	Melinda Eckrich-Brájer
Peter Foldhazi	David Frei	Karol Frühauf
Jan Giesen	Attila Gyuri	Matthias Hamburg
Tamás Horváth	N. Khimanand	Jan te Kock
Attila Kovács	Claire Lohr	Rik Marselis
Marton Matyas	Judy McKay	Dénes Medzihradszky
Petr Neugebauer	Ingvar Nordström	Pálma Polyák
Meile Posthuma	Stuart Reid	Lloyd Roden
Adam Roman	Jan Sabak	Péter Sótér
Benjamin Timmermans	Stephanie van Dijck	Paul Weymouth

Das Autorenteam der vorliegenden Übersetzung in die deutsche Sprache dankt den Reviewern für ihre Kommentare und Beiträge:

Matthias Hamburg, Marc-Florian Wendland, Jan te Kock, Monika Bögge, Dr. Klaudia Dussa-Zieger.

© German Testing Board e.V.

0. Einführung in diesen Lehrplan

0.1 Zweck dieses Dokuments

Dieser Lehrplan bildet die Grundlage für die Qualifikation als Technical Test Analyst auf der Aufbaustufe Advanced Level des Softwaretest-Ausbildungsprogramms des International Software Testing Qualifications Board (im Folgenden ISTQB® genannt). Das German Testing Board e.V. (im Folgenden GTB® genannt) hat diesen Lehrplan in die deutsche Sprache übersetzt. Das GTB® und ISTQB® stellen den Lehrplan folgenden Adressaten zur Verfügung:

- 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 Schulungskurses 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 Das Certified Tester Advanced Level Technical Test Analyst

Die Qualifikation zum Advanced Level Technical Test Analyst richtet sich an alle Personen im Bereich Softwaretesten, die ihr Wissen über Technical Test Analyst vertiefen möchten, bzw. an Personen, die sich in ihrer beruflichen Laufbahn auf Technical Test Analyst spezialisieren wollen.

Dieser Lehrplan behandelt die folgenden Hauptaspekte des Technical Test Analyst:

- Statische Analyse
- Dynamische Analyse
- Strukturbasierter Test
- Qualitätsmerkmale bei technischen Tests
- Reviews für den Technical Test Analyst

© German Testing Board e.V.

- Testwerkzeuge und Automatisierung für den Technical Test Analyst

Die Inhalte zum Thema Technical Test Analyst, die im ISTQB® Advanced Level Lehrplan beschrieben werden, sind konsistent mit diesem Lehrplan und wurden auf gemeinsamer Grundlage entwickelt.

0.3 Geschäftlicher Nutzen

Dieser Abschnitt zeigt den geschäftlichen Nutzen auf, der von Kandidaten mit Zertifizierung im Advanced Level Technical Test Analyst erwartet werden kann.

TTA1 die typischen Risiken in Zusammenhang mit Performanz, IT-Sicherheit, Zuverlässigkeit, Übertragbarkeit und Wartbarkeit von Softwaresystemen erkennen und klassifizieren.

TTA2 technische Details für die Planung, den Entwurf und die Durchführung von zur Reduzierung der Risiken in Zusammenhang mit Performanz, IT-Sicherheit, Zuverlässigkeit, Übertragbarkeit und Wartbarkeit bereitstellen.

TTA3 angemessene White-Box-Testverfahren auswählen und anwenden, und damit sicher-stellen, dass die Tests, basierend auf der erzielten Designüberdeckung, ausreichend Vertrauen erzeugen.

TTA4 effektiv an technischen Reviews mit Entwicklern und Softwarearchitekten mitwirken und hierbei seine Kenntnisse über typische Fehler in Programmcode und Architektur ein-bringen.

TTA5 die Qualitätsmerkmale von Programmcode und Architektur durch den Einsatz verschiedener Analyseverfahren verbessern.

TTA6 die Kosten und Nutzen aufzeigen, die durch den Einsatz bestimmter Arten von Testautomatisierung zu erwarten sind.

TTA7 die richtigen Werkzeuge für die Automatisierung technischer Testaktivitäten auswählen.

TTA8 die technischen Belange und Konzepte in Zusammenhang mit dem Einsatz von Test-automatisierung verstehen.

0.4 Prüfungsrelevante Lernziele

Die Lernziele unterstützen die geschäftlichen Ziele und dienen zur Ausarbeitung der Prüfung für die Zertifizierung zum Advanced Level Technical Test Analyst. Den einzelnen Lernzielen ist jeweils eine kognitive Wissensstufe (K-Stufe) zugeordnet.

© German Testing Board e.V.

Die K-Stufe (bzw. kognitive Wissensstufe) dient dazu, Lernziele gemäß der überarbeiteten Taxonomie von Bloom [Anderson01] zu klassifizieren. Das ISTQB® verwendet diese Taxonomie bei der Erstellung der Prüfungen zu den Lehrplänen.

Dieser Lehrplan berücksichtigt folgende der insgesamt sechs möglichen kognitiven Wissensstufen:

Stufe	Kennwort	Beschreibung
1	Kennen	Der Lernende kann einen Begriff oder ein Konzept erkennen, abrufen und sich daran erinnern.
2	Verstehen	Der Lernende kann Aussagen zum Thema begründen oder erläutern.
3	Anwenden	Der Lernende kann erworbenes Wissen auf gegebene neue Situationen übertragen oder zur Problemlösung anwenden.
4	Analysieren	Der Lernende kann Informationen mit Bezug auf eine Vorgehensweise oder ein Verfahren zum besseren Verständnis in ihre einzelnen Bestandteile aufgliedern sowie zwischen Fakten und Rückschlüssen unterscheiden.

Jedes Lernziel wird gemäß seiner kognitiven Stufe geprüft. Die relevanten Lernziele der kognitiven Stufen K2 und höher werden immer zu Beginn des jeweiligen Kapitels angegeben. Darüber hinaus sind für alle Schlüsselbegriffe, die zu Beginn des jeweiligen Kapitels gelistet werden, die jeweiligen Definitionen aus dem GTB/ISTQB Standardglossar der Testbegriffe [GTB-GLOSSAR] entsprechend der kognitiven Stufe K1 prüfungsrelevant.

Allgemein gilt, dass der gesamte Inhalt des vorliegenden Lehrplans entsprechend der kognitiven Stufe K1 geprüft werden kann. Dies bedeutet, dass die Prüfungskandidaten Begriffe und Inhalte erkennen, sich an sie erinnern und sie wiedergeben können.

0.5 Empfohlene Unterrichtszeiten

Für jedes Lernziel dieses Lehrplans wurde die Mindestschulungszeit festgelegt. Die gesamte Unterrichtszeit für jedes Kapitel ist in der Kapitelüberschrift angegeben.

© German Testing Board e.V.

0.6 Zulassung zur Prüfung

Voraussetzung für die Prüfung zum Advanced Level Technical Test Analyst ist das erworbene Zertifikat zum ISTQB® Certified Tester Foundation Level (CTFL®).

0.7 Informationsquellen

Die im Lehrplan verwendeten Fachbegriffe des Testens in Deutsch sind im GTB/ISTQB Standardglossar der Testbegriffe definiert [GTB-GLOSSAR].

Kapitel 6 enthält eine Liste empfohlener Bücher und Fachartikel zum Thema Technical Test Analyst.

0.8 Detaillierungsgrad des Lehrplans

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

- Allgemeine Lernziele, die die Absicht des Advanced Level Technical Test Analyst-Lehrplans beschreiben.
- Eine Liste von Punkten, die Schulungsteilnehmer erinnern müssen.
- Lernziele der einzelnen Wissensgebiete, die die zu erreichenden kognitiven Lernergebnisse beschreiben.
- Eine Beschreibung der Schlüsselkonzepte, einschließlich Verweisen auf Quellen wie anerkannte Literatur oder Normen

Der Lehrplaninhalt ist keine Beschreibung des gesamten Wissensgebietes, sondern spiegelt den Detaillierungsgrad wider, der in Advanced Level-Trainingskursen abzudecken ist. Der Lehrplan konzentriert sich auf Materialien, die für alle Softwareprojekte gelten können, einschließlich agiler Softwareentwicklung. Der Lehrplan enthält keine spezifischen Lernziele in Bezug auf einen bestimmten Softwareentwicklungslebenszyklus, aber es wird behandelt, wie diese Konzepte in agiler Softwareentwicklung, anderen Arten von iterativen und inkrementellen Lebenszyklen und in sequentiellen Lebenszyklen angewendet werden.

0.9 Aufbau des Lehrplans

Der Lehrplan besteht aus sechs Kapiteln mit prüfungsrelevanten Inhalten. Die Kapitelüberschrift spezifiziert die Mindestzeit für den Unterricht und die Übungen des Kapitels; eine weitere Aufgliederung der Zeitangabe ist nicht vorgesehen. Für

© German Testing Board e.V.

akkreditierte Trainingskurse erfordert der Lehrplan eine Gesamtunterrichtszeit von mindestens 21 Stunden, die sich wie folgt auf die sechs Kapitel verteilt:

- Kapitel 1: Die Aufgaben des Technical Test Analysten im Testprozess (30 Minuten)
- Kapitel 2: White-Box-Testverfahren (300 Minuten)
- Kapitel 3: Statische und dynamische Analyse (240 Minuten)
- Kapitel 4: Qualitätsmerkmale bei technischen Tests (345 Minuten)
- Kapitel 5: Reviews (165 Minuten)
- Kapitel 6: Testwerkzeuge und Testautomatisierung (180 Minuten)

© German Testing Board e.V.

1. Die Aufgaben des Technical Test Analysten beim risikobasierten Testen – 30 Min.

Schlüsselbegriffe

Produktrisiko, Projektrisiko, risikobasiertes Testen, Risikobewertung, Risikoidentifizierung, Risikominderung

Lernziele für "Die Aufgaben des Technical Test Analysten beim risikobasierten Testen"

1.2 Risikobasierte Testaufgaben

TTA-1.2.1 (K2) Die allgemeinen Risikofaktoren zusammenfassen, die der Technical Test Analyst in der Regel berücksichtigen muss

TTA-1.2.2 (K2) Die Aktivitäten des Technical Test Analysten bei einer risikobasierten Testvorgehensweise zusammenfassen

1.1 Einführung

Das Aufsetzen und Management einer risikobasierten Teststrategie liegt in der Verantwortung des Testmanagers. In der Regel wird der Testmanager jedoch die Unterstützung des Technical Test Analysten anfordern, um sicherzustellen, dass die risikobasierte Testvorgehensweise fachgerecht implementiert wird.

Technical Test Analysten arbeiten innerhalb des vom Testmanager für das Projekt festgelegten risikobasierten Testrahmens. Sie bringen ihr Wissen über die technischen Produktrisiken ein, die mit dem Projekt verbunden sind, wie z.B. Risiken in Bezug auf IT-Sicherheit, Systemzuverlässigkeit und Performanz. Außerdem sollten sie zur Identifizierung und Behandlung von Projektrisiken in Zusammenhang mit Testumgebungen beitragen, z.B. bezüglich der Beschaffung und Einrichtung von Testumgebungen für Performanz-, Zuverlässigkeits- und IT-Sicherheitstests.

1.2 Risikobasierte Testaufgaben

Technical Test Analysten wirken aktiv an den folgenden risikobasierten Testaufgaben mit:

- Risikoidentifizierung
- Risikobewertung
- Risikominderung

© German Testing Board e.V.

Diese Aufgaben werden während des gesamten Projekts iterativ durchgeführt, um neu auftretende Risiken und sich ändernde Prioritäten zu berücksichtigen und um den Risikostatus regelmäßig zu bewerten und zu kommunizieren.

1.2.1 Risikoidentifizierung

Durch die Einbeziehung einer möglichst breiten Auswahl an Stakeholdern kann der Risikoidentifizierungsprozess höchstwahrscheinlich die größtmögliche Menge an wichtigen Risiken aufdecken. Da Technical Test Analysten über spezifische technische Fähigkeiten verfügen, sind sie besonders gut dafür geeignet, Experten-Interviews zu führen, Brainstorming mit Kollegen durchzuführen, sowie ihre Erfahrungen zu analysieren, um die Bereiche für mögliche Produktrisiken zu bestimmen. Insbesondere arbeiten Technical Test Analysten eng mit anderen Stakeholdern wie Entwicklern, Architekten, Betriebsingenieuren, Produktverantwortlichen (Product Owner), dem lokalen Support, technischen Experten und den Mitarbeitern des Service-Desks-zusammen, um die technischen Risiken zu ermitteln, die sich auf das Produkt und das Projekt auswirken. Die Einbeziehung anderer Stakeholder wird normalerweise vom Testmanager gefördert, denn sie stellt sicher, dass alle Ansichten berücksichtigt werden.

Die Risiken, die vom Technical Test Analysten identifiziert werden könnten, basieren typischerweise auf den in Kapitel 4 aufgeführten Produkt-Qualitätsmerkmalen [gemäß ISO25010].

1.2.2 Risikobewertung

Während es bei der Risikoidentifikation darum geht, möglichst viele vorhandene Risiken zu identifizieren, befasst sich die Risikobewertung mit der Untersuchung der identifizierten Risiken. Dabei werden die einzelnen Risiken kategorisiert und jeweils die Eintrittswahrscheinlichkeit und das Schadensausmaß bestimmt.

Die Eintrittswahrscheinlichkeit eines Produktrisikos wird üblicherweise als Wahrscheinlichkeit interpretiert, dass die entsprechende Fehlerwirkung im System unter Test eintritt. Der Technical Test Analyst trägt zum Verständnis der Eintrittswahrscheinlichkeiten eines jeden technischen Produktrisikos bei, während der Test Analyst dazu beiträgt, die potenziellen betriebswirtschaftlichen Auswirkungen des Problems zu verstehen, falls dieses auftritt.

Projektrisiken, die zum Problem werden, können den Gesamterfolg des Projekts beeinträchtigen. Bei der Bewertung des Projektrisikos müssen normalerweise folgende allgemeine Risikofaktoren berücksichtigt werden:

- Konflikte zwischen Stakeholdern hinsichtlich der technischen Anforderungen

© German Testing Board e.V.

- Kommunikationsprobleme aufgrund der räumlichen Verteilung der Entwicklungsorganisation
- Werkzeuge und Technologie (einschließlich entsprechender Fähigkeiten)
- Zeitdruck, knappe Ressourcen, Druck durch das Management
- Fehlen einer frühzeitigen Qualitätssicherung
- häufige Änderungen bei technischen Anforderungen

Produktrisiken, die zum Problem werden, können zu einer erhöhten Anzahl an Fehlerzuständen führen. Bei der Risikobewertung müssen normalerweise folgende allgemeine Risikofaktoren berücksichtigt werden:

- Komplexität der Technologie
- Komplexität der Codestruktur
- Umfang der Änderungen im Programmcode (Einfügungen, Löschungen, Modifikationen)
- hohe Anzahl bisheriger Fehlerzustände in Zusammenhang mit technischen Qualitätsmerkmalen (Fehlerhistorie)
- technische Schnittstellen-/Integrationsprobleme

Anhand der vorhandenen Informationen über das Risiko und basierend auf den vom Testmanager vorgegebenen Richtlinien schlägt der Technical Test Analyst eine erste Einschätzung der Eintrittswahrscheinlichkeit des Risikos vor. Der Anfangswert kann vom Testmanager angepasst werden, nachdem die Ansichten der Stakeholder in Betracht gezogen wurden. Das Schadensausmaß des Risikos wird normalerweise vom Test Analysten bestimmt.

1.2.3 Risikominderung

Während des Projekts beeinflussen Technical Test Analysten wie das Testen auf die identifizierten Risiken reagiert. Dazu gehört im Allgemeinen:

- Entwurf von Testfällen für Risiken, die sich auf Bereiche mit hohem Risiko beziehen und Helfen bei der Bewertung des Restrisikos
- Reduzieren der Risiken durch die Ausführung der entworfenen Testfälle und durch die Umsetzung von Maßnahmen zur Risikominderung und Vorkehrung gegen Risiken, die im Testkonzept festgelegt sind
- Bewerten der Risiken basierend auf zusätzlichen Informationen und Erkenntnissen, die im Projektverlauf gewonnen wurden, sowie die Nutzung dieser Informationen für die Umsetzung von Maßnahmen zur Risikominderung, die die Eintrittswahrscheinlichkeit dieser Risiken reduzieren

© German Testing Board e.V.

Der Technical Test Analyst arbeitet häufig mit Spezialisten in Bereichen wie IT-Sicherheit und Performanz zusammen, um Maßnahmen zur Risikominderung und Elemente der Teststrategie festzulegen. Weitere Informationen finden Sie in den ISTQB® Specialist-Lehrplänen, z.B. im Lehrplan Sicherheitstester [CT_SEC_SYL] und im Lehrplan Performanztest [CT_PT_SYL]

© German Testing Board e.V.

2. White-Box-Testverfahren – 300 Min.

Schlüsselbegriffe

Anomalie, Anweisungstest, API-Testen, atomare Bedingung, Entscheidungstest, Kontrollfluss, Mehrfachbedingungstest, modifizierter Bedingungs-/Entscheidungstest, Safety Integrity Level, White-Box-Testverfahren

Lernziele für "White-Box-Testverfahren"

2.2 Anweisungstest

TTA-2.2.1 (K3) Den Anweisungstest anwenden, um Testfälle für ein bestimmtes Testobjekt zu entwerfen, die eine definierte Überdeckung erzielen

2.3 Entscheidungstest

TTA-2.3.1 (K3) Den Entscheidungstest anwenden, um Testfälle für ein bestimmtes Testobjekt zu entwerfen, die eine definierte Überdeckung erzielen

2.4 Modifizierter Bedingungs-/Entscheidungstest

TTA-2.4.1 (K3) Den modifizierten Bedingungs-/Entscheidungstest anwenden, um Testfälle für ein bestimmtes Testobjekt zu entwerfen, die eine vollständige modifizierte Bedingungs-/Entscheidungsüberdeckung (MC/DC) erzielen

2.5 Mehrfachbedingungstest

TTA-2.5.1 (K3) Den Mehrfachbedingungstest anwenden, um Testfälle für ein bestimmtes Testobjekt zu entwerfen, die eine definierte Überdeckung erzielen

2.6 Basispfadtest (*wurde aus der Version v4.0 dieses Lehrplans entfernt*)

TTA-2.6.1 wurde aus der Version v4.0 dieses Lehrplans entfernt

2.7 API-Testen

TTA-2.7.1 (K2) Die Anwendbarkeit des API-Tests und die Art der damit gefundenen Fehlerzustände verstehen

2.8 White-Box-Testverfahren auswählen

TTA-2.8.1 (K4) Für eine vorgegebene Projektsituation ein geeignetes White-Box-Testverfahren auswählen

© German Testing Board e.V.

2.1 Einführung

In diesem Kapitel werden White-Box-Testverfahren beschrieben. Diese Verfahren sind auf Code und andere Strukturen, wie z.B. Geschäftsprozesse anwendbar, die einen Kontrollfluss haben.

Jedes der beschriebenen Verfahren leitet Testfälle systematisch aus strukturellen Elementen ab und befasst sich gezielt mit besonderen Aspekten der betrachteten Struktur. Die mit diesen Verfahren entworfenen Testfälle erfüllen die Überdeckungskriterien, die als Ziel festgelegt sind und gemessen werden. Das Erreichen einer vollständigen Überdeckung (d.h. 100%) bedeutet nicht, dass die Menge von Tests vollständig ist. Es bedeutet vielmehr, dass das vorliegende Testverfahren für die untersuchte Struktur keine weiteren nützlichen Tests bietet.

Es werden Testeingaben erzeugt, die sicherstellen, dass ein Testfall einen bestimmten Teil des Codes (z.B. eine Anweisung oder einen Entscheidungsausgang) ausführt. Die Ermittlung der Testeingaben, die dazu führen, dass ein bestimmter Teil des Codes ausgeführt wird, kann herausfordernd sein, insbesondere wenn sich der entsprechende Teil des Codes am Ende eines langen Kontrollfluss-Teilpfades mit mehreren Entscheidungen befindet. Die erwarteten Ergebnisse werden auf Basis einer Quelle außerhalb dieser Struktur ermittelt, wie z.B. einer Anforderungs- oder Entwurfsspezifikation oder einer sonstigen Testbasis.

Im vorliegenden Lehrplan werden die folgenden Testverfahren behandelt:

- Anweisungstest
- Entscheidungstest
- Modifizierter Bedingungs-/Entscheidungstest
- Mehrfachbedingungstest
- API-Test

Anweisungstest und Entscheidungstest werden im Foundation Level-Lehrplan [CTFL_SYL] eingeführt. Beim Anweisungstest liegt der Fokus auf der Überprüfung der ausführbaren Anweisungen im Code, während der Entscheidungstest auf die Entscheidungsausgänge fokussiert ist.

Der modifizierte Bedingungs-/Entscheidungstest und der Mehrfachbedingungstest basieren auf Entscheidungsprädikaten, die mehrere Bedingungen enthalten; beide Verfahren decken ähnliche Arten von Fehlerzuständen auf. Ganz gleich wie komplex ein Entscheidungsprädikat auch sein mag, die Entscheidung wird letztlich zu WAHR oder FALSCH evaluiert, was wiederum bestimmt, welcher Pfad im Code ausgeführt wird. Ein Fehlerzustand ist festgestellt, wenn der beabsichtigte Pfad nicht genommen wird, weil ein Entscheidungsprädikat nicht wie erwartet evaluiert wurde.

© German Testing Board e.V.

Siehe [ISO 29119] für weitere Einzelheiten zur Spezifikation sowie Beispiele für diese Verfahren.

2.2 Anweisungstest

Beim Anweisungstest werden die ausführbaren Anweisungen im Code ausgeführt. Die Überdeckung wird an der Anzahl der im Test ausgeführten Anweisungen dividiert durch die Gesamtzahl aller ausführbaren Anweisungen im gesamten Testobjekt gemessen und üblicherweise als Prozentsatz angegeben.

Anwendbarkeit

Das Erreichen vollständiger Anweisungsüberdeckung sollte als das Minimum für jeden zu testenden Programmcode angestrebt werden, auch wenn dies in der Praxis nicht immer möglich ist.

Einschränkungen/Schwierigkeiten

Das Erreichen vollständiger Anweisungsüberdeckung sollte als das Minimum für jeden zu testenden Programmcode angestrebt werden, auch wenn dies in der Praxis aufgrund von Einschränkungen hinsichtlich der verfügbaren Zeit und/oder des Aufwands nicht immer möglich ist. Selbst bei einer hohen Anweisungsüberdeckung können bestimmte Fehlerzustände in der Logik des Codes unerkannt bleiben. In vielen Fällen ist das Erreichen von 100 % Anweisungsüberdeckung aufgrund von nicht erreichbarem Code gar nicht möglich. Auch wenn unerreichbarer Code generell nicht als gute Programmierpraxis gilt, kann er dennoch vorkommen, z.B. wenn eine Switch-Anweisung einen Standardfall (Default) haben muss, aber alle möglichen Fälle explizit behandelt werden.

2.3 Entscheidungstest

Beim Entscheidungstest werden die Entscheidungsausgänge im Code untersucht. Dazu folgen die Testfälle den Kontrollflüssen, die von einem Entscheidungspunkt ausgehen. (z.B. „bei einer IF-Anweisung einen für “wahr” und einen für “falsch”; bei einer CASE-Anweisung wären Testfälle für alle möglichen Ergebnisse nötig; bei einer LOOP-Anweisung gibt es je einen Kontrollfluss für den wahren und den falschen Ausgang der Schleifenbedingung).

Die Überdeckung wird als die Anzahl der im Test ausgeführten Entscheidungs-ausgänge dividiert durch die Gesamtzahl aller Entscheidungsausgänge im gesamten Testobjekt gemessen und üblicherweise als Prozentsatz angegeben. Es ist zu beachten, dass ein einzelner Testfall mehrere Entscheidungsausgänge ausführen kann.

© German Testing Board e.V.

Im Vergleich zu den nachfolgend beschriebenen modifizierten Bedingungs-/Entscheidungstests und Mehrfachbedingungstests betrachten Entscheidungstests die gesamte Entscheidung als Ganzes ungeachtet der Komplexität ihrer internen Struktur und überprüfen nur die Entscheidungsausgänge WAHR oder FALSCH.

Der Zweigtest wird oft mit dem Entscheidungstest austauschbar verwendet, da die Überdeckung aller Zweige und die Überdeckung aller Entscheidungsausgänge mit denselben Tests erzielt werden kann. Beim Zweigtest werden die Zweige im Code durchlaufen, wobei ein Zweig normalerweise als eine Kante des Kontrollflussgraphen betrachtet wird. Für Programmcode ohne Entscheidungen führt die oben beschriebene Definition der Entscheidungsüberdeckung zu einer undefinierten Überdeckung von 0/0 und zwar völlig unabhängig davon, wie viele Tests durchgeführt werden. Dagegen erzielt der Zweigtest eines einzelnen Zweigs vom Eintritts- zum Austrittspunkt (unter der Voraussetzung eines Eintritts- und Austrittspunktes) 100% Zweigüberdeckung. Um den Unterschied zwischen diesen beiden Messungen auszugleichen, verlangt ISO 29119-4, dass mindestens ein Test auch auf Code ohne Entscheidungen ausgeführt wird, um 100% Entscheidungsüberdeckung zu erzielen, weshalb 100% Entscheidungsüberdeckung und 100% Zweigüberdeckung für fast alle Programme gleichzusetzen sind. Viele Testwerkzeuge, die Überdeckungsmessungen liefern, einschließlich derer, die für das Testen sicherheitsrelevanter Systeme eingesetzt werden, verwenden einen ähnlichen Ansatz.

Anwendbarkeit

Dieser Überdeckungsgrad sollte dann in Betracht gezogen werden, wenn der zu testende Code wichtig oder sogar kritisch ist (siehe Tabellen in Abschnitt 2.8.2 für sicherheitskritische Systeme). Dieses Verfahren kann für Programmcode und für alle Modelle verwendet werden, die Entscheidungspunkte enthalten, wie z.B. Geschäftsprozessmodelle.

Einschränkungen/Schwierigkeiten

Der Entscheidungstest berücksichtigt nicht im Detail, wie eine Entscheidung mit mehreren Bedingungen getroffen wird; daher können Fehlerzustände, die bei Kombinationen von Bedingungsausgänge auftreten, unerkannt bleiben.

2.4 Modifizierter Bedingungs-/Entscheidungstest

Im Vergleich zum Entscheidungstest, der die gesamte Entscheidung als Ganzes betrachtet und die Entscheidungsausgänge zu WAHR oder FALSCH überprüft, befasst sich der modifizierte Bedingungs-/Entscheidungstest damit, wie eine Entscheidung strukturiert ist, wenn sie mehrere Bedingungen enthält. (Wenn eine Entscheidung nur aus einer einzigen atomaren Bedingung besteht, handelt es sich einfach um einen Entscheidungstest).

© German Testing Board e.V.

Jede Entscheidung besteht aus einer oder aus mehreren atomaren Bedingungen, die jeweils zu einem booleschen Wert ausgewertet werden. Diese werden logisch kombiniert, um den Entscheidungsausgang zu bestimmen. Mit diesem Verfahren wird überprüft, ob jede der atomaren Bedingungen unabhängig und korrekt in das Ergebnis der Gesamtentscheidung einfließt.

Dieses Verfahren bietet eine höhere Überdeckung als die Überdeckung beim Anweisungs- und Entscheidungstest, wenn es Entscheidungen gibt, die mehrere Bedingungen enthalten. Wenn N einzelne unabhängige atomare Bedingungen vorliegen, dann lässt sich die modifizierte Bedingungs-/Entscheidungsüberdeckung normalerweise mit N+1 Ausführungen der Entscheidung erzielen. Der modifizierte Bedingungs-/Entscheidungstest erfordert Paare von Tests, die zeigen, dass eine Änderung eines einzelnen atomaren Bedingungsausgangs das Ergebnis der Entscheidung unabhängig beeinflussen kann. Es ist zu beachten, dass ein einzelner Testfall mehrere Bedingungskombinationen ausführen kann; daher sind für die modifizierte Bedingungs-/Entscheidungsüberdeckung nicht immer N+1 separate Testfälle erforderlich.

Anwendbarkeit

Dieses Verfahren ist in der Softwareentwicklung von sicherheitskritischen Systemen der Luft- und Raumfahrtindustrie, der Automobilindustrie und von weiteren Industriezweigen weit verbreitet. Es wird beim Testen eingesetzt, wenn eine Fehlerwirkung zu einer Katastrophe führen könnte. Modifizierte Bedingungs-/Entscheidungstests können einen guten Kompromiss zwischen Entscheidungstests und Mehrfachbedingungstests darstellen (aufgrund der großen Anzahl der zu testenden Kombinationen). Dieses Verfahren ist gründlicher als der Entscheidungstest, erfordert jedoch die Überprüfung von weitaus weniger Testbedingungen als der Mehrfachbedingungstest, falls die Entscheidung mehrere atomare Bedingungen enthält.

Einschränkungen/Schwierigkeiten

Es kann kompliziert sein, die modifizierte Bedingungs-/Entscheidungsüberdeckung zu erzielen, wenn eine Variable in einer Entscheidung mit mehreren Bedingungen mehrfach vorkommt; in solchen Fällen spricht man von „gekoppelten" Bedingungen. Je nach Entscheidung kann es unmöglich sein, den Wert der gekoppelten Bedingung so zu variieren, dass sie allein zu einer Änderung des Entscheidungsausgangs führt. Ein möglicher Ansatz für den Umgang mit diesem Problem ist, dass nur nicht gekoppelte atomare Teilbedingungen durch die modifizierte Bedingungs-/Entscheidungsüberdeckung getestet werden. Ein anderer Ansatz besteht darin, jede Entscheidung mit gekoppelten Bedingungen zu analysieren.

Einige Compiler und/oder Interpreter sind so ausgelegt, dass sie komplexe Entscheidungsanweisungen im Code verkürzt auswerten. Das heißt, dass der

© German Testing Board e.V.

ausführende Code möglicherweise nicht den gesamten Ausdruck auswertet, wenn das Endergebnis der Bewertung bereits nach der Auswertung nur eines Teils des Ausdrucks feststeht. Wenn zum Beispiel die Entscheidung "A und B" ausgewertet werden soll, dann gibt es keinen Grund B auszuwerten, wenn A bereits zu FALSCH ausgewertet wurde. Kein Wert von B kann den Entscheidungsausgang ändern, so dass Ausführungszeit eingespart werden kann, wenn B nicht ausgewertet werden muss. Die verkürzte Auswertung kann die Erzielung der modifizierten Bedingungs-/Entscheidungsüberdeckung beeinträchtigen, da manche erforderlichen Tests möglicherweise nicht ausgeführt werden können. Normalerweise ist es möglich, den Compiler so zu konfigurieren, dass die verkürzte Auswertung für die Tests ausgeschaltet ist. Dies ist jedoch bei sicherheitskritischen Anwendungen, bei denen der getestete Code und der ausgelieferte Code identisch sein müssen, möglicherweise nicht zulässig.

2.5 Mehrfachbedingungstest

In seltenen Fällen kann es erforderlich sein, alle möglichen Kombinationen von atomaren Bedingungen einer Entscheidung zu testen. Diese Intensitätsstufe des Testens wird als Mehrfachbedingungstest bezeichnet. Bei N einzelnen, voneinander unabhängigen atomaren Teilbedingungen kann die vollständige Mehrfachbedingungsüberdeckung für eine Entscheidung erzielt werden, indem sie 2^N mal ausgeführt wird. Dabei ist zu beachten, dass ein einzelner Testfall mehrere Bedingungskombinationen ausführen kann und es daher nicht immer notwendig ist, 2^N separate Testfälle auszuführen, um 100% Mehrfachbedingungsüberdeckung zu erreichen.

Die erzielte Überdeckung wird als die Anzahl der im Test ausgeführten Kombinationen atomarer Bedingungen über alle im Testobjekt enthaltenen Entscheidungen gemessen und normalerweise als Prozentsatz ausgedrückt.

Anwendbarkeit

Dieses Testverfahren wird zum Testen eingebetteter Software mit hohem Risiko verwendet, von der erwartet wird, dass sie über lange Zeiträume hinweg zuverlässig ohne Abbruch läuft.

Einschränkungen/Schwierigkeiten

Da die Anzahl der Testfälle direkt aus einer Wahrheitstabelle mit allen atomaren Teilbedingungen abgeleitet werden kann, lässt sich dieser Überdeckungsgrad leicht bestimmen. Aufgrund der großen Anzahl der beim Mehrfachbedingungstest erforderlichen Testfälle ist jedoch die modifizierte Bedingungs-/Entscheidungs-überdeckung praktikabler, wenn mehrere atomare Teilbedingungen in einer Entscheidung enthalten sind.

© German Testing Board e.V.

Falls der Compiler die verkürzte Auswertung verwendet, verringert sich dadurch häufig die Anzahl der überprüfbaren Bedingungskombinationen, in Abhängigkeit von der Reihenfolge und Gruppierung der logischen Operationen, die auf den atomaren Bedingungen ausgeführt werden.

2.6 Basispfadtest

Dieses Kapitel wurde aus der Version v4.0 dieses Lehrplans entfernt.

2.7 API-Testen

Eine Programmierschnittstelle, abgekürzt API (engl. Application Programming Interface, API), ist eine definierte Schnittstelle, die es einem Programm ermöglicht, ein anderes Softwaresystem aufzurufen, das ihm einen Dienst, z.B. den Zugriff auf eine entfernte Ressource, zur Verfügung stellt. Typische Dienste sind unter anderem Webdienste, Enterprise Service Buses, Datenbanken, Mainframes und Web-UIs.

API-Testen ist eher eine Testart als ein Testverfahren. In gewisser Hinsicht ist das API-Testen dem Testen einer grafischen Benutzungsschnittstelle (engl. graphical user interface, GUI) recht ähnlich. Der Schwerpunkt liegt auf der Auswertung von Eingabewerten und zurückgegebenen Daten.

Beim Umgang mit APIs sind Negativtests häufig von entscheidender Bedeutung. Es ist möglich, dass Programmierer, die APIs verwenden um auf Dienste außerhalb ihres eigenen Codes zuzugreifen, versuchen könnten APIs in einer nicht vorgesehenen Art zu verwenden. Das bedeutet, dass eine robuste Fehlerbehandlung unerlässlich ist, um einen fehlerhaften Betrieb zu vermeiden. Möglicherweise ist kombinatorisches Testen zahlreicher Schnittstellen erforderlich, weil APIs oft in Verbindung mit anderen APIs verwendet werden, und weil eine einzige Schnittstelle mehrere Parameter enthalten kann, deren Werte auf viele Arten kombiniert werden können.

APIs sind häufig lose miteinander gekoppelt, was zu dem sehr realen Problem von verlorenen Transaktionen und Timing-Fehlern führen kann. Daher ist gründliches Testen der Wiederherstellungs- und Wiederholungsmechanismen notwendig. Unternehmen, die APIs bereitstellen, müssen sicherstellen, dass alle Services die zugesicherte Verfügbarkeit leisten; daher sind oft gründliche Zuverlässigkeitstests seitens des API-Anbieters sowie Support für die Infrastruktur notwendig.

Anwendbarkeit

API-Testen gewinnt zunehmend an Bedeutung, insbesondere in Zusammenhang mit dem Testen von Multisystemen, da diese verteilt arbeiten oder Remote-Verarbeitung einsetzen, um einen Teil der Aufgaben an andere Prozessoren auszulagern. Beispiele hierfür sind:

© German Testing Board e.V.

- Betriebssystemaufrufe
- Service-orientierte Architekturen (SOA)
- Fernaufruf von Funktionen (engl. Remote Procedure Calls, RPC)
- Webservices

Die Kapselung von Software in Containern führt zur Aufteilung eines Softwareprogramms in mehrere Container, die über Mechanismen wie die oben aufgeführten miteinander kommunizieren. API-Tests sollten auch diese Schnittstellen gezielt testen.

Einschränkungen/Schwierigkeiten

Um eine API direkt zu testen, benötigt der Technical Test Analyst normalerweise spezialisierte Werkzeuge. Da es normalerweise keine direkte grafische Benutzungsoberfläche zur API gibt, könnten Werkzeuge erforderlich sein, um einen initialen Testrahmen aufzusetzen, das Daten-Marshalling durchzuführen, die API aufzurufen und das Ergebnis zu bestimmen.

Überdeckung

API-Testen beschreibt eine Art des Testens und bezeichnet keinen bestimmten Grad an Überdeckung. API-Tests sollten zumindest die Aufrufe an die API mit realistischen gültigen Eingabewerten enthalten, sowie unerwartete Eingaben zur Überprüfung der Ausnahmebehandlung. Gründlichere API-Tests können sicherstellen, dass alle aufrufbaren Einheiten mindestens einmal ausgeführt werden bzw. alle möglichen Funktionen mindestens einmal aufgerufen werden.

Representational State Transfer (REST) ist ein Architekturstil. RESTful-Webservices ermöglichen anfragenden Systemen den Zugriff auf Webressourcen unter Verwendung eines einheitlichen Satzes von zustandslosen Operationen. Es gibt mehrere Überdeckungskriterien für RESTful-Web-APIs, den de-facto-Standard für Softwareintegration [Web-7]. Sie können in zwei Gruppen eingeteilt werden: Überdeckungskriterien für Eingaben und Überdeckungskriterien für Ausgaben. Bei den Kriterien für Eingaben kann unter anderem die Ausführung aller möglichen API-Operationen, die Verwendung aller möglichen API-Parameter und die Überdeckung von Sequenzen von API-Operationen erforderlich sein. Bei den Ausgabekriterien kann unter anderem die Generierung aller korrekten und fehlerhaften Statuscodes erforderlich sein, sowie die Generierung von Antworten, die Ressourcen mit allen möglichen Eigenschaften (oder allen Arten von Eigenschaften) enthalten.

Fehlerarten

Die Fehlerzustände, die beim API-Test gefunden werden können, sind recht unterschiedlich. Häufig geht es um Schnittstellenprobleme, sowie um Probleme mit der

© German Testing Board e.V.

Datenbehandlung, dem Timing, dem Verlust oder der Duplizierung von Transaktionen, sowie um Probleme bei der Ausnahmebehandlung.

2.8 White-Box-Testverfahren auswählen

Das ausgewählte White-Box-Testverfahren wird normalerweise mittels des benötigten Überdeckungsgrades spezifiziert, der durch die Anwendung des Testverfahrens erzielt wird. Wenn beispielsweise 100% Anweisungsüberdeckung gefordert ist, wird normalerweise der Anweisungstest eingesetzt. In der Regel werden jedoch zuerst Black-Box-Testverfahren durchgeführt, dann wird der Überdeckungsgrad gemessen, und ein White-Box-Testverfahren wird nur eingesetzt, wenn der erforderliche White-Box-Überdeckungsgrad nicht erzielt wurde. In manchen Situationen können White-Box-Tests weniger formal eingesetzt werden, um Hinweise darauf zu erhalten, wo die Überdeckung möglicherweise erhöht werden muss (z.B. durch die Erstellung zusätzlicher Tests, wenn die White-Box-Überdeckung besonders niedrig ist). Für eine solche informelle Messung der Überdeckung sind normalerweise Anweisungstests ausreichend.

Bei der Festlegung der zu erzielenden White-Box-Überdeckung ist es gängige Praxis, diese nur mit 100 % anzugeben. Wenn nämlich eine geringere Überdeckung vorgeschrieben ist, bedeutet dies meist, dass genau jene Teile des Codes, die am schwierigsten zu testen sind, nicht in den Tests berücksichtigt werden, obwohl genau diese Teile normalerweise am komplexesten und fehleranfälligsten sind. Wenn also beispielsweise eine Überdeckung von 80% gefordert und erzielt wird, kann dies bedeuten, dass der Code, der die meisten zu findenden Fehlerzustände enthält, nicht getestet wird. Aus diesem Grund werden die White-Box-Überdeckungskriterien in Standards fast immer mit 100 % angegeben. Die strikte Auslegung der Überdeckung hat manchmal dazu geführt, dass der geforderte Grad an Überdeckung nicht erzielbar war. Die in ISO 29119-4 spezifizierten Überdeckungen bieten jedoch die Möglichkeit, nicht durchführbare Überdeckungselemente aus den Berechnungen auszuklammern, so dass 100% Überdeckung ein erreichbares Ziel darstellt.

Bei der Spezifikation der geforderten White-Box-Überdeckung für ein Testobjekt reicht es zudem aus, diese nur für ein einziges Überdeckungskriterium zu spezifizieren. Es ist z.B. nicht notwendig, sowohl 100% Anweisungsüberdeckung als auch 100% modifizierte Bedingungs-/Entscheidungsüberdeckung zu fordern. Mit Endekriterien von 100 % ist es möglich, einige Endekriterien in einer Hierarchie zu betrachten, die zeigt, dass Überdeckungskriterien andere Überdeckungskriterien umfassen. Ein Überdeckungskriterium umfasst ein anderes, wenn für alle Komponenten und deren Spezifikationen jeder Satz von Testfällen, der das erste Kriterium erfüllt, auch das zweite erfüllt. Beispiel: Die Zweigüberdeckung umfasst die Anweisungsüberdeckung, denn wenn 100 % Zweigüberdeckung erreicht ist, ist garantiert auch 100 % Anweisungsüberdeckung erreicht. Im Hinblick auf die in diesem Lehrplan beschriebenen White-Box-Testverfahren kann gesagt werden, dass die Zweig- und

© German Testing Board e.V.

Entscheidungsüberdeckung die Anweisungsüberdeckung umfassen, dass die modifizierte Bedingungs-/Entscheidungsüberdeckung die Entscheidungs- und Zweigüberdeckung umfasst, und dass die Mehrfachbedingungsüberdeckung die modifizierte Bedingungs-/Entscheidungsüberdeckung umfasst (wenn wir die Zweigüberdeckung und die Entscheidungsüberdeckung von 100 % als gleich betrachten, lässt sich sagen, dass sie sich gegenseitig umfassen).

Es ist durchaus üblich, dass bei der Festlegung der zu erzielenden White-Box-Überdeckungsgrade für ein System unterschiedliche Grade für die verschiedenen Teile des Systems definiert werden. Dies liegt daran, dass verschiedene Teile eines Systems in unterschiedlichem Maße zum Risiko beitragen. Beispiel: Bei Avionik-Systemen wird den Teilsystemen für die Bordunterhaltung ein geringeres Risiko zugeordnet als den Teilsystemen für die Flugsteuerung. Das Testen von Schnittstellen ist für alle Arten von Systemen üblich und ist bei sicherheitsrelevanten Systemen in der Regel für alle Sicherheitsintegritätsstufen erforderlich (weitere Informationen über Sicherheitsintegritätsstufen, siehe Abschnitt 2.8.2). Der im API-Test zu erzielende Überdeckungsgrad erhöht sich normalerweise mit dem damit verbundenen Risiko (z.B. kann das mit einer öffentlichen Schnittstelle verbundene höhere Risiko gründlichere API-Tests erforderlich machen).

Die Auswahl des zu verwendenden White-Box-Testverfahrens richtet sich im Allgemeinen nach der Art des Testobjekts und den damit verbundenen wahrgenommenen Risiken. Wenn das Testobjekt als sicherheitsrelevant angesehen wird (d.h. eine Fehlerwirkung könnte Menschen oder die Umwelt schädigen), dann sind regulatorische Standards anzuwenden, die den erforderlichen White-Box-Überdeckungsgrad definieren (siehe Abschnitt 2.8.2). Wenn das Testobjekt nicht sicherheitsbezogen ist, erfolgt die Auswahl der zu erzielenden White-Box-Überdeckungsgrade eher subjektiv, sollte aber dennoch weitgehend auf den wahrgenommenen Risiken basieren, wie in Abschnitt 2.8.1 beschrieben

2.8.1 Nicht sicherheitsbezogene Systeme

Bei der Auswahl von White-Box-Testverfahren für nicht sicherheitsbezogene Systeme werden normalerweise die folgenden Faktoren berücksichtigt (nicht nach Priorität aufgelistet):

- Vertrag – Wenn laut Vertrag ein bestimmter Überdeckungsgrad erzielt werden muss, kann das Nichterreichen dieses Überdeckungsgrads zu einem Vertragsbruch führen.
- Kunde – Wenn der Kunde einen bestimmten Überdeckungsgrad verlangt, z.B. als Teil der Testplanung, dann kann das Nichterreichen dieses Überdeckungsgrads zu Problemen mit dem Kunden führen

© German Testing Board e.V.

- Regulatorischer Standard – In einigen Branchen (z.B. im Finanzsektor) gilt für erfolgskritische Systeme ein branchenspezifischer Standard, der die erforderlichen White-Box-Überdeckungskriterien definiert. Siehe Abschnitt 2.8.2 bezüglich regulatorischer Standards für sicherheitsbezogene Systeme.
- Teststrategie – Wenn in der Teststrategie des Unternehmens Anforderungen an die Codeüberdeckung bei White-Box-Tests festgelegt sind, kann eine Nichtübereinstimmung mit der Unternehmensstrategie eine Missbilligung seitens der höheren Führungsebene zur Folge haben.
- Programmierstil – Wenn der Code keine Mehrfachbedingungen innerhalb von Entscheidungen enthält, wäre es verschwenderisch, für White-Box-Tests die modifizierte Bedingungs-/Entscheidungsüberdeckung oder Mehrfachbedingungsüberdeckung vorzuschreiben.
- Historische Fehlerinformationen – Wenn historische Daten die Effektivität eines bestimmten zu erzielenden Überdeckungsgrades nahelegen, dass dieser für das Testobjekt angemessen wäre, dann wäre es riskant, die verfügbaren Daten zu ignorieren. Die entsprechenden Daten können innerhalb des Projekts, der Organisation oder der Branche verfügbar sein.
- Fähigkeiten und Erfahrung – Wenn die Tester, die für die Durchführung der Tests zur Verfügung stehen, nicht genügend Erfahrung und Kenntnisse in einem bestimmten White-Box-Verfahren haben, kann die Auswahl dieses Verfahrens zu Missverständnissen führen und ein unnötiges Risiko erzeugen.
- Werkzeuge – Die White-Box-Überdeckung kann in der Praxis nur mit Hilfe von Überdeckungswerkzeugen gemessen werden. Wenn keine Werkzeuge zur Messung der vorgegebenen Überdeckung zur Verfügung stehen, dann würde die Festlegung dieses Maßes zur Zielerreichung ein hohes Risiko mit sich bringen.

Bei der Auswahl von White-Box-Tests für nicht sicherheitsbezogene Systeme hat der Technical Test Analyst mehr Spielraum um die geeignete Überdeckung zu empfehlen als dies bei sicherheitsbezogenen Systemen der Fall ist. Solche Entscheidungen sind meist ein Kompromiss zwischen den wahrgenommenen Risiken und den Kosten, Ressourcen und dem Zeitaufwand für die Behandlung dieser Risiken durch White-Box-Tests. In manchen Situationen können andere Maßnahmen, die durch andere Testarten oder auf andere Weise (z.B. durch andere Entwicklungsansätze) umgesetzt werden, möglicherweise angemessener sein.

2.8.2 Sicherheitsbezogene Systeme

Beim Testen von Software, die Teil eines sicherheitsbezogenen Systems ist, muss normalerweise ein Standard eingehalten werden, der die zu erreichende Überdeckung festlegt Diese Standards schreiben normalerweise die Durchführung einer Gefährdungs- und Risikoanalyse für das System vor; die darin identifizierten Risiken

© German Testing Board e.V.

dienen dazu, den verschiedenen Teilen des Systems Integritätsstufen zuzuordnen. Für die einzelnen Integritätsstufen ist die jeweils erforderliche Überdeckungsgrad definiert.

IEC 61508 (Funktionale Sicherheit programmierbarer, elektronischer, sicherheitsbezogener Systeme [IEC 61508]) ist eine internationale Normenserie, die für solche Zwecke verwendet wird. Theoretisch könnte sie für jedes sicherheitsbezogene System angewendet werden, jedoch haben einige Branchen ihre eigenen Varianten des Standards erstellt (z.B. gilt ISO 26262 [ISO 26262] für Softwaresysteme in der Automobilindustrie), während andere Industriezweige einen eigenen branchenspezifischen Standard erstellt haben (z.B. DO-178C [DO-178C] für Softwaresysteme in der Luftfahrt). Weitere Informationen zu ISO 26262 finden Sie im ISTQB® Automotive Software Tester Lehrplan [CT_AuT_SYL].

IEC 61508 definiert vier Sicherheitsintegritätsstufen (engl. Safety Integrity Level, SIL), die jeweils als relatives Maß für die risikomindernde Wirksamkeit einer Sicherheitsfunktion definiert sind und mit der Häufigkeit und dem Schweregrad der wahrgenommenen Gefährdungen korreliert sind. Wenn ein Testobjekt eine sicherheitsbezogene Funktion ausführt, bedeutet dies, dass eine Fehlerwirkung mit einem erhöhten Risiko daherkommt, so dass das Testobjekt eine höhere Zuverlässigkeit aufweisen sollte. Die folgende Tabelle zeigt die mit den SILs verbundenen Zuverlässigkeitsstufen. Es ist zu beachten, dass die Zuverlässigkeitsstufe für SIL 4 für den Fall des Dauerbetriebs extrem hoch ist, da es einer durchschnittlichen Zeitspanne zwischen aufeinanderfolgenden Fehlerwirkungen (Mean Time Between Failure, MTBF) von mehr als 10.000 Jahren entspricht.

IEC 61508 SIL	**Dauerbetrieb (Wahrscheinlichkeit der gefahrbringenden Fehlerwirkung pro Stunde)**	**Auf Anforderung (Wahrscheinlichkeit der gefahrbringenden Fehlerwirkung bei Anforderung)**
1	$\geq 10^{-6}$ bis $< 10^{-5}$	$\geq 10^{-2}$ bis $< 10^{-1}$
2	$\geq 10^{-7}$ bis $< 10^{-6}$	$\geq 10^{-3}$ bis $< 10^{-2}$
3	$\geq 10^{-8}$ bis $< 10^{-7}$	$\geq 10^{-4}$ bis $< 10^{-3}$
4	$\geq 10^{-9}$ bis $< 10^{-8}$	$\geq 10^{-5}$ bis $< 10^{-4}$

© German Testing Board e.V.

In der folgenden Tabelle sind die Empfehlungen für die im White-Box-Test zu erzielende Überdeckungsgrade für die einzelnen SILs aufgeführt. Wenn "sehr empfohlen" angegeben ist, gilt das Erzielen des aufgeführten Überdeckungsgrads in der Praxis normalerweise als obligatorisch. Wird dagegen nur "empfohlen" angegeben, wird das Erzielen des aufgeführten Überdeckungsgrads von vielen Praktikern als optional betrachtet und mit einer geeigneten Begründung vermieden. So wird ein Testobjekt, das als SIL 3 eingestuft ist, normalerweise so getestet, dass es eine 100%ige Zweigüberdeckung erreicht (es erreicht automatisch eine 100%ige Anweisungsabdeckung, wie die Rangfolge zeigt).

IEC 61508 SIL	100% Anweisungs-überdeckung	100% Zweig-überdeckung	100% modifizierte Bedingungs-/Entscheidungsüberdeckung
1	Empfohlen	Empfohlen	Empfohlen
2	Sehr empfohlen	Empfohlen	Empfohlen
3	Sehr empfohlen	Sehr empfohlen	Empfohlen
4	Sehr empfohlen	Sehr empfohlen	Sehr empfohlen

Es ist zu beachten, dass die oben genannten SILs und Anforderungen an die Überdeckung aus der IEC 61508 in der ISO 26262 anders sind, und in der DO-178C noch einmal anders.

© German Testing Board e.V.

3. Statische und dynamische Analyse – 240 Min.

Schlüsselbegriffe

Datenflussanalyse, Definition-Verwendungs-Paar, dynamische Analyse, Kontrollflussanalyse, Speicherleck, statische Analyse, wilder Zeiger, zyklomatische Komplexität

Lernziele für "Statische und dynamische Analyse"

3.2 Statische Analyse

TTA-3.2.1 (K3) Die Kontrollflussanalyse anwenden, um zu ermitteln, ob der Programmcode Anomalien im Kontrollfluss aufweist und um die zyklomatische Komplexität zu messen

TTA-3.2.2 (K3) Die Datenflussanalyse anwenden, um zu ermitteln, ob der Programmcode Anomalien im Datenfluss aufweist

TTA-3.2.3 (K3) Möglichkeiten vorschlagen, wie die Wartbarkeit von Programmcode durch statische Analyse verbessert werden kann

TTA-3.2.4 wurde aus der Version v4.0 dieses Lehrplans entfernt.

3.3 Dynamische Analyse

TTA-3.3.1 (K3) Die dynamische Analyse anwenden, um ein bestimmtes Ziel zu erreichen

3.1 Einführung

Die statische Analyse (siehe Abschnitt 3.2) ist eine Art des Testens, die durchgeführt wird, ohne die Software auszuführen. Die Qualität der Software wird von einem Werkzeug oder einer Person hinsichtlich ihrer Form, Struktur, ihres Inhalts oder ihrer Dokumentation bewertet. Diese statische Sicht auf die Software ermöglicht eine detaillierte Analyse, ohne dass die für die Ausführung von Testfällen erforderlichen Daten und Vorbedingungen erstellt werden müssen.

Neben der statischen Analyse umfassen die statischen Testverfahren auch verschiedene Reviewarten. Die für den Technical Test Analysten relevanten Reviewarten werden in Kapitel 5 behandelt.

Die dynamische Analyse (siehe Abschnitt 3.3) erfordert die tatsächliche Ausführung des Codes und zielt auf die Aufdeckung von Fehlerzuständen ab, die einfacher zu finden sind, wenn die Software ausgeführt wird (z.B. Speicherlecks). Wie auch die statische Analyse erfolgt die dynamische Analyse durch Werkzeuge oder durch eine

© German Testing Board e.V.

Person, die die Ausführung des Systems überwachen und auf bestimmte Indikatoren achten wie z.B. einen schnellen Anstieg der Speichernutzung.

3.2 Statische Analyse

Ziel der statischen Analyse ist es, tatsächliche oder potenzielle Fehlerzustände im Code und in der Systemarchitektur zu entdecken, sowie deren Wartbarkeit zu verbessern.

3.2.1 Kontrollflussanalyse

Die Kontrollflussanalyse ist ein statisches Analyseverfahren, bei dem die Schritte, die ein Programm durchläuft, mit Hilfe eines Kontrollflussgraphen analysiert werden, in der Regel mit Hilfe eines Werkzeugs. Es gibt eine Reihe von Anomalien in Systemen, die mit Hilfe dieses Verfahrens gefunden werden können. Dazu gehören schlecht entworfene Schleifen (z.B. Schleifen mit mehreren Eingangspunkten oder Schleifen, die nicht beendet werden), unklare/inkorrekt deklarierte Ziele von Funktionsaufrufen in bestimmten Sprachen, inkorrekte Ablaufsequenzen, unerreichbarer Code, nicht aufgerufene Funktionen usw.

Die Kontrollflussanalyse kann zur Bestimmung der zyklomatischen Komplexität verwendet werden. Der für die zyklomatische Komplexität ermittelte Wert ist eine positive ganze Zahl, die die Anzahl der linear unabhängigen Kontrollflusspfade in einem stark zusammenhängenden Kontrollflussgraphen angibt.

Die zyklomatische Komplexität wird allgemein als ein Indikator für die Komplexität einer Komponente verwendet. Nach der Theorie von Thomas McCabe [McCabe 76] gilt, dass, je komplexer ein System ist, desto schwieriger ist dessen Wartung und desto mehr Fehlerzustände sind darin enthalten. Diese Korrelation zwischen Komplexität und der Anzahl der darin enthaltenen Fehlerzustände wurde in vielen Studien festgestellt. Komponenten, für die eine höhere Komplexität ermittelt wurde, sollten auf ein mögliches Refactoring hin überprüft werden, z.B. Aufteilung in mehrere Komponenten.

3.2.2 Datenflussanalyse

Die Datenflussanalyse umfasst eine Vielzahl von Verfahren, die Informationen über die Verwendung von Variablen in einem System sammeln. Dabei wird der Lebenszyklus von Variablen entlang eines Kontrollflusspfades untersucht (d.h. wo sie deklariert, definiert, verwendet und zerstört werden), da potenzielle Anomalien identifiziert werden können, wenn die richtige Reihenfolge der Aktionen nicht eingehalten wird [Beizer90].

© German Testing Board e.V.

Ein gebräuchliches Verfahren unterteilt die Verwendung einer Variablen in eine von drei atomaren Aktionen:

- Die Variable wird definiert, deklariert oder initialisiert (z.B. x:=3)
- Die Variable wird verwendet oder gelesen (z.B. if x > temp)
- Die Variable wird gelöscht oder zerstört oder ist nicht mehr erreichbar (z.B. Text_Datei_1.schließen, Schleifenkontrollvariable (i) beim Verlassen der Schleife)

Zu den Reihenfolgen solcher Aktionen, die auf potenzielle Anomalien hinweisen, gehören:

- Definition einer Variable, gefolgt von einer weiteren Definition oder Zerstörung, ohne dass die Variable zwischenzeitlich verwendet wurde
- Definition einer Variable ohne anschließende Zerstörung (was z.B. bei dynamisch zugewiesenen Variablen zu einem Speicherleck führen kann)
- Verwendung oder Zerstörung einer Variable, bevor die Variable definiert wurde
- Verwendung oder Zerstörung einer Variable, nachdem die Variable zerstört wurde

Je nach Programmiersprache können einige dieser Anomalien vom Compiler erkannt werden; allerdings könnte ein separates statisches Analysewerkzeug erforderlich sein, um die Anomalien im Datenfluss zu erkennen. So ist beispielsweise eine Neudefinition ohne zwischenzeitliche Verwendung in den meisten Programmiersprachen zulässig und kann sogar absichtlich programmiert sein, würde aber von einem Datenflussanalysewerkzeug als potenzielle Anomalie identifiziert, die überprüft werden sollte.

Die Verwendung von Kontrollflusspfaden zur Bestimmung der Reihenfolge der Aktionen für eine Variable kann zur Meldung potenzieller Anomalien führen, die in der Praxis nicht auftreten können. Beispielsweise können statische Analysewerkzeuge nicht immer erkennen, ob ein Kontrollflusspfad durchführbar ist, da manche Pfade nur auf Grundlage der den Variablen zur Laufzeit zugewiesenen Werte bestimmt werden. Es gibt auch bestimmte Probleme bei der Datenflussanalyse, die von den Werkzeugen nur schwer zu identifizieren sind, wenn die analysierten Daten Teil von Datenstrukturen mit dynamisch zugewiesenen Variablen sind, wie z.B. Datensätze und Arrays. Statische Analysewerkzeuge haben auch Schwierigkeiten, potenzielle Datenflussanomalien zu erkennen, wenn Variablen von parallel ablaufenden Ausführungspfaden (Threads) in einem Programm gemeinsam genutzt werden, da die Reihenfolge der Aktionen auf Daten schwer vorherzusagen ist.

Im Gegensatz zur Datenflussanalyse, die ein statischer Test ist, handelt es sich beim Datenflusstest um einen dynamischen Test, bei dem Testfälle erstellt werden, um

© German Testing Board e.V.

"Definition-Verwendungs-Paare" im Programmcode zu testen. Datenflusstests verwenden teilweise die gleichen Konzepte wie die Datenflussanalyse, da diese Definition-Verwendungs-Paare Kontrollflusspfade zwischen einer Definition und der nachfolgenden Verwendung einer Variablen in einem Programm sind.

3.2.3 Wartbarkeit durch statische Analyse verbessern

Die statische Analyse kann unterschiedlich eingesetzt werden, um die Wartbarkeit von Programmcode, Softwarearchitektur und Webseiten zu verbessern.

Grundsätzlich gilt, dass schlecht geschriebener, unkommentierter und unstrukturierter Code schwieriger zu warten ist. Es kann für die Entwickler aufwändiger sein, Fehlerzustände im Code zu lokalisieren und zu analysieren. Ferner führen Änderungen des Codes zur Fehlerbehebung oder zum Hinzufügen eines Features wahrscheinlich dazu, dass weitere Fehlerzustände eingebaut werden.

Die statische Analyse wird dazu genutzt, die Einhaltung von Programmierstandards und -richtlinien im vorhandenen Programmcode zu verifizieren. Wird dabei nicht konformer Code erkannt, kann dieser überarbeitet werden, um seine Wartbarkeit zu verbessern. Diese Programmierstandards und -richtlinien beschreiben die geforderten Kodierungs- und Entwurfspraktiken, wie z.B. Namenskonventionen, Kommentierung, Quelltexteinrückung und Modularisierung. Es ist zu beachten, dass statische Analysewerkzeuge im Allgemeinen eher Warnungen als Fehlerzustände anzeigen. Diese Warnungen (z.B. bezüglich der Komplexität des Codes) können ausgegeben werden, auch wenn die Syntax des Programmcodes korrekt ist.

Ein modularer Aufbau verbessert in der Regel die Wartbarkeit des Codes. Statische Analysewerkzeuge unterstützen die Entwicklung von modularem Code auf folgende Weise:

- Sie suchen nach Wiederholungen im Code. Diese Codeabschnitte bieten sich an für ein Refactoring zu Komponenten (wenngleich sich die Aufrufe der Komponenten auf die Laufzeit auswirken können, was bei Echtzeitsystemen ein Problem sein könnte).
- Sie erzeugen Metriken, die wertvolle Indikatoren für die Codemodularisierung sind. Dazu gehören u.a. Metriken für die Kopplung und Kohäsion. Ein System, das eine gute Wartbarkeit aufweisen soll, hat eher weniger gekoppelte Komponenten (d.h. Komponenten, die während der Ausführung des Codes aufeinander angewiesen sind) und ein hohes Maß an Kohäsion (der Grad, zu dem Komponenten in sich geschlossen und nur für die Durchführung einer einzigen Aufgabe vorgesehen sind).
- Bei objektorientiertem Code zeigen sie an, wo abgeleitete Klassen eventuell zu viel oder zu wenig Sichtbarkeit zu übergeordneten (Eltern-)Klassen haben.

© German Testing Board e.V.

- Sie heben Bereiche im Programmcode oder in der Systemarchitektur hervor, die ein hohes Maß an struktureller Komplexität aufweisen.

Auch die Wartung von Webseiten kann durch statische Analysewerkzeuge unterstützt werden. Hier geht es darum, zu prüfen, ob die Baumstruktur der Webseite ausgewogen ist oder ob ein Ungleichgewicht vorliegt, das die folgenden Konsequenzen haben könnte:

- Schwierigere Testaufgaben
- Höherer Arbeitsaufwand für die Wartung

Neben der Bewertung der Wartbarkeit können statische Analysewerkzeuge den zugrunde liegenden Implementierungscode von Webseiten auf mögliche Gefährdung durch IT-Sicherheitsschwachstellen prüfen, wie Code-Einschleusung (code injection), Cookie-Sicherheit, webseitenübergreifendes Skripten (Cross-Site-Scripting, XSS), Ressourcenmanipulation und SQL-Einschleusung (SQL injection). Weitere Einzelheiten sind in Abschnitt 4.3 und im Lehrplan Sicherheitstester [CT_SEC_SYL] enthalten.

3.3 Dynamische Analyse

3.3.1 Überblick

Die dynamische Analyse wird eingesetzt, um Fehlerwirkungen zu erkennen, deren Symptome erst bei der Ausführung des Codes sichtbar werden. Beispiel: Mögliche Speicherlecks können durch statische Analyse erkannt werden (es wird Code gefunden, der Speicher zuweist, aber nie Speicher freigibt), sind mit der dynamischen Analyse jedoch sehr leicht erkennbar.

Fehlerwirkungen, die nicht ohne Weiteres reproduzierbar sind, können erhebliche Konsequenzen für den Testaufwand, die Freigabe der Software oder deren produktiven Einsatz haben. Die Ursache solcher Fehlerwirkungen können Speicher- oder Ressourcenlecks, ein inkorrekter Einsatz von Zeigern und andere Korruptionen (z.B. des Systemstacks) sein [Kaner02]. Diese Art von Fehlerwirkungen können zu einer graduellen Verschlechterung der Systemleistung oder sogar zu System-abstürzen führen. Daher muss die Teststrategie die mit solchen Fehlern verbundenen Risiken berücksichtigen und gegebenenfalls zur Risikominderung eine dynamische Analyse durchzuführen (normalerweise mit Hilfe von Testwerkzeugen). Da diese Fehlerwirkungen oftmals am teuersten sind was Aufdeckung und Behebung betrifft, wird empfohlen, mit der dynamischen Analyse frühzeitig im Projekt zu beginnen.

Die dynamische Analyse kann angewendet werden, um Folgendes zu erreichen:

- Fehlerwirkungen verhindern, indem Speicherlecks (siehe Abschnitt 3.3.2) und wilde Zeiger (siehe Abschnitt 3.3.3) aufgedeckt werden

© German Testing Board e.V.

- Systemausfälle analysieren, die nicht leicht reproduzierbar sind
- Netzwerkverhalten bewerten
- Verbesserung der Systemleistung durch die Verwendung von Code-Profilern, die Informationen über das Laufzeitverhalten des Systems liefern, um fundierte Änderungen zu ermöglichen

Die dynamische Analyse kann in jeder Teststufe durchgeführt werden. Es sind technische Fähigkeiten und Systemkenntnisse zur Durchführung der folgenden Aufgaben erforderlich:

- Spezifizieren der Testziele für die dynamische Analyse
- Bestimmen des geeigneten Zeitpunkts für Beginn und Ende der Analyse
- Analysieren der Testergebnisse

Dynamische Analysewerkzeuge können selbst dann eingesetzt werden, wenn der Technical Test Analyst nur wenig technische Fähigkeiten besitzt. Die verwendeten Werkzeuge erstellen meist umfangreiche Protokolle, die von Personen mit den erforderlichen technischen und analytischen Kompetenzen analysiert werden können.

3.3.2 Speicherlecks aufdecken

Ein Speicherleck tritt auf, wenn Speicherbereiche (RAM) einem Programm zugewiesen werden, aber anschließend nicht wieder freigegeben werden, wenn sie nicht mehr benötigt werden. Dieser Speicherbereich bleibt belegt und steht nicht zur Wiederverwendung zur Verfügung. Wenn dies häufig vorkommt oder wenn wenig Speicher vorhanden ist, kann der nutzbare Speicherplatz ausgehen. Traditionell lag die Manipulation der Speicherbereiche in der Verantwortung der Programmierer. Dynamisch belegter Speicherplatz musste vom Programm wieder freigegeben werden, um ein Speicherleck zu vermeiden. Viele moderne Programmierumgebungen sind mit automatischen oder halbautomatischen Funktionen zur Speicherbereinigung ausgestattet, die dafür sorgen, dass belegter Speicherplatz ohne direktes Eingreifen des Programmierers wieder freigegeben wird. Es kann sehr schwierig sein, Speicherlecks zu identifizieren, wenn belegter Speicherplatz durch automatische Speicherbereinigung freigegeben werden sollte.

Speicherlecks verursachen in der Regel erst nach einiger Zeit Probleme, nämlich dann, wenn ein erheblicher Teil des Speichers verloren gegangen ist und nicht mehr zur Verfügung steht. Wenn Software neu installiert oder das System neu gestartet wird, wird der Speicher neu zugewiesen, so dass Speicherlecks nicht auffallen. Beim Testen können zum Beispiel häufige Speicherzuweisungen die Erkennung von Speicherlecks leicht verhindern. Daher werden die negativen Auswirkungen von Speicherlecks oft erst bemerkt, wenn das Programm in produktiv gegangen ist.

© German Testing Board e.V.

Das primäre Symptom eines Speicherlecks ist eine kontinuierliche Verschlechterung der Antwortzeiten des Systems, die letztlich zu einem Systemausfall führen kann. Solche Fehlerwirkungen können zwar durch einen Neustart (Rebooten) des Systems gelöst werden, doch ist dies bei manchen Systemen nicht immer praktikabel oder gar möglich.

Viele dynamische Analysewerkzeuge identifizieren Bereiche im Code, in denen Speicherlecks vorkommen, so dass diese korrigiert werden können. Mit einem einfachen Speichermonitor lässt sich ebenfalls herausfinden, ob der verfügbare Speicherplatz im Laufe der Zeit abnimmt, auch wenn hier noch eine Folgeanalyse erforderlich wäre, um die genaue Ursache zu ermitteln.

3.3.3 Wilde Zeiger aufdecken

Wilde Zeiger in einem Programm sind Zeiger, die nicht mehr korrekt sind und nicht mehr benutzt werden dürfen. Sie entstehen, wenn es die Objekte oder die Funktionen, auf die sie zeigen, nicht mehr gibt, oder wenn sie nicht auf den vorgesehenen Speicherbereich verweisen, sondern auf einen Speicherbereich außerhalb des zugewiesenen Arrays. Wenn ein Programm wilde Zeiger verwendet, kann dies verschiedene Folgen haben:

- Das Programm kann seine erwartete Leistung erbringen. Dies kann passieren, wenn der Zeiger auf einen Speicherbereich zeigt, der quasi frei ist und derzeit vom Programm nicht benutzt wird, und/oder der einen plausiblen Wert enthält.
- Das Programm kann abstürzen. Dies kann passieren, wenn der Zeiger auf einen Systembereich zeigt, der dann falsch verwendet wird, aber eigentlich für den Betrieb der Software benötigt wird (z.B. das Betriebssystem).
- Das Programm funktioniert nicht korrekt, weil es auf benötigte Objekte nicht zugreifen kann. Unter diesen Bedingungen kann das Programm zwar weiterhin funktionieren, gibt aber Fehlermeldungen aus.
- Daten in den Speicherbereichen können durch den wilden Zeiger zerstört oder unbrauchbar werden, sodass dann inkorrekte Werte verwendet werden (was auch ein IT-Sicherheitsrisiko darstellen kann).

Änderungen an der Speichernutzung des Programms (z.B. ein neuer Build nach einer Softwareänderung) können jede dieser vier Folgen auslösen. Dies ist vor allem dann kritisch, wenn das System zunächst wie erwartet funktioniert, obwohl es fehlerhafte Zeiger enthält, aber nach einer Softwareänderung unerwartet abstürzt (vielleicht sogar im Produktivbetrieb). Werkzeuge können fehlerhafte Zeiger identifizieren, die vom Programm verwendet werden, unabhängig von deren Folgen für die Programmausführung. Manche Betriebssysteme haben integrierte Funktionen zur Überwachung von Speicherzugriffsverletzungen während der Laufzeit. So kann das Betriebssystem beispielsweise eine Ausnahme melden, wenn eine Anwendung

© German Testing Board e.V.

versucht, auf einen Speicherbereich zuzugreifen, die außerhalb des zulässigen Speicherbereichs dieser Anwendung ist.

3.3.4 Performanz des Systems analysieren

Die dynamische Analyse ist nicht nur zur Erkennung von Fehlerwirkungen und zur Lokalisierung der zugehörigen Fehlerzustände geeignet. Bei der dynamischen Analyse der Systemperformanz helfen Werkzeuge, Performanzengpässe zu identifizieren und eine ganze Reihe von Performanzmetriken zu erzeugen, mit denen die Entwickler die Systemperformanz optimieren können. Beispiel: Es können Informationen darüber geliefert werden, wie oft eine Komponente während der Ausführung aufgerufen wird. Komponenten, die häufig aufgerufen werden, bieten sich für eine Performanzverbesserung geradezu an. Oft trifft hier das Paretoprinzip zu: ein Programm verwendet einen überproportionalen Teil (80%) seiner Laufzeit in einer kleinen Zahl (20%) von Komponenten [Andrist20].

Die dynamische Analyse der Systemperformanz wird häufig während des Systemtests durchgeführt, kann aber auch schon in früheren Testphasen erfolgen, wenn ein einzelnes Teilsystems mit Hilfe eines Testrahmens getestet wird. Weitere Informationen sind im Lehrplan Performanztest [CT_PT_SYL] enthalten.

© German Testing Board e.V.

4. Qualitätsmerkmale bei technischen Tests – 345 Min.

Schlüsselbegriffe

Analysierbarkeit, Anpassbarkeit, Austauschbarkeit, Authentizität, Fehlertoleranz, Installierbarkeit, Integrität, IT-Sicherheit, Kapazität, Koexistenz, Kompatibilität, Modifizierbarkeit, Modularität, Nichtabstreitbarkeit, Nutzungsprofil, Performanz, Qualitätsmerkmal, Reife, Ressourcennutzung, Testbarkeit, Übertragbarkeit, Verfügbarkeit, Vertraulichkeit, Wartbarkeit, Wiederherstellbarkeit, Wiederverwendbarkeit, Zeitverhalten, Zurechenbarkeit, Zuverlässigkeit, Zuverlässigkeitswachstumsmodell

Lernziele für "Qualitätsmerkmale bei technischen Tests"

4.2 Allgemeine Planungsaspekte

TTA-4.2.1 (K4) Für ein bestimmtes Szenario die nicht-funktionalen Anforderungen analysieren und dafür die entsprechenden Inhalte des Testkonzepts erstellen

TTA-4.2.2 (K3) Für ein bestimmtes Produktrisiko die spezifische nicht-funktionale Testart (bzw. Testarten) definieren, die am besten geeignet ist (sind)

TTA-4.2.3 (K2) Verstehen und erläutern, in welchen Phasen des Softwareentwicklungslebenszyklus nicht-funktionales Testen typischerweise erfolgen sollte

TTA-4.2.4 (K3) Für ein vorgegebenes Szenario festlegen, welche Fehlerarten Sie durch die Anwendung nicht-funktionaler Testarten erwarten aufzudecken

4.3 IT-Sicherheitstest

TTA-4.3.1 (K2) Die Gründe für die Einbeziehung von IT-Sicherheitstests in eine Testvorgehensweise erläutern

TTA-4.3.2 (K2) Die wichtigsten Aspekte erläutern, die bei der Planung und Spezifizierung von IT-Sicherheitstests zu berücksichtigen sind

4.4 Zuverlässigkeitstest

TTA-4.4.1 (K2) Die Gründe für die Einbeziehung von Zuverlässigkeitstests in eine Testvorgehensweise erläutern

TTA-4.4.2 (K2) Die wichtigsten Aspekte erläutern, die bei der Planung und Spezifizierung von Zuverlässigkeitstests zu berücksichtigen sind

4.5 Performanztest

© German Testing Board e.V.

TTA-4.5.1 (K2) Die Gründe für die Einbeziehung von Performanztests in eine Testvorgehensweise erläutern

TTA-4.5.2 (K2) Die wichtigsten Aspekte erläutern, die bei der Planung und Spezifizierung von Performanztests zu berücksichtigen sind

4.6 Wartbarkeitstest

TTA-4.6.1 (K2) Die Gründe für die Einbeziehung von Wartbarkeitstests in eine Testvorgehensweise erläutern

4.7 Übertragbarkeitstest

TTA-4.7.1 (K2) Die Gründe für die Einbeziehung von Übertragbarkeitstests in eine Testvorgehensweise erläutern

4.8 Kompatibilitätstest

TTA-4.8.1 (K2) Die Gründe für die Einbeziehung von Koexistenztests in eine Testvorgehensweise erläutern

4.1 Einführung

Im Allgemeinen fokussieren sich Technical Test Analysten bei ihren Tests darauf, "wie" das Produkt funktioniert (und weniger auf die funktionalen Aspekte, die besagen, "was" das Produkt kann). Diese Tests können auf jeder Teststufe stattfinden. Beispiel: Beim Komponententest von Echtzeitsystemen und eingebetteten Systemen sind das Performanz-Benchmarking und das Testen der Ressourcennutzung besonders wichtig. Beim Systemtest und den betrieblichen Abnahmetests ist das Testen von Zuverlässigkeitsmerkmalen (z.B. Wiederherstellbarkeit) angebracht. Die Tests dieser Teststufe sind auf ein spezifisches System ausgerichtet, d.h. auf eine spezifische Kombination von Hardware und Software. Zu diesem System unter Test können verschiedene Server, Clients, Datenbanken, Netzwerke und andere Ressourcen gehören. Unabhängig von der Teststufe sollte das Testen entsprechend der Risikoprioritäten und den verfügbaren Ressourcen durchgeführt werden.

Sowohl dynamische als auch statische Tests, einschließlich Reviews (siehe Kapitel 2, 3 und 5), können angewandt werden, um die in diesem Kapitel beschriebenen nicht-funktionalen Qualitätsmerkmale zu testen.

Die Beschreibung der Produktqualitätsmerkmale im ISO Standard 25010 [ISO25010] dient als Richtschnur für die Merkmale und ihre Untermerkmale. Diese sind in der nachstehenden Tabelle aufgeführt, aus der ebenfalls hervorgeht, welche Merkmale/Untermerkmale im Test Analyst-Lehrplan und welche im Technical Test Analyst-Lehrplan behandelt werden.

© German Testing Board e.V.

Qualitätsmerkmal	Untermerkmale	Test Analyst	Technical Test Analyst
Funktionale Eignung	Funktionale Korrektheit, funktionale Angemessenheit, funktionale Vollständigkeit	X	
Zuverlässigkeit	Reife, Fehlertoleranz, Wiederherstellbarkeit, Verfügbarkeit		X
Gebrauchstauglichkeit (Usability)	Erkennbare Angemessenheit, Erlernbarkeit, Operabilität, Ästhetik der Benutzungsschnittstelle, Benutzerfehlerschutz, Zugänglichkeit	X	
Performanz	Zeitverhalten, Ressourcennutzung, Kapazität		X
Wartbarkeit	Analysierbarkeit, Modifizierbarkeit, Testbarkeit, Modularität, Wiederverwendbarkeit		X
Übertragbarkeit	Anpassbarkeit, Installierbarkeit, Austauschbarkeit	X	X
IT-Sicherheit	Vertraulichkeit, Integrität, Nichtabstreitbarkeit, Zurechenbarkeit, Authentizität		X
Kompatibilität	Koexistenz		X
	Interoperabilität	X	

Anhang A enthält eine Tabelle, in der die im inzwischen aufgehobenen Standard ISO 9126-1 (wie in Version 2012 dieses Lehrplans verwendet) beschriebenen Merkmale mit denjenigen des neueren Standards ISO 25010 verglichen werden.

Für alle in diesem Abschnitt behandelten Qualitätsmerkmale und -Untermerkmale müssen die typischen Risiken erkannt werden, damit eine geeignete Testvorgehensweise erarbeitet und dokumentiert werden kann. Beim Testen von Qualitätsmerkmalen müssen der Zeitplan im Softwareentwicklungslebenszyklus, benötigte Werkzeuge, die geforderten Standards, die Verfügbarkeit von Software und Dokumentation sowie technisches Fachwissen besondere Beachtung finden. Ohne eine Strategie zur Behandlung jedes einzelnen Merkmals und dessen spezifischem

© German Testing Board e.V.

Testbedarfs wird der Tester möglicherweise nicht genügend Zeit für Planung, Vorbereitung und Durchführung der entsprechenden Tests im Zeitplan bekommen.

Einige dieser Tests, z.B. die Performanztests, können eine umfangreiche Planung, spezielle Ausrüstung, bestimmte Werkzeuge, spezielle Testfähigkeiten und in den meisten Fällen auch einen erheblichen Zeitaufwand erfordern. Das Testen der Qualitätsmerkmale und -Untermerkmale muss in die Gesamttestplanung einfließen, und es müssen für die Aufgaben angemessene Ressourcen ausgewiesen werden.

Während sich Testmanager mit dem Zusammenstellen und Berichten der zusammengefassten Informationen aus Metriken über die Qualitätsmerkmale und -Untermerkmale befassen, sind die Test Analysten oder die Technical Test Analysten für das Erheben der Informationen zu den jeweiligen Metriken zuständig (Zuständigkeiten siehe Tabelle oben).

Metriken von Qualitätsmerkmalen, die der Technical Test Analyst in Tests vor dem Produktivbetrieb erhebt, können als Grundlage für Service-Level-Vereinbarungen zwischen Lieferanten und Stakeholdern (z.B. Kunden, Betreiber) dienen. In manchen Fällen können die Tests auch noch durchgeführt werden, nachdem die Software produktiv gegangen ist. Dann ist oft ein separates Team oder ein separater Unternehmensbereich dafür zuständig. Dies ist häufig bei Performanz- und Zuverlässigkeitstests der Fall, die in der Produktivumgebung zu anderen Testergebnissen führen können als in einer Testumgebung.

4.2 Allgemeine Planungsaspekte

Wenn versäumt wird, nicht-funktionale Tests einzuplanen, kann das den Erfolg eines Projekts ernsthaft gefährden. Der Technical Test Analyst kann vom Testmanager damit beauftragt werden, die wichtigsten Risiken für die relevanten Qualitätsmerkmale (siehe Tabelle in Abschnitt 4.1) zu identifizieren und sich um die Planungsaufgaben in Zusammenhang mit den vorgesehenen Tests zu kümmern. Die Ergebnisse gehen dann möglicherweise auch in das Mastertestkonzept ein

Bei der Durchführung dieser Aufgaben sind die folgenden allgemeinen Faktoren zu berücksichtigen:

- Anforderungen der Stakeholder
- Anforderungen an die Testumgebung
- Beschaffung benötigter Werkzeuge und Schulungen
- Organisatorische Faktoren
- Fragen der Datensicherheit

© German Testing Board e.V.

4.2.1 Anforderungen der Stakeholder

Nicht-funktionale Anforderungen sind oft schlecht spezifiziert oder gar nicht vorhanden. Daher müssen Technical Test Analysten in der Planungsphase die Erwartungshaltungen der betroffenen Stakeholder bezüglich der technischen Qualitätsmerkmale sondieren und bewerten, welche Risiken mit diesen verbunden sind.

Ein üblicher Ansatz ist, davon auszugehen, dass Kunden, die mit der bestehenden Systemversion zufrieden sind, auch mit den neuen Versionen zufrieden sein werden, solange das bisher erreichte Qualitätsniveau erhalten bleibt. In diesem Fall kann die bestehende Version des Systems als Referenzsystem dienen. Für einige der nicht-funktionalen Qualitätsmerkmale kann dies ein sehr nützlicher Ansatz sein, beispielsweise für die Performanz des Systems, wenn es Stakeholdern schwerfällt, die Anforderungen zu spezifizieren.

Es ist ratsam, beim Ermitteln der nicht-funktionalen Anforderungen verschiedene Perspektiven zu berücksichtigen. Hierzu müssen unterschiedliche Stakeholder, wie z.B. Kunden, Product Owner, Anwender, Betriebs- und Wartungspersonal, befragt werden. Wenn wichtige Stakeholder ausgelassen werden, dann werden wahrscheinlich einige Anforderungen nicht berücksichtigt. Weitere Informationen zur Erfassung von Anforderungen sind im Lehrplan für Advanced Level Testmanager [CTAL_TM_SYL] enthalten.

In der agilen Softwareentwicklung können die nicht-funktionalen Anforderungen als User-Stories vorliegen, oder sie können in den Anwendungsfällen als nicht-funktionale Einschränkungen zur spezifizierten Funktionalität hinzugefügt sein.

4.2.2 Anforderungen an die Testumgebung

Viele technische Tests (z.B. IT-Sicherheitstests, Zuverlässigkeitstests, Performanztests) erfordern eine produktivähnliche Testumgebung, um realistische Messungen zu ermöglichen. Größe und Komplexität des Systems unter Test können die Planung und Finanzierung der Tests maßgeblich beeinflussen. Da die Kosten für derartige Umgebungen hoch sein können, sollten die folgenden Alternativen in Betracht gezogen werden:

- Verwendung der tatsächlichen Produktivumgebung
- Verwendung einer reduzierten Version des Systems. Hierbei muss aber sorgfältig vorgegangen werden, damit die erhaltenen Testergebnisse für das tatsächliche Produktivsystem ausreichend repräsentativ sind
- Verwendung von Cloud-basierten Ressourcen als Alternative zur direkten Beschaffung der Ressourcen
- Verwendung virtualisierter Umgebungen

© German Testing Board e.V.

Die Zeiten der Testdurchführung müssen sorgfältig geplant werden, da manche Tests wahrscheinlich nur zu bestimmten Zeiten (z.B. Zeiten geringer Nutzung) stattfinden können.

4.2.3 Beschaffung benötigter Werkzeuge und Schulungen

Werkzeuge sind Teil der Testumgebung. Kommerzielle Werkzeuge oder Simulatoren sind besonders relevant für den Performanztest und für bestimmte IT-Sicherheitstests. Technical Test Analysten sollten die Kosten und Vorlaufzeiten für Beschaffung, Training und Einführung der benötigten Werkzeuge abschätzen. Wenn Spezialwerkzeuge zum Einsatz kommen, sind die damit verbundenen Lernkurven und/oder die Kosten für den Einsatz externer Spezialisten ebenfalls in der Planung zu berücksichtigen.

Die Eigenentwicklung von komplexen Werkzeugen oder Simulatoren kann durchaus zu einem eigenständigen Entwicklungsprojekt werden und sollte auch dementsprechend geplant werden. Insbesondere müssen das Testen und die Dokumentation des eigenentwickelten Werkzeugs in die Zeit- und Ressourcenplanung einfließen. Auch für Aktualisierungen und Nachtests der Simulatoren, die notwendig werden, wenn sich das simulierte System ändert, ist ausreichend Budget und Zeit vorzusehen. Wenn Simulatoren für IT-sicherheitskritische Anwendungen eingesetzt werden sollen, sind auch die entsprechenden Abnahmetests und eine etwaige Zertifizierung des Simulators durch eine unabhängige Instanz zu berücksichtigen.

4.2.4 Organisatorische Faktoren

Bei nicht-funktionalen Tests ist oft auch das Verhalten mehrerer Komponenten eines Gesamtsystems zu messen (z.B. Server, Datenbanken, Netzwerke). Wenn diese Komponenten auf unterschiedliche Standorte und Unternehmen verteilt sind, kann dies erheblichen Aufwand für die Planung und Koordination der Tests bedeuten. So könnten bestimmte Softwarekomponenten nur zu einer bestimmten Tageszeit für Systemtests zur Verfügung stehen. Unternehmen, die das Testen unterstützen, stehen möglicherweise nur für eine begrenzte Anzahl von Tagen zur Verfügung. Es kann zu empfindlichen Störungen der geplanten Tests führen, wenn nicht im Vorfeld geklärt worden ist, dass die Systemkomponenten und das Personal anderer Unternehmen (d.h. „ausgeliehene“ externe Expertise) für die Testzwecke auf Abruf bereitstehen.

4.2.5 Datensicherheit und Datenschutz

Spezifische Sicherheitsmaßnahmen für ein System sollten bereits in der Testplanungsphase berücksichtigt werden, damit alle vorgesehenen Testaktivitäten tatsächlich durchführbar sind. Zum Beispiel kann die Nutzung von verschlüsselten Daten die Erstellung von Testdaten und die Verifizierung der Ergebnisse erschweren.

© German Testing Board e.V.

Richtlinien und gesetzliche Bestimmungen zum Datenschutz können ausschließen, dass die erforderlichen Testdaten auf der Grundlage von Produktivdaten generiert werden (z.B. persönliche Daten, Kreditkartendaten). Die Anonymisierung von Testdaten ist eine schwierige Aufgabe und muss als Teil der Testrealisierung eingeplant werden.

4.3 IT-Sicherheitstest

4.3.1 Gründe für die Berücksichtigung von IT-Sicherheitstests

IT-Sicherheitstests untersuchen die Verwundbarkeit eines Systems durch diverse Gefährdungen, indem versucht wird, die IT-Sicherheitsrichtlinie eines Systems gezielt außer Kraft zu setzen. Die nachfolgende Liste enthält mögliche Gefährdungen, die beim IT-Sicherheitstest untersucht werden sollten:

- Nicht autorisiertes Kopieren von Anwendungen oder Daten
- Nicht autorisierter Zugriff (z.B. Aktionen ausführen, für die der Benutzer keine Berechtigung besitzt). Diese Tests sind auf Benutzerrechte, Zugriffsberechtigungen und Privilegien fokussiert. Diese Informationen sollten in den Spezifikationen des Systems enthalten sein.
- Software, die nicht beabsichtigte Seiteneffekte aufweist, wenn sie ihre beabsichtigte Funktion ausführt. Beispiel: Ein Media Player spielt Audio korrekt ab, schreibt dabei aber Dateien in einen nicht verschlüsselten Zwischenspeicher. Softwarepiraten könnten eine solche Seiteneffekte ausnutzen.
- Bösartiger Code wird von außen in eine Webseite eingeschleust und durch nachfolgende Benutzer ausgeführt (webseitenübergreifendes Skripten, engl. cross-site scripting, XSS).
- Pufferüberlauf (Speicherüberlauf), der durch die Eingabe von extrem langen Zeichenketten in ein Eingabefeld der Benutzungsschnittstelle ausgelöst werden kann, die vom Code nicht korrekt verarbeitet werden können. Über Pufferüberlaufgefährdung besteht die Möglichkeit, bösartigen Code auszuführen.
- Dienstblockade (engl. denial of service, DoS), die bewirkt, dass Benutzer nicht mit einer Anwendung interagieren können (z.B. durch Überlastung eines Webservers mit Störanfragen)
- Das Abfangen, Nachahmen und/oder Verändern und die anschließende Weiterleitung von Kommunikationen (z.B. Daten von Kreditkartentransaktionen) durch einen Dritten, ohne dass der Benutzer das Vorhandensein der dritten Partei bemerkt (Man-in-the-middle-Angriff)
- Knacken der Verschlüsselungscodes, die zum Schutz sensibler Daten verwendet werden

© German Testing Board e.V.

- Logische Fallen (engl. logic bombs, auch “easter eggs” genannt), die in bösartiger Absicht in den Code eingeschleust werden und die nur unter bestimmten Bedingungen aktiviert werden (z.B. an einem bestimmten Datum). Wenn diese logischen Fallen aktiviert werden, lösen sie Schadaktionen aus, wie das Löschen von Dateien oder das Formatieren von Festplatten.

4.3.2 IT-Sicherheitstest planen

Für die Planung von IT-Sicherheitstests sind die folgenden Themen besonders relevant:

- Da IT-Sicherheitsprobleme schon beim Architekturentwurf, Systementwurf und der Systemimplementierung verursacht werden können, sind IT-Sicherheitstests bei den Komponententests, Integrationstests und Systemtests einzuplanen. Da sich IT-Sicherheitsbedrohungen ständig ändern können, sollten regelmäßige IT-Sicherheitstests auch für die Zeit nach dem Produktivgang des Systems eingeplant werden. Dies gilt insbesondere für dynamische offene Architekturen wie das Internet of Things, bei denen die Produktivphase durch viele Aktualisierungen der verwendeten Software- und Hardwareelemente gekennzeichnet ist.
- Die vom Technical Test Analysten vorgeschlagenen Testvorgehensweisen sollten Code-Reviews, Reviews der Architektur und des Entwurfs, sowie die statische Analyse des Codes mit Hilfe von IT-Sicherheitstestwerkzeugen beinhalten. Diese Maßnahmen können sehr effektiv sein, um IT-Sicherheitsprobleme aufzudecken, die beim dynamischen Test leicht übersehen werden.
- Technical Test Analysten werden häufig auch gebeten, bestimmte IT-Sicherheitsangriffe zu entwerfen und auszuführen (siehe unten). Dies erfordert eine sorgfältige Planung und Koordination mit Stakeholdern (einschließlich der Spezialisten für IT-Sicherheitstests). Andere IT-Sicherheitstests können in Zusammenarbeit mit Entwicklern oder Test Analysten durchgeführt werden (z.B. Tests der Benutzerrechte, Zugriffsberechtigungen und Privilegien).
- Ein wesentlicher Aspekt bei der Planung von IT-Sicherheitstests ist das Einholen von Genehmigungen. Für den Technical Test Analysten bedeutet dies konkret, dass vom Testmanager die ausdrückliche Erlaubnis zur Durchführung der geplanten IT-Sicherheitstests eingeholt werden muss. Alle zusätzlichen, spontan angesetzten Tests könnten für echte Angriffe gehalten werden, was für die ausführende Person rechtliche Konsequenzen haben könnte. Wenn es keine schriftlichen Unterlagen gibt, aus denen die Beauftragung und Genehmigung der Tests hervorgehen, wird die Ausrede "Wir haben einen IT-Sicherheitstest durchgeführt" wenig überzeugend klingen.

© German Testing Board e.V.

- Die gesamte Planung von IT-Sicherheitstests sollte mit dem IT-Sicherheitsbeauftragten des Unternehmens koordiniert werden, falls diese Rolle im Unternehmen vorhanden ist.
- Es ist zu beachten, dass Verbesserungen der IT-Sicherheit eines Systems sich auf dessen Performanz oder Zuverlässigkeit auswirken können. Nach der Durchführung von Verbesserungen der IT-Sicherheit ist es ratsam, die Durchführung von Performanz- oder Zuverlässigkeitstests in Erwägung zu ziehen (siehe Abschnitte 4.5 und 4.4).

Für die Planung von IT-Sicherheitstests können ggf. spezifische Standards zur Anwendung kommen, wie z.B. [IEC 62443-3-2], der für industrielle Automatisierungs- und Steuerungssysteme gilt.

Der Lehrplan Sicherheitstester [CT_SEC_SYL] enthält weitere Informationen über die wichtigsten Elemente des IT-Sicherheitstestkonzepts.

4.3.3 Spezifikation von IT-Sicherheitstests

Bestimmte IT-Sicherheitstests lassen sich je nach Ursprung des IT-Sicherheitsrisikos unterscheiden [Whittaker04]:

- Die Benutzungsschnittstelle betreffend – Nicht autorisierter Zugriff und böswillige Eingaben
- Das Dateisystem betreffend – Zugriff auf vertrauliche Daten in Dateien oder Repositorien
- Das Betriebssystem betreffend – Speicherung sensibler Informationen wie Passwörter in unverschlüsselter Form. Wird das System durch böswillige Eingaben zum Absturz gebracht, können die Informationen zugänglich werden.
- Externe Software betreffend – Interaktionen zwischen externen Komponenten, die das System nutzt. Diese können auf Netzwerkebene auftreten (wenn beispielsweise inkorrekte Datenpakete oder Meldungen übertragen werden) oder auf der Ebene der Softwarekomponenten (wenn beispielsweise eine Softwarekomponente ausfällt, die vom System benötigt wird).

Die Untermerkmale der IT-Sicherheit laut ISO 25010[ISO25010] liefern auch eine Grundlage, auf der IT-Sicherheitstests spezifiziert werden können. Diese konzentrieren sich auf die folgenden Aspekte der IT-Sicherheit:

- Vertraulichkeit
- Integrität
- Nichtabstreitbarkeit
- Zurechenbarkeit

© German Testing Board e.V.

- Authentizität

Die folgende Vorgehensweise [Whittaker04] kann zur Entwicklung von IT-Sicherheitstests verwendet werden:

- Nützliche Informationen zur Spezifikation von Tests zusammentragen, wie z.B. Namen von Mitarbeitern, physikalische Adressen, Details der internen Netzwerke, IP-Nummern, Typ/Version der verwendeten Software / Hardware und die Betriebssystemversion.
- Das System mit gängigen Werkzeugen auf Verwundbarkeiten scannen. Diese Werkzeuge dienen nicht direkt dazu, das System/die Systeme zu kompromittieren, sondern um Verwundbarkeiten zu identifizieren, die eine Verletzung der IT-Sicherheitsrichtlinie darstellen oder zu einer solchen führen können. Spezifische Verwundbarkeiten lassen sich auch mit Hilfe von Informationen und Checklisten identifizieren, z.B. mit Checklisten des National Institute of Standards and Technology (NIST, US-Bundesbehörde) [Web-1] und des Open Web Application Security Project™ (OWASP) [Web-4].
- IT-Sicherheitsangriffe/Angriffspläne entwickeln, d.h. mit den gesammelten Informationen werden Testschritte geplant, die die IT-Sicherheitsrichtlinie eines bestimmten Systems durchbrechen sollen. Für die geplanten Angriffe müssen mehrere Eingaben über verschiedene Schnittstellen (z.B. Benutzungsschnittstelle, Dateisystem) spezifiziert werden, um die schwerwiegendsten IT-Sicherheitsfehler aufzudecken. Die verschiedenen in [Whittaker04] beschriebenen "Angriffe" sind eine wertvolle Quelle für Verfahren, die speziell für IT-Sicherheitstests entwickelt wurden.

Anmerkung: Angriffspläne können für Penetrationstests entwickelt werden.

Abschnitt 3.2 (statische Analyse) und der Lehrplan Sicherheitstester [CT_SEC_SYL] enthalten weitere Einzelheiten über IT-Sicherheitstests.

4.4 Zuverlässigkeitstest

4.4.1 Einführung

Der Standard ISO 25010 klassifiziert die folgenden Untermerkmale für das Produktqualitätsmerkmal Zuverlässigkeit: Reife, Verfügbarkeit, Fehlertoleranz und Wiederherstellbarkeit. Beim Testen der Zuverlässigkeit geht es um die Fähigkeit eines Systems oder einer Software, spezifizierte Funktionen unter bestimmten Bedingungen über eine spezifizierte Zeitspanne auszuführen.

© German Testing Board e.V.

4.4.2 Softwarereife testen

Reife ist der Grad, zu dem ein System (oder eine Software) die Anforderungen an die Zuverlässigkeit im Normalbetrieb erfüllt, die normalerweise mithilfe eines Nutzungsprofils festgelegt werden (siehe Abschnitt 4.9). Wenn Reife gemessen wird, dient ihr Messwert häufig als eines der Freigabekriterien für ein System.

Traditionell wird die Reife für hochzuverlässige Systeme spezifiziert und gemessen, wie z.B. für Systeme mit sicherheitskritischen Funktionen (z.B. ein Flugsteuerungssystem), bei denen das Ziel der Reife in einem regulatorischen Standard definiert ist. Eine Reifeanforderung für ein solches hochzuverlässiges System kann eine mittlere Betriebsdauer zwischen Ausfällen (Mean Time Between Failures, MTBF) von bis zu 10^9 Stunden sein (obwohl dies praktisch unmöglich zu messen ist).

Der übliche Ansatz, die Reife von hochzuverlässigen Systemen zu testen, ist die Modellierung des Zuverlässigkeitswachstums, die normalerweise am Ende des Systemtests erfolgt, nachdem die Tests für andere Qualitätsmerkmale abgeschlossen und alle Fehlerzustände in Verbindung mit aufgedeckten Fehlerwirkungen behoben wurden. Es handelt sich um einen statistischen Ansatz, der normalerweise in einer Testumgebung durchgeführt wird, die der Produktivumgebung so nahe wie möglich kommt. Um eine bestimmte MTBF zu messen, werden auf der Grundlage des Nutzungsprofils Testeingaben generiert, die Software wird ausgeführt und Fehlerwirkungen/Ausfälle werden aufgezeichnet (und anschließend behoben). Eine Abnahme der Häufigkeit von Fehlerwirkungen/Ausfällen ermöglicht die Vorhersage der MTBF mithilfe eines Zuverlässigkeitswachstumsmodells.

Wenn Reife als Ziel für Systeme mit geringerer Zuverlässigkeit (z.B. nicht sicherheitsbezogene Systeme) dient, kann die Anzahl der Fehlerwirkungen/Ausfälle, die während einer bestimmten Zeitspanne der erwarteten Nutzung beobachtet werden (z.B. nicht mehr als zwei Ausfälle mit hohem Schweregrad pro Woche), herangezogen werden und als Teil der Service-Level-Vereinbarung (SLA) für das System festgehalten werden.

4.4.3 Verfügbarkeit testen

Die Verfügbarkeit wird normalerweise als die Zeitspanne angegeben, in der ein System (oder eine Software) unter normalen Betriebsbedingungen für Benutzer und andere Systeme verfügbar ist. Systeme können eine niedrige Reife haben, aber dennoch eine hohe Verfügbarkeit aufweisen. Beispiel: Ein Telefonnetz kann mehrere Anrufe nicht verbinden (und damit eine niedrige Reife aufweisen), aber solange sich das System schnell erholt und die nächsten Verbindungsversuche erfolgreich sind, werden die meisten Benutzer zufrieden sein. Eine einzige Fehlerwirkung, die zu einem mehrstündigen Ausfall des Telefonnetzes führt, würde jedoch einen inakzeptablen Grad der Verfügbarkeit darstellen. Die Verfügbarkeit wird häufig als Teil einer Service-

© German Testing Board e.V.

Level-Vereinbarung spezifiziert und für operative Systeme wie Webseiten und Software-as-a-Service-Anwendungen (SaaS) gemessen. Die Verfügbarkeit eines Systems kann mit 99,999 % ("fünf Neunen") beziffert werden; in diesem Fall darf das System nicht länger als 5 Minuten pro Jahr nicht verfügbar sein. Alternativ kann die Systemverfügbarkeit als die Nichtverfügbarkeit angegeben werden (z.B. darf das System nicht länger als 60 Minuten pro Monat ausfallen).

Bei der Messung der Verfügbarkeit vor dem produktiven Betrieb (z.B. als Teil der Entscheidung über die Produktivfreigabe) werden häufig die gleichen Tests wie bei der Messung der Reife durchgeführt; diese Tests basieren auf einem Nutzungsprofil der erwarteten Nutzung über einen längeren Zeitraum und erfolgen in einer Testumgebung, die der Produktivumgebung so nahe wie möglich kommt. Die Verfügbarkeit kann mit der Formel MTTF/(MTTF + MTTR) berechnet werden, wobei MTTF die mittlere Betriebsdauer bis zum Ausfall (Mean Time To Failure) und MTTR die mittlere Reparaturzeit nach einem Ausfall (Mean Time To Repair) ist. Diese Metriken werden im Rahmen von Wartbarkeitstests oft erhoben. Handelt es sich um ein System mit hoher Zuverlässigkeit und Wiederherstellbarkeit (siehe Abschnitt 4.4.5), kann die MTTR in der Gleichung durch die mittlere Wiederherstellzeit ersetzt werden, wenn das System einige Zeit braucht, um sich von einem Ausfall zu erholen.

4.4.4 Fehlertoleranz testen

Systeme (oder Software) mit extrem hohen Zuverlässigkeitsanforderungen sind oft fehlertolerant ausgelegt, so dass das System im Idealfall ohne merkliche Ausfallzeiten weiterarbeiten kann, wenn es zu Ausfällen kommt. Das wichtigste Maß für die Fehlertoleranz eines Systems ist die Fähigkeit des Systems, Fehlerwirkungen und Ausfälle zu tolerieren. Beim Testen der Fehlertoleranz werden daher Fehlerwirkungen/Ausfälle simuliert, um festzustellen, ob das System bei einem solchen Ausfall weiterarbeiten kann. Die Identifizierung potenzieller Fehlerbedingungen, die getestet werden sollen, ist ein wichtiger Teil des Testens auf Fehlertoleranz.

Ein fehlertoleranter Systementwurf sieht meist ein oder mehrere identische Teilsysteme vor, so dass im Falle eines Ausfalls ein gewisses Maß an Redundanz gegeben ist. Im Falle von Software müssen solche duplizierten Systeme unabhängig voneinander entwickelt werden, um gleichartige Fehlerwirkungen (engl. common mode failures) zu vermeiden. Dieser Ansatz ist als N-Versionsprogrammierung bekannt. Flugsteuerungssysteme können drei oder vier Redundanzstufen haben, wobei die kritischsten Funktionen in mehreren Varianten implementiert sind. Wenn die Zuverlässigkeit der Hardware eine Rolle spielt, kann ein eingebettetes System auf mehreren unterschiedlichen Prozessoren laufen.Eine kritische Webseite kann mit einem Spiegelserver betrieben werden kann, der dieselben Funktionen ausführt ("spiegelt") und immer verfügbar ist, um bei einem Ausfall des Hauptservers zu übernehmen (Failover). Unabhängig davon, welcher Ansatz für die Fehlertoleranz

© German Testing Board e.V.

implementiert wird, müssen beim Testen sowohl die Aufdeckung der Fehlerwirkung als auch die anschließende Reaktion auf die Fehlerwirkung getestet werden.

Das Testen mit Fehlereinfügung bewertet die Robustheit eines Systems gegenüber Fehlerzuständen in der Umgebung des Systems (z.B. eine fehlerhafte Stromversorgung, schlecht formulierte Eingabemeldungen, nicht verfügbarer Prozess oder Dienst, nicht gefundene Datei oder nicht verfügbarer Speicher) und Fehlerzustände im System selbst (z.B. ein durch kosmische Strahlung verursachtes umgedrehtes Bit, schlechtes Design oder Programmierung). Fehlereinfügungstests sind eine Form von Negativtests; es werden absichtlich Fehlerzustände in das System eingefügt, um sich davon zu überzeugen, dass das System wie erwartet reagiert (d.h. funktionssicher für ein sicherheitsbezogenes System). Manchmal werden Fehlerszenarien getestet, die niemals auftreten sollten (z.B. sollte eine Softwaretransaktion niemals abstürzen oder in einer Endlosschleife enden) und deshalb durch herkömmliche Systemtests nicht simuliert werden können. Mit Hilfe der Fehlereinfügung wird das Fehlerszenario erstellt und das anschließende Systemverhalten gemessen, um sicherzustellen, dass es die Fehlerwirkung erkennt und behandelt.

4.4.5 Wiederherstellbarkeitstest

Die Wiederherstellbarkeit ist ein Maß für die Fähigkeit eines Systems (oder einer Software), sich von einem Ausfall zu erholen, entweder in Bezug auf die für die Wiederherstellung (ggf. zu einem eingeschränktem Betriebsmodus) benötigte Zeit oder auf die Menge der verlorenen Daten. Zu den Ansätzen für Wiederherstellbarkeitstests gehören Ausfallsicherungstests sowie Sicherungs- und Wiederherstellungstests. Beide beinhalten normalerweise das Testen der Wiederherstellungsverfahren im Probelauf und nur gelegentlich praktische, im Idealfall unangekündigte Tests in Produktivumgebungen.

Sicherungs- und Wiederherstellungstests fokussieren auf die eingesetzten Verfahren zur Minimierung der Auswirkungen von Fehlerwirkungen auf die Systemdaten. Solche Tests bewerten die Verfahren sowohl zur Sicherung als auch zur Wiederherstellung der Daten. Während das Testen der Datensicherung relativ einfach ist, kann das Testen der Wiederherstellung eines Systems aus gesicherten Daten komplexer sein und erfordert oft eine sorgfältige Planung, um sicherzustellen, dass das Produktivsystem so wenig wie möglich beeinträchtigt wird. Mögliche Metriken, die bei Sicherungs- und Wiederherstellungstests gemessen werden können, sind: Zeitaufwand für die verschiedenen Sicherungsarten (z.B. vollständig, inkrementell), Zeitaufwand für die Wiederherstellung der Daten (das Ziel der Wiederherstellungszeit), sowie das Ausmaß des Datenverlustes, das akzeptabel ist (das Ziel des Wiederherstellungspunkts).

© German Testing Board e.V.

Die Ausfallsicherung wird getestet, wenn die Systemarchitektur neben dem Primärsystem auch ein Ausfallsicherungssystem umfasst, das bei einem Ausfall des Primärsystems dessen Aufgaben übernimmt. Wenn ein System in der Lage sein muss, sich von einem Systemausfall nach einem katastrophalen Ereignis zu erholen (z.B. einer Überschwemmung, einem Terroranschlag oder einem schwerwiegenden Ransomware-Angriff), werden Ausfallsicherungstests oft als Disaster-Recovery-Test (in Deutsch auch Notfallwiederherstellungstest genannt) bezeichnet. Dabei befinden sich meist ein oder mehrere Ausfallsicherungssysteme an anderen geografischen Standorten. Die Durchführung eines vollständigen Disaster-Recovery-Tests an einem Produktivsystem erfordert aufgrund der Risiken und der damit verbundenen Beeinträchtigungen eine äußerst sorgfältige Planung (oft wird die Freizeit von Führungskräften für die Wiederherstellung in Anspruch genommen). Wenn ein vollständiger Disaster-Recovery-Test fehlschlägt, erfolgt eine sofortige Rückkehr zum Primärsystem (da dieses nicht wirklich beschädigt wurde!). Ausfallsicherungstests prüfen auch, dass die erforderlichen Service Levels eingehalten werden, nachdem das Ausfallsicherungssystem übernommen hat.

4.4.6 Zuverlässigkeitstests planen

Im Allgemeinen sind für die Planung von Zuverlässigkeitstests folgende Aspekte besonders relevant:

- Zeitpunkt – Zuverlässigkeitstests erfordern normalerweise, dass das Gesamtsystem getestet wird und andere Testarten bereits abgeschlossen sind – die Durchführung kann viel Zeit in Anspruch nehmen.
- Kosten – Das Testen von hochzuverlässigen Systemen ist sehr kostspielig, da die Systeme über einen langen Zeitraum ohne Fehlerwirkung/Ausfall getestet werden müssen, um eine hohe MTBF vorhersagen zu können.
- Zeitdauer – Das Testen der Softwarereife mithilfe von Zuverlässigkeitswachstumsmodellen basiert auf den erkannten Fehlerwirkungen/Ausfällen und wird bei einem hohen Zuverlässigkeitsgrad viel Zeit in Anspruch nehmen, um statistisch aussagekräftige Ergebnisse zu erhalten.
- Testumgebung – Die Testumgebung muss der Produktivumgebung so ähnlich wie möglich sein, oder es kann die Produktivumgebung verwendet werden. Bei Verwendung der Produktivumgebung kann dies jedoch für die Benutzer störend sein und ein hohes Risiko darstellen, wenn z.B. ein Disaster-Recovery-Test das Produktivsystem beeinträchtigt.
- Umfang – Einzelne Teilsysteme und Komponenten können auf unterschiedliche Arten und Grade der Zuverlässigkeit getestet werden.
- Endekriterien – Die Zuverlässigkeitsanforderungen sollten durch regulatorische Standards für sicherheitsbezogene Anwendungen festgelegt werden.

© German Testing Board e.V.

- Fehlerwirkung/Ausfall – Zuverlässigkeitsmessungen sind in hohem Maße von der erfassten Anzahl von Fehlerwirkungen/Ausfällen abhängig, weshalb im Vorfeld vereinbart werden muss, was als Fehlerwirkung/Ausfall gilt.
- Entwickler – Für das Testen der Softwarereife unter Verwendung von Zuverlässigkeitswachstumsmodellen muss mit den Entwicklern vereinbart werden, dass die identifizierten Fehlerzustände so schnell wie möglich behoben werden.
- Die Messung der Zuverlässigkeit im Betrieb ist im Vergleich zur Messung der Zuverlässigkeit vor der Freigabe relativ einfach, da hierbei nur Fehlerwirkungen/Ausfälle erfasst werden müssen; dies kann eine Zusammenarbeit mit dem Betriebspersonal erfordern.
- Frühzeitiges Testen – Um eine hohe Zuverlässigkeit zu erzielen (im Unterschied zur Messung der Zuverlässigkeit), muss mit dem Testen so früh wie möglich begonnen werden, mit strengen Reviews der Dokumenten von frühzeitigen Baselines und der statischen Analyse von Programmcode.

4.4.7 Spezifikation von Zuverlässigkeitstests

Das Testen der Reife und der Verfügbarkeit basiert weitgehend auf dem Testen des Systems unter normalen Betriebsbedingungen. Für solche Tests ist ein Nutzungsprofil erforderlich, das die erwartete Nutzung des Systems definiert. Siehe Abschnitt 4.9 für weitere Einzelheiten zu Nutzungsprofilen.

Zum Testen von Fehlertoleranz und Wiederherstellbarkeit ist es oft notwendig, Tests zu erstellen, die Fehlerwirkungen in der Umgebung und im System selbst simulieren, um festzustellen, wie das System reagiert. Hierfür werden häufig Fehlereinfügungstests verwendet. Für die Identifizierung möglicher Fehlerzustände und entsprechender Fehlerwirkungen stehen verschiedene Verfahren und Checklisten zur Verfügung (z.B. Fehlerbaum-Analyse (engl. failure-tree analysis, FTA), Fehlermöglichkeits- und Einflussanalyse (engl. failure mode and effect analysis, FMEA)).

4.5 Performanztest

4.5.1 Einführung

Die Klassifikation der Produktqualitätsmerkmale im Standard ISO 25010 definiert die folgenden Untermerkmale der Performanz: Zeitverhalten, Ressourcennutzung und Kapazität. Beim Testen der Performanz geht es um die Messung der Performanz eines Systems oder einer Software unter bestimmten Bedingungen im Verhältnis zur Menge der genutzten Ressourcen. Zu den typischen Ressourcen gehören Zeit, Prozessorleistung, Speicherkapazität und Bandbreite.

© German Testing Board e.V.

4.5.2 Testen des Zeitverhaltens

Beim Testen des Zeitverhaltens werden die folgenden Aspekte eines Systems (oder einer Software) unter bestimmten Betriebsbedingungen gemessen:

- Zeit, die vom Eingang einer Anfrage bis zur ersten Antwort vergeht (d.h. die Zeit bis zum Beginn der Antwort, nicht die Zeit bis zum Abschluss der angeforderten Aktivität), auch Antwortzeit genannt;
- Durchlaufzeit vom Beginn einer Aktivität bis zum Abschluss der Aktivität, auch Bearbeitungszeit genannt;
- Anzahl der abgeschlossenen Aktivitäten pro Zeiteinheit (z.B. Anzahl der Datenbankoperationen pro Sekunde), auch Durchsatzrate genannt.

Für viele Systeme sind maximale Antwortzeiten für verschiedene Systemfunktionen als Anforderungen spezifiziert. In solchen Fällen entspricht die Antwortzeit der Zeitspanne vom Eingang einer Anfrage bis zur ersten Antwort plus der Durchlaufzeit. Wenn ein System bis zum Abschluss einer Aktivität eine Reihe von Schritten durchführen muss (z.B. eine Pipeline), kann es sinnvoll sein, die für jeden Schritt benötigte Zeit zu messen und die Ergebnisse zu analysieren, um festzustellen, ob ein oder mehrere Schritte einen Engpass verursachen.

4.5.3 Testen der Ressourcennutzung

Beim Testen der Ressourcennutzung werden die folgenden Aspekte eines Systems (oder einer Software) unter bestimmten Betriebsbedingungen gemessen:

- CPU-Auslastung, normalerweise als Prozentsatz der verfügbaren CPU-Zeit;
- Speicherauslastung, normalerweise als Prozentsatz des verfügbaren Speichers;
- E/A-Gerätenutzung, normalerweise als Prozentsatz der verfügbaren E/A-Gerätezeit;
- Bandbreitennutzung, normalerweise als Prozentsatz der verfügbaren Bandbreite.

4.5.4 Testen der Kapazität

Beim Testen der Kapazität werden die Obergrenzen der folgenden Aspekte eines Systems (oder einer Software) unter bestimmten Betriebsbedingungen gemessen:

- verarbeitete Transaktionen pro Zeiteinheit (z.B. maximal 687 übersetzte Wörter pro Minute);
- Benutzer, die gleichzeitig auf das System zugreifen (z.B. maximal 1223 Benutzer);

© German Testing Board e.V.

- neu hinzugefügte Benutzer, die pro Zeiteinheit Zugang zum System haben (z.B. maximal 400 hinzugefügte Benutzer pro Sekunde).

4.5.5 Allgemeine Aspekte von Performanztests

Beim Testen des Zeitverhaltens, der Ressourcennutzung oder der Kapazität ist es normal, dass mehrere Messungen durchgeführt werden und der Mittelwert als Metrik verwendet wird, da die gemessenen Zeitwerte je nach anderen Hintergrundaufgaben, die das System ausführt, schwanken können. In manchen Situationen werden die Messergebnisse sorgfältiger ausgewertet (z.B. die Varianz oder als ein anderes statistisches Maß), oder es werden Ausreißer untersucht und gegebenenfalls entfernt.

Die dynamische Analyse (siehe Abschnitt 3.3.4) kann verwendet werden, um Komponenten zu identifizieren, die einen Performanzengpass verursachen, um die verwendeten Ressourcen beim Testen der Ressourcennutzung zu messen, und um maximale Grenzen beim Testen der Kapazität zu messen.

4.5.6 Arten von Performanztests

Performanztests unterscheiden sich von den meisten anderen Testarten dadurch, dass sie zwei verschiedene Ziele haben können. Das erste besteht darin, festzustellen, ob die Software unter Test die spezifizierten Abnahmekriterien erfüllt. Beispielsweise soll festgestellt werden, ob das System eine aufgerufene Webseite wie spezifiziert innerhalb von maximal 4 Sekunden anzeigt. Das zweite Ziel besteht darin, den Systementwicklern Informationen zur Verfügung zu stellen, die ihnen helfen, die Effizienz des Systems zu verbessern. So können beispielsweise Performanzengpässe entdeckt und beeinträchtigte Teile der Systemarchitektur identifiziert werden, wenn eine unerwartet hohe Anzahl von Benutzern gleichzeitig auf das System zugreift.

Alle in den Abschnitten 4.5.2, 4.5.3 und 4.5.4 oben beschriebenen Performanztests können dazu verwendet werden, um festzustellen, ob die Software unter Test die spezifizierten Abnahmekriterien erfüllt. Sie dienen auch zur Messung von Referenzwerten, die bei späteren Änderungen des Systems zum Vergleich herangezogen werden. Die nachfolgenden Arten von Performanztests werden eher dazu verwendet, den Entwicklern Informationen darüber zu liefern, wie sich das System unter verschiedenen Betriebsbedingungen verhält.

Lasttests

Lasttests untersuchen die Fähigkeit eines Systems, verschiedene Systemlasten zu bewältigen. Diese Lasten werden normalerweise anhand der Anzahl der Benutzer, die gleichzeitig auf das System zugreifen, oder der Anzahl der gleichzeitig ablaufenden Prozesse definiert und können als Nutzungsprofile definiert sein (siehe Abschnitt 4.9 für weitere Einzelheiten zu Nutzungsprofilen). Der Umgang mit diesen Lasten wird

© German Testing Board e.V.

normalerweise am Zeitverhalten des Systems und an der Ressourcennutzung gemessen (z.B. Bestimmung der Auswirkung einer Verdoppelung der Benutzerzahl auf die Antwortzeit). Bei der Durchführung von Lasttests ist es üblich, mit einer geringen Last zu beginnen und die Last schrittweise zu erhöhen, während das Zeitverhalten und die Ressourcennutzung des Systems gemessen werden. Typische Informationen aus Lasttests, die für Entwickler nützlich sein könnten, sind unerwartete Veränderungen bei den Antwortzeiten oder der Nutzung von Systemressourcen, während das System eine bestimmte Last bewältigt.

Stresstests

Es gibt zwei Arten von Stresstests: Die erste ist dem Lasttest ähnlich, der zweite ist eine Variante des Robustheitstests.

Im ersten Fall wird die Last normalerweise zunächst auf den erwarteten Maximalwert eingestellt und dann so lange erhöht, bis das System ausfällt (z.B. wenn die Antwortzeiten unangemessen lang werden oder das System abstürzt). Anstatt das System zum Ausfall zu zwingen, wird bei manchen Stresstests zunächst eine hohe Last verwendet, um das System zu belasten, die dann auf ein normales Niveau reduziert wird, um zu prüfen, ob sich die Systemleistung wieder auf dem Niveau vor der hohen Belastung stabilisiert.

Im zweiten Fall werden Performanztests durchgeführt, bei denen das System absichtlich beeinträchtigt wird, indem der Zugang zu erwarteten Ressourcen reduziert wird (z.B. durch Verringerung des verfügbaren Speichers oder der verfügbaren Bandbreite). Die Ergebnisse der Stresstests können den Entwicklern Aufschluss darüber geben, welche Aspekte des Systems kritisch bzw. Schwachstellen sind und daher eventuell verbessert werden müssen.

Skalierbarkeitstests

Ein skalierbares System kann sich an unterschiedliche Systemlasten anpassen. Beispielsweise kann eine skalierbare Webseite bei steigender Belastung mehr Backend-Server verwenden und bei sinkender Belastung weniger. Skalierbarkeitstests ähneln den Lasttests, testen aber die Fähigkeit eines Systems, bei wechselnden Belastungen (z.B. mehr Benutzer als die aktuelle Hardware bewältigen kann) nach oben und unten zu skalieren.

Im Lehrplan Performanztest [CT_PT_SYL] werden weitere Arten von Performanztests beschrieben.

4.5.7 Performanztest planen

Im Allgemeinen sind für die Planung von Performanztests folgende Aspekte besonders relevant:

© German Testing Board e.V.

- Zeitpunkt – Für Performanztests muss meist das gesamte System implementiert sein und in einer repräsentativen Testumgebung ausgeführt werden, was bedeutet, dass diese Tests normalerweise als Teil der Systemtests durchgeführt werden.
- Reviews – Code-Reviews können Performanzprobleme identifizieren, insbesondere, wenn sie auf Datenbankinteraktionen, Komponenteninteraktionen und Fehlerbehandlung fokussiert sind, speziell im Zusammenhang mit „Wait/Retry-Logik“ und mit ineffizienten Abfragen. Diese Code-Reviews sollten vorgesehen werden, sobald der Code verfügbar ist (d.h. vor den dynamischen Tests).
- Frühzeitiges Testen – Manche Performanztests (z.B. die Ermittlung der CPU-Auslastung einer kritischen Komponente) können als Teil der Komponententests geplant werden. Komponenten, die bei den Performanztests als ein Performanzengpass identifiziert wurden, können verbessert und im Rahmen der Komponententests einzeln nachgetestet werden.
- Änderungen an der Systemarchitektur – Schlechte Ergebnisse von Performanztests können manchmal zu einer Änderung der Systemarchitektur führen. Wenn absehbar ist, dass die Ergebnisse von Performanztests solch wesentliche Änderungen am System nahelegen, dann sollten die Performanztests so früh wie möglich beginnen, damit möglichst viel Zeit für die Behebung der Probleme zur Verfügung steht.
- Kosten – Da Werkzeuge und Testumgebungen kostenintensiv sein können, ist es oft sinnvoll, Cloud-basierte Testumgebungen mit den nötigen Werkzeuglizenzen temporär anzumieten. In solchen Fällen muss die Zeit für die Durchführung der Tests in der Testplanung optimiert werden, um die Kosten zu minimieren.
- Testumgebung – Die Testumgebung sollte die Produktivumgebung so repräsentativ wie möglich abbilden, da sonst die Skalierung der Performanztestergebnisse von der Testumgebung auf die tatsächliche Produktivumgebung eine größere Herausforderung darstellt.
- Endekriterien – Performanzanforderungen sind bisweilen nur schwer vom Kunden zu erhalten und werden daher oft von Referenzwerten früherer oder ähnlicher Systeme abgeleitet. Bei eingebetteten sicherheitsbezogenen Systemen können einige Anforderungen, wie z.B. die maximale CPU- und Speichernutzung, durch regulatorische Standards festgelegt sein.
- Werkzeuge – Zur Unterstützung von Performanztests werden meist Last-generierungswerkzeuge benötigt. Beispiel: Zum Testen der Skalierbarkeit einer beliebten Webseite kann die Simulation von Hunderttausenden von virtuellen Benutzern erforderlich sein. Werkzeuge, die knappe Ressourcen simulieren, sind vor allem für Stresstests besonders nützlich. Es sollte darauf geachtet werden,

© German Testing Board e.V.

dass alle zur Unterstützung der Tests beschafften Werkzeuge mit den Kommunikationsprotokollen kompatibel sind, die das System unter Test verwendet.

Der Lehrplan Performanztest [CT_PT_SYL] enthält weitere Einzelheiten zur Planung von Performanztests.

4.5.8 Spezifikation von Performanztests

Performanztests basieren im Wesentlichen auf dem Testen des Systems unter bestimmten Betriebsbedingungen. Für solche Tests ist ein Nutzungsprofil erforderlich, das definiert, wie das System voraussichtlich genutzt wird. Siehe Abschnitt 4.9 für weitere Einzelheiten zu Nutzungsprofilen.

Für Performanztests ist es oft erforderlich, die Systemlast zu ändern, indem Teile des Nutzungsprofils verändert werden, um eine Abweichung von der erwarteten Nutzung des Systems zu simulieren. Beispiel: Bei Kapazitätstests ist es meist notwendig, das Nutzungsprofil hinsichtlich der Variablen, deren Kapazität getestet wird, zu modifizieren (z.B. Erhöhung der Anzahl der Benutzer, die auf das System zugreifen, bis das System nicht mehr reagiert, um so die Kapazitätsgrenze für Benutzerzugriffe zu ermitteln). In ähnlicher Weise kann für Lasttests die Transaktionsrate schrittweise erhöht werden.

Der Lehrplan Performanztest [CT_PT_SYL] enthält weitere Einzelheiten zum Entwurf von Performanztests.

4.6 Wartbarkeitstest

Software verbringt oft einen wesentlich größeren Teil seines Lebenszyklus in der Wartung als in der Entwicklung. Um sicherzustellen, dass die Wartungsaufgabe so effizient wie möglich durchgeführt werden kann, wird im Rahmen der Wartbarkeitstests untersucht, wie leicht der Code analysiert, geändert, getestet, modularisiert und wiederverwendet werden kann. Wartbarkeitstests sollten nicht mit Wartungstests verwechselt werden, die durchgeführt werden, um die Auswirkungen von Änderungen einer im Betrieb laufenden Software zu testen.

Zu den typischen Wartbarkeitszielen der Stakeholder (z.B. Systemeigentümer oder -betreiber) gehören:

- Minimierung der Besitz- oder Betriebskosten der Software
- Minimierung der Ausfallzeiten für die Softwarewartung

Wartbarkeitstests sollten in der Testvorgehensweise enthalten sein, wenn einer oder mehrere der folgenden Faktoren:

© German Testing Board e.V.

- Änderungen der Software sind wahrscheinlich, nachdem diese produktiv gegangen ist (z.B. Fehlerbehebungen oder geplante Aktualisierungen)
- Nach Ansicht der Stakeholder überwiegt der Nutzen, der sich aus dem Erreichen der oben erwähnten Wartbarkeitsziele für den Softwareentwicklungs-lebenszyklus ergibt, die Kosten für die Durchführung der Wartbarkeitstests und das Einbringen der erforderlichen Änderungen.
- Das Risiko einer schlechten Wartbarkeit der Software (z.B. lange Reaktions-zeiten nach Fehlerzuständen, die von Anwendern und/oder Kunden berichtet werden) rechtfertigt die Durchführung von Wartbarkeitstests.

4.6.1 Statische und dynamische Wartbarkeitstests

Geeignete Verfahren für statische Wartbarkeitstests sind statische Analysen und Reviews, wie in den Abschnitten 3.2 und 5.2 beschrieben. Mit den Wartbarkeitstests sollte begonnen werden, sobald die Entwurfsdokumente zur Verfügung stehen, und sie sollten während der gesamten Phase der Code-Implementierung fortgesetzt werden. Da die Wartbarkeit in den Code und in die zugehörige Dokumentation jeder Komponente eingebaut ist, kann die Wartbarkeit schon früh im Software-entwicklungslebenszyklus bewertet werden; ohne zu warten, bis das System vollständig ist und läuft.

Dynamische Wartbarkeitstests untersuchen in erster Linie die dokumentierten Verfahren, die für die Wartung einer Anwendung entwickelt wurden (z.B. für die Durchführung von Software-Upgrades). Als Testfälle dienen Wartungsszenarien, um sicherzustellen, dass die geforderten Service-Levels mit den dokumentierten Verfahren erreicht werden können. Diese Art des Testens ist besonders bei komplexen Infrastrukturen wichtig, wenn mehrere Abteilungen/Unternehmen an den Supportverfahren beteiligt sind. Diese Tests können Teil der betrieblichen Abnahmetests sein.

4.6.2 Untermerkmale der Wartbarkeit

Die Wartbarkeit eines Systems lässt sich auf unterschiedliche Arten messen:

- Analysierbarkeit
- Modifizierbarkeit
- Testbarkeit

Zu den Faktoren, die diese Merkmale beeinflussen, gehören die Verwendung guter Programmierpraktiken (z.B. Kommentierung, Namensgebung für Variablen, Ein-rückung) und die Verfügbarkeit technischer Dokumentation (z.B. Spezifikationen für den Systementwurf, Schnittstellenspezifikationen).

© German Testing Board e.V.

Weitere relevante Untermerkmale des Qualitätsmerkmals Wartbarkeit [ISO25010]] sind:

- Modularität
- Wiederverwendbarkeit

Die Modularität kann mit Hilfe der statischen Analyse getestet werden (siehe Abschnitt 3.2.3). Das Testen der Wiederverwendbarkeit kann in Form von Architektur-Reviews erfolgen (siehe Kapitel 5).

4.7 Übertragbarkeitstest

4.7.1 Einführung

Übertragbarkeitstests untersuchen im Allgemeinen den Grad, zu dem eine Softwarekomponente oder ein System in eine vorgesehene Umgebung übertragen werden kann (entweder bei der Erstinstallation oder aus einer bestehenden Umgebung), an eine neue Umgebung angepasst werden kann oder eine andere Softwarekomponente oder ein System ersetzen kann.

Der Standard ISO 25010 [ISO 25010] enthält die folgenden Untermerkmale von Übertragbarkeit:

- Installierbarkeit
- Anpassbarkeit
- Austauschbarkeit

Übertragbarkeitstests können schon mit einzelnen Komponenten beginnen (z.B. Austauschbarkeit einer bestimmten Komponente wie beispielsweise Wechsel von einem Datenbankmanagementsystem auf ein anderes) und werden erweitert, wenn mehr Code zur Verfügung steht. Die Installierbarkeit ist möglicherweise erst dann testbar, wenn alle Komponenten des Produkts funktionieren.

Da die Übertragbarkeit beim Systementwurf berücksichtigt und in das Produkt eingebaut werden muss, ist dieses Qualitätsmerkmal schon in den frühen Phasen des Systementwurfs und der Systemarchitektur wichtig. Reviews des Entwurfs und der Architektur können ein sehr wirksames Mittel sein, um potenzielle Übertragbarkeitsanforderungen und -probleme zu identifizieren (z.B. die Abhängigkeit von einem bestimmten Betriebssystem).

© German Testing Board e.V.

4.7.2 Installierbarkeitstests

Installierbarkeitstests testen die Installation von Software in einer Zielumgebung samt den dokumentierten Installationsanleitungen. Diese Tests können auch Software adressieren, mit der ein Betriebssystem installiert wird, oder einen Installations-Wizard, mit dem ein Produkt auf einem Client-PC installiert wird.

Typische Ziele von Installierbarkeitstests sind:

- Validieren, dass die Software installiert werden kann, indem die Anweisungen in einem Installationshandbuch (einschließlich der Ausführung von Installationsskripten) befolgt werden oder ein Installations-Wizard verwendet wird. Dabei sind auch die unterschiedlichen Optionen für mögliche Hardware-/Software-Konfigurationen sowie für verschiedene Installationsstufen (Erstinstallation oder Update) zu testen.
- Testen, ob die Installationssoftware richtig mit Fehlerwirkungen umgeht, die bei der Installation auftreten (wenn z.B. bestimmte DLLs nicht geladen werden), ohne das System in einem undefinierten Zustand zu belassen (beispielsweise mit nur teilweise installierter Software oder inkorrekten Systemkonfigurationen)
- Testen, ob sich eine teilweise Installation/Deinstallation der Software abschließen lässt
- Testen, ob ein Installations-Wizard eine ungültige Hardware-Plattform oder Betriebssystemkonfigurationen identifizieren kann
- Messen, ob der Installationsvorgang im spezifizierten Zeitrahmen oder mit weniger als der spezifizierten Anzahl von Schritten abgeschlossen werden kann
- Validieren, dass sich eine frühere Version der Software installieren („Downgrade“ der aktuellen Software) oder, dass sich die Software deinstallieren lässt

Nach dem Installierbarkeitstest wird normalerweise die funktionale Eignung getestet, um Fehlerzustände aufzudecken, die durch die Installation eingeschleust worden sein können (beispielsweise inkorrekte Konfigurationen, nicht mehr verfügbare Funktionen). Parallel zu den Installierbarkeitstests werden in der Regel Gebrauchstauglichkeitstests durchgeführt (beispielsweise zur Validierung, dass die Anwender bei der Installation verständliche Anweisungen und Feedback/Fehlermeldungen erhalten).

4.7.3 Anpassbarkeitstests

Anpassbarkeitstests überprüfen, ob eine bestimmte Anwendung in allen vorgesehenen Zielumgebungen (Hardware, Software, Middleware, Betriebssystem usw.) korrekt funktioniert. Für die Spezifikation von Anpassbarkeitstests müssen die beabsichtigten Zielumgebungen identifiziert, konfiguriert und dem Testteam zur

© German Testing Board e.V.

Verfügung gestellt werden. Diese Umgebungen werden dann mit ausgewählten funktionalen Testfällen getestet, bei denen die verschiedenen Komponenten der Anwendung in der Testumgebung geprüft werden.

Anpassbarkeit kann sich darauf beziehen, dass sich die Software durch Ausführung eines vordefinierten Verfahrens auf verschiedene spezifizierte Umgebungen übertragen lässt. Beim Testen kann dieses Verfahren bewertet werden.

4.7.4 Austauschbarkeitstests

Austauschbarkeitstests überprüfen die Fähigkeit einer Softwarekomponente, eine vorhandene Softwarekomponente in einem System zu ersetzen. Das ist besonders relevant bei Systemen, die kommerzielle Standardsoftware für bestimmte Systemkomponenten verwenden, oder für IoT-Anwendungen.

Austauschbarkeitstests können parallel zu den funktionalen Integrationstests durchgeführt werden, wenn alternative Komponenten für die Integration in das Gesamtsystem verfügbar sind. Die Austauschbarkeit lässt sich auch in technischen Reviews oder Inspektionen der Systemarchitektur oder des Systementwurfs bewerten, bei denen Wert auf eine klare Definition der Schnittstellen der möglichen Austauschkomponenten gelegt wird.

4.8 Kompatibilitätstests

4.8.1 Einführung

Kompatibilitätstests untersuchen die folgenden Aspekte [ISO25010]:

- Koexistenz
- Interoperabilität

4.8.2 Koexistenztests

Computersysteme, die nicht miteinander interagieren, sind dann kompatibel, wenn sie in derselben Umgebung (beispielsweise auf derselben Hardware) ausgeführt werden können, ohne sich gegenseitig in ihrem Verhalten zu beeinflussen (beispielsweise durch Ressourcenkonflikte). Koexistenztests sollten durchgeführt werden, wenn neue oder aktualisierte Software in Umgebungen installiert wird, in denen bereits andere Anwendungen installiert sind.

Koexistenzprobleme können auftreten, wenn eine Anwendung in einer Umgebung getestet wird, in der sie die einzige installierte Anwendung ist (in der Inkompatibilitätsprobleme nicht erkannt werden können), und dann in einer anderen

© German Testing Board e.V.

Umgebung (z.B. in der Produktivumgebung) eingesetzt wird, in den auch anderen Anwendungen laufen.

Typische Ziele von Koexistenztests sind:

- Bewertung möglicher negativer Auswirkungen auf die funktionale Eignung, wenn Anwendungen in derselben Umgebung geladen werden (z.B. Konflikte der Ressourcennutzung, wenn mehrere Anwendungen auf demselben Server ausgeführt werden)
- Bewertung der Auswirkungen auf jede Anwendung, die sich aus Modifikationen oder dem Installieren einer aktuelleren Betriebssystemversion ergeben

Koexistenzprobleme sollten analysiert werden, wenn die vorgesehene Zielumgebung geplant wird, die Tests werden jedoch normalerweise erst nach erfolgreichem Abschluss der Systemtests durchgeführt.

4.9 Nutzungsprofile

Nutzungsprofile werden als Teil der Testspezifikation für mehrere nicht-funktionale Testarten verwendet, einschließlich Zuverlässigkeitstests und Performanztests. Sie sind besonders nützlich, wenn die zu testende Anforderung die Einschränkung "unter bestimmten Bedingungen" enthält, da sie dann zur Definition dieser Bedingungen verwendet werden können.

Das Nutzungsprofil definiert ein Nutzungsmuster für das System, typischerweise in Bezug auf die Benutzer des Systems und die vom System durchgeführten Operationen. Für die Benutzer wird normalerweise spezifiziert, wie viele von ihnen das System voraussichtlich nutzen werden (und zu welchen Zeiten), und welche Art von Benutzern (z.B. Primärbenutzer, Sekundärbenutzer) dies sein wird. Die verschiedenen Operationen, die vom System ausgeführt werden sollen, werden normalerweise zusammen mit ihrer Häufigkeit (und der Wahrscheinlichkeit ihres Auftretens) spezifiziert. Diese Informationen können mit Hilfe von Monitoren (bzw. Überwachungswerkzeugen) erfasst werden (falls die tatsächliche oder eine ähnliche Anwendung bereits verfügbar ist) oder durch eine Vorhersage der Nutzung auf der Grundlage von Algorithmen oder Schätzungen, die vom Geschäftsbereich des Unternehmens bereitgestellt werden.

Mit Hilfe von Werkzeugen können Testeingaben auf der Grundlage des Nutzungsprofils generiert werden, oft mit einem Ansatz, der die Testeingaben pseudo-zufällig generiert. Solche Werkzeuge können verwendet werden, um "virtuelle" oder simulierte Benutzer in Größenordnungen zu erzeugen, die dem Nutzungsprofil entsprechen (z.B. für Zuverlässigkeits- und Verfügbarkeitstests) oder sogar höher liegen (z.B. für Stress- oder Kapazitätstests). Siehe Abschnitt 6.2.3 für weitere Einzelheiten zu diesen Werkzeugen.

© German Testing Board e.V.

5. Reviews– 165 Min.

Schlüsselbegriffe

Review, technisches Review

Lernziele für "Reviews"

5.1 Aufgaben von Technical Test Analysten bei Reviews

TTA 5.1.1 (K2) Erklären, warum die Vorbereitung des Reviews für Technical Test Analysten wichtig ist

5.2 Checklisten in Reviews verwenden

TTA 5.2.1 (K4) Einen Architekturentwurf analysieren und Probleme anhand einer im Lehrplan enthaltenen Checkliste identifizieren

TTA 5.2.2 (K4) Ein Stück Programmcode oder Pseudo-Code analysieren und Probleme anhand einer im Lehrplan enthaltenen Checkliste identifizieren

5.1 Aufgaben von Technical Test Analysten bei Reviews

Technical Test Analysten müssen aktiv an den technischen Reviews teilnehmen und ihre individuelle Sichtweise einbringen. Alle Reviewteilnehmer sollten ein formales Reviewtraining erhalten haben, damit sie ihre jeweiligen Rollen besser verstehen. Zudem sollten sie vom Nutzen gut durchgeführter technischer Reviews überzeugt sein. Dazu gehört auch eine konstruktive Zusammenarbeit mit den Autoren bei der Beschreibung und Diskussion von Reviewbefunden. Für eine detaillierte Beschreibung technischer Reviews, einschließlich zahlreicher Review-Checklisten, siehe [Wiegers02]. Technical Test Analysten nehmen normalerweise an technischen Reviews und Inspektionen teil, bei denen sie Gesichtspunkte des Systemverhaltens einbringen, die von den Entwicklern möglicherweise übersehen werden. Darüber hinaus spielen Technical Test Analysten eine wichtige Rolle bei der Definition, Anwendung und Pflege von Review-Checklisten und bei der Bereitstellung von Informationen über den Schweregrad von Fehlerzuständen.

Unabhängig von der Art des Reviews muss der Technical Test Analyst ausreichend Zeit zur Vorbereitung bekommen. Die Vorbereitungszeit wird benötigt, um das Arbeitsergebnis zu prüfen, um die Dokumente, auf die verwiesen wird, zu prüfen und zu verifizieren, dass das Arbeitsergebnis mit diesen konsistent ist, sowie zu bestimmen, was im Arbeitsergebnis fehlt. Ohne eine angemessene Vorbereitungszeit könnte das Review auf eine redaktionelle Überarbeitung des Dokuments reduziert werden, anstatt ein echtes Review zu sein. Zu einem guten Review gehört es, die

© German Testing Board e.V.

Inhalte zu verstehen, zu bestimmen was fehlt, die Korrektheit hinsichtlich technischer Aspekte zu verifizieren, und zu verifizieren ob das beschriebene Produkt mit anderen Produkten, die entweder bereits entwickelt wurden oder sich in der Entwicklung befinden, konsistent ist. Beispiel: Beim Review eines Stufentestkonzepts für den Integrationstest muss der Technical Test Analyst auch die Objekte berücksichtigen, die integriert werden sollen. Wann werden sie für die Integration bereit sein? Gibt es Abhängigkeiten, die dokumentiert werden müssen? Sind Daten für den Test der Integrationspunkte verfügbar? Ein Review konzentriert sich nicht allein auf das Arbeitsergebnis, das geprüft wird, sondern muss auch die Interaktion des Arbeitsergebnisses mit anderen Arbeitsergebnissen des Systems berücksichtigen.

5.2 Checklisten in Reviews verwenden

Checklisten werden bei Reviews verwendet, damit die Teilnehmer angehalten sind, bestimmte Punkte im Laufe des Reviews zu verifizieren. Checklisten können dazu beitragen, Reviews zu entpersonalisieren, z.B. mit der Aussage "Dies ist dieselbe Checkliste, die für alle Reviews verwendet wird, nicht nur für das vorliegende Arbeitsergebnis". Checklisten können allgemein gehalten sein und für alle Reviews verwendet werden, oder sie können sich zielgerichtet mit bestimmten Qualitäts-merkmalen oder Themenbereichen befassen. Beispiel: Mit einer allgemeinen Checkliste könnte verifiziert werden, ob die Begriffe „soll” und „sollte” richtig verwendet sind, oder ob die Formatierung und ähnliche Elemente korrekt bzw. konform sind. Eine Checkliste könnte auf die IT-Sicherheit oder die Performanz des Systems fokussiert sein.

Die nützlichsten Checklisten sind die, die sukzessive von einem Unternehmen bzw. einer organisatorischen Einheit entwickelt wurden, da diese Checklisten folgendes wiedergeben:

- Art des Produkts
- Lokales Entwicklungsumfeld
  - Personal
  - Werkzeuge
  - Prioritäten
- Historie früherer Erfolge und gefundener Fehlerzustände
- Besondere Themen (z.B. Performanz, IT-Sicherheit)

Checklisten sollten an das Unternehmen und möglicherweise sogar an das jeweilige Projekt angepasst werden. Die in diesem Kapitel beschriebenen Checklisten sind lediglich Beispiele.

© German Testing Board e.V.

5.2.1 Architekturreviews

Die Softwarearchitektur besteht aus den grundlegenden Konzepten oder Eigenschaften eines Systems, die in seinen Elementen, Beziehungen und in den für Systementwurf und -entwicklung geltenden Grundsätzen verkörpert sind [ISO 42010].

Die für ein Architekturreview des Zeitverhaltens von Webseiten verwendeten Checklisten[1] könnten beispielsweise verifizieren, dass die folgenden Aspekte korrekt implementiert sind (zitiert aus [Web-2]):

- „Verbindungspooling“ – den für die Ausführung benötigten Zeitaufwand, der mit dem Aufbau von Datenbankverbindungen zusammenhängt, reduzieren und einen gemeinsamen Pool von Verbindungen schaffen
- Lastverteilung – gleichmäßige Verteilung der Last zwischen einer Menge von Ressourcen
- Verteilte Verarbeitung
- Caching – eine lokale Kopie der Daten verwenden, um Zugriffszeiten zu reduzieren
- Verzögerte Instanziierung (lazy instantiation)
- Nebenläufigkeit von Transaktionen
- Prozesstrennung zwischen OLTP (Online Transactional Processing) und OLAP (Online Analytical Processing)
- Replikation von Daten

5.2.2 Code-Reviews

Checklisten für Code-Reviews sind zwangsläufig code-nah. Sie sind dann am nützlichsten, wenn sie spezifisch auf eine Programmiersprache zugeschnitten sind. Die Einbeziehung von Anti-Patterns (d.h. von typischen Mustern schlechten Stils) auf Code-Ebene ist hilfreich, insbesondere für weniger erfahrene Softwareentwickler.

Die für Code-Reviews verwendeten Checklisten[1] könnten folgende Punkte enthalten:

Struktur

- Implementiert der Code den Entwurf vollständig und korrekt?
- Entspricht der Code den einschlägigen Programmierkonventionen?
- Ist der Code gut strukturiert, konsistent im Stil und einheitlich formatiert?
- Gibt es Prozeduren, die nicht aufgerufen oder nicht benötigt werden, oder gibt es unerreichbaren Code?

© German Testing Board e.V.

- Gibt es im Code Überbleibsel vom Testen (Platzhalter, Testroutinen)?
- Kann Code durch Aufrufe externer wiederverwendbarer Komponenten oder Bibliotheksfunktionen ersetzt werden?
- Gibt es Codeblöcke, die sich wiederholen und in einer einzigen Prozedur zusammengefasst werden könnten?
- Ist die Speichernutzung effizient?
- Wird Symbolik benutzt anstatt „magische Nummern”-Konstanten (engl. magic numbers) oder String-Konstanten?
- Gibt es Module, die übermäßig komplex sind und daher umstrukturiert oder in mehrere Module aufgeteilt werden sollten?

Dokumentation

- Ist der Code verständlich und angemessen dokumentiert und in einem leicht zu wartenden Kommentierungstil geschrieben?
- Passen alle Kommentare zum Code?
- Entspricht die Dokumentation den geltenden Standards?

Variablen

- Sind alle Variablen richtig definiert und liegt eine sinnvolle, konsistente und eindeutige Namensgebung vor?
- Gibt es redundante oder ungenutzte Variablen?

Arithmetische Operationen

- Wird im Code vermieden, dass Gleitkommazahlen auf Gleichheit geprüft werden?
- Verhindert der Code Rundungsfehler systematisch?
- Vermeidet der Code Additionen und Subtraktionen mit sehr unterschiedlich großen Zahlen?
- Werden Teiler (Divisoren) auf 0 oder auf Rauschen getestet?

Schleifen und Zweige

- Sind alle Schleifen, Verzweigungen und Logikkonstrukte vollständig, korrekt und richtig verschachtelt?
- Werden in IF-ELSEIF-Ketten die häufigsten Fälle zuerst getestet?

© German Testing Board e.V.

- Werden alle Fälle in einem IF-ELSEIF- oder CASE-Block behandelt, einschließlich der ELSE- oder DEFAULT-Klauseln?
- Ist für jede Case-Anweisung ein Standardwert vorgegeben?
- Sind die Abbruchbedingungen von Schleifen offensichtlich und ausnahmslos erreichbar?
- Sind die Index-Variablen oder Teilskripte unmittelbar vor der Schleife richtig initialisiert?
- Können Anweisungen, die sich innerhalb der Schleife befinden, auch außerhalb der Schleife platziert werden?
- Wird vermieden, dass der Code in der Schleife die Index-Variable manipuliert, oder diese nach Beendigung der Schleife verwendet?

Defensive Programmierung

- Werden Indizes, Zeiger und Teilskripte mit Bezug auf Arrays, Datensätze oder Dateigrenzen getestet?
- Werden importierte Daten und Eingabeparameter auf Gültigkeit und Vollständigkeit getestet?
- Sind alle Ausgangsvariablen zugewiesen?
- Wird in jeder Anweisung das richtige Datenelement verarbeitet?
- Wird jeder zugewiesene Speicher freigegeben?
- Werden bei Zugriffen auf externe Geräte Fehlerbehandlungen und Zeitüberschreitungen (Timeouts) verwendet?
- Wird vor einem Zugriff auf Dateien geprüft, ob diese existieren?
- Sind alle Dateien und Geräte in einem korrekten Zustand, wenn das Programm beendet wird?

© German Testing Board e.V.

6. Testwerkzeuge und Testautomatisierung – 180 Min.

Schlüsselbegriffe

datengetriebenes Testen, Emulator, Fehlereinfügung, Fehlereinpflanzung, Mitschnitt, modellbasiertes Testen (MBT), schlüsselwortgetriebenes Testen, Simulator, Testdurchführung

Lernziele für "Testwerkzeuge und Testautomatisierung"

6.1 Ein Testautomatisierungsprojekt definieren

TTA-6.1.1 (K2) Die Aktivitäten des Technical Test Analysten in Zusammenhang mit dem Aufsetzen eines Testautomatisierungsprojekts zusammenfassen

TTA-6.1.2 (K2) Die Unterschiede zwischen datengetriebener und schlüsselwortgetriebener Testautomatisierung zusammenfassen

TTA-6.1.3 K2) Die technischen Probleme zusammenfassen, die häufig dafür verantwortlich sind, dass Testautomatisierungsprojekte nicht die geplante Investitionsrendite erzielen

TTA-6.1.4 (K3) Schlüsselwörter erstellen, die auf einem vorgegebenen Geschäftsprozess basieren

6.2 Spezifische Testwerkzeuge

TTA-6.2.1 (K2) Den Zweck von Werkzeugen zur Fehlereinpflanzung und zum Fehlereinfügen zusammenfassen

TTA-6.2.2 (K2) Die wichtigsten Eigenschaften von Performanztestwerkzeugen sowie die Themen rund um deren Implementierung zusammenfassen

TTA-6.2.3 (K2) Die allgemeinen Verwendungszwecke von Werkzeugen für das webbasierte Testen erläutern

TTA-6.2.4 (K2) Erläutern, wie Werkzeuge das Konzept des modellbasierten Testens unterstützen

TTA-6.2.5 (K2) Darlegen, wie Werkzeuge eingesetzt werden können, um den Komponententest und den Build-Prozess zu unterstützen

TTA-6.2.6 (K2) Darlegen, wie Werkzeuge eingesetzt werden können, um das Testen mobiler Applikationen zu unterstützen

© German Testing Board e.V.

6.1 Ein Testautomatisierungsprojekt definieren

Um kosteneffektiv zu sein, müssen Testwerkzeuge (und insbesondere solche, die die Testausführung unterstützen) sorgfältig konzipiert und entworfen werden. Die Implementierung einer Testautomatisierungsstrategie ohne eine solide Testautomatisierungsarchitektur führt meist zu Werkzeuglandschaften, deren Wartung kostspielig ist, die ihren Zweck nicht ausreichend erfüllen und die die angestrebte Investitionsrendite (engl. Return on investment, ROI) nicht erzielen.

Ein Testautomatisierungsprojekt sollte als ein Softwareentwicklungsprojekt verstanden werden. Das bedeutet, dass Architekturdokumente, detaillierte Entwurfsdokumente, Reviews des Entwurfs und des Codes, Komponenten- und Komponentenintegrationstests, sowie ein abschließender Systemtest erforderlich sind. Beim Testen kann es zu unnötigen Verzögerungen oder Komplikationen kommen, wenn der verwendete Testautomatisierungscode instabil oder ungenau ist.

In Zusammenhang mit der Testautomatisierung führen Technical Test Analysten unter anderem folgende Aufgaben durch:

- Bestimmen, wer für die Testausführung verantwortlich sein wird (eventuell in Abstimmung mit einem Testmanager)
- Das für das Unternehmen am besten geeigneten Werkzeug, den Zeitrahmen, die Kompetenzen innerhalb des Teams, die Wartungsanforderungen usw. auswählen (dies könnte bedeuten, dass entschieden wird, ein eigenes Werkzeug zu entwickeln, anstatt eines zu beschaffen)
- Die Anforderungen für die Schnittstellen zwischen dem Automatisierungswerkzeug und anderen Werkzeugen (z.B. Testmanagementwerkzeug, Fehlermanagementwerkzeug, Werkzeuge für die kontinuierliche Integration) definieren
- Adapter entwickeln, die für die Schnittstelle zwischen dem Testausführungswerkzeug und der Software unter Test erforderlich sein können
- Den Ansatz für die Automatisierung auswählen, d.h. einen schlüsselwortgetriebenen oder datengetriebenen Ansatz (siehe Abschnitt 6.1.1)
- In Zusammenarbeit mit dem Testmanager die Kosten für Implementierung (einschließlich Schulung) schätzen. Bei einer agilen Softwareentwicklung erfolgt dies normalerweise in Projekt-/Sprintplanungssitzungen mit dem gesamten Team.
- Die Zeitplanung für das Automatisierungsprojekt erstellen und Zeit für die Wartung einplanen
- Die Test Analysten und Businessanalysten darin schulen, wie die Daten für die Automatisierung zu verwenden und zu liefern sind

© German Testing Board e.V.

- Bestimmen, wann und wie die automatisierten Tests ausgeführt werden
- Bestimmen, wie die Testergebnisse der automatisierten Tests mit denen der manuellen Tests kombiniert werden

In Projekten mit Schwerpunkt auf der Testautomatisierung kann auch ein Testautomatisierungsentwickler (TAE) mit vielen dieser Aufgaben beauftragt werden (für weitere Informationen siehe Lehrplan Testautomatisierungsentwickler [CT_TAE_SYL]). Je nach den Erfordernissen des Projekts und vorhandenen Präferenzen können bestimmte organisatorische Aufgaben auch von einem Testmanager übernommen werden. In der agilen Softwareentwicklung ist die Zuordnung dieser Aufgaben zu Rollen meist flexibler und weniger formal.

Diese Aktivitäten und die getroffenen Entscheidungen beeinflussen die Skalierbarkeit und die Wartbarkeit der Automatisierungslösung. Es muss ausreichend Zeit investiert werden, um die Optionen zu prüfen, verfügbare Werkzeuge und Technologien zu untersuchen und die Zukunftspläne des Unternehmens zu verstehen.

6.1.1 Den Automatisierungsansatz auswählen

In diesem Abschnitt werden die folgenden Faktoren behandelt, die den Test-automatisierungsansatz beeinflussen:

- Automatisieren über die grafische Benutzungsschnittstelle (engl. graphical user interface – GUI), die Programmierschnittstelle (eng. application programming interface – API) und die Kommandozeilenschnittstelle (engl. Command-line interface – CLI)
- Verwendung des datengetriebenen Testens
- Verwendung des schlüsselwortgetriebenen Testens
- Umgang mit Fehlerwirkungen und Ausfällen
- Berücksichtigen des Systemzustands

Der Lehrplan Testautomatisierungsentwickler [CT_TAE_SYL] enthält weitere Informationen über die Auswahl eines Automatisierungsansatzes.

Automatisieren über die GUI, API und CLI

Testautomatisierung ist nicht auf das Testen über die grafische Benutzungsschnittstelle beschränkt. Es gibt auch Werkzeuge, die das automatisierte Testen auf API-Ebene, durch eine Kommandozeilenschnittstelle und über sonstige Schnittstellen der Software unter Test unterstützen. Eine der ersten Entscheidungen, die der Technical Test Analyst treffen muss, ist die Bestimmung der am besten geeigneten Schnittstelle, die für die Testautomatisierung verwendet werden soll.

© German Testing Board e.V.

Allgemeine Testausführungswerkzeuge erfordern die Entwicklung von Adaptern für diese Schnittstellen; dieser Aufwand ist bei der Planung zu berücksichtigen.

Eine der Schwierigkeiten beim Testen über die grafische Benutzungsschnittstelle ist, dass sich diese im Zuge der Softwareentwicklung ändert. Je nach Entwurf des Testautomatisierungscodes kann dies zu einem erheblichen Wartungsaufwand führen. Beispiel: Wird die Mitschnittfunktion eines Testautomatisierungswerkzeugs verwendet, kann dies zur Folge haben, dass die automatisierten Testfälle (oft als Testskripte bezeichnet) nicht mehr wie gewünscht ausgeführt werden können, wenn sich die grafische Benutzungsschnittstelle ändert. Dies liegt daran, dass das aufgezeichnete Skript die Interaktionen mit den grafischen Objekten erfasst, während der Tester die Software manuell ausführt. Ändern sich die Objekte, auf die zugegriffen wird, dann müssen die aufgezeichneten Skripte auch aktualisiert und an diese Änderungen angepasst werden.

Mitschnittwerkzeuge können als geeignete Ausgangsbasis für die Entwicklung von Automatisierungsskripten verwendet werden. Der Tester zeichnet eine Testsitzung auf, und das aufgezeichnete Skript wird dann modifiziert, um die Wartbarkeit zu verbessern (indem beispielsweise Abschnitte im aufgezeichneten Skript durch wiederverwendbare Funktionen ersetzt werden).

Verwendung eines datengetriebenen Automatisierungsansatzes

Je nach Art der zu testenden Software können die für die einzelnen Tests verwendeten Daten unterschiedlich, die ausgeführten Testschritte aber fast identisch sein (z.B., wenn beim Testen der Fehlerbehandlung eines Eingabefelds mehrere ungültige Werte eingegeben und die jeweils ausgegebenen Fehlermeldungen überprüft werden). Es wäre nicht effizient für jeden dieser zu testenden Werte ein dediziertes automatisiertes Testskript zu entwickeln und zu warten. Dieses Problem wird üblicherweise gelöst, indem die Daten aus den Skripten in einen externen Speicher, z.B. eine Tabellenkalkulation oder eine Datenbank, verschoben werden. Es werden Funktionen erstellt, die auf die spezifischen Daten für jede Ausführung des Testskripts zugreifen. So kann mit einem einzigen Skript eine Menge von Testdaten abgearbeitet werden, die die Eingabewerte und erwarteten Ergebniswerte liefern (z.B. ein Wert, der in einem Textfeld angezeigt wird, oder eine Fehlermeldung). Dies wird als datengetriebenes Testen bezeichnet.

Bei diesem Ansatz werden zusätzlich zu den Testskripten, die die gelieferten Daten verarbeiten, ein Testrahmen und eine Infrastruktur benötigt, um die Ausführung des Skripts oder der Menge von Skripten zu unterstützen. Die eigentlichen Daten, die in der Tabellenkalkulation oder Datenbank gespeichert sind, werden von Test Analysten erstellt, die sich mit der Geschäftslogik der Software auskennen. In der agilen Softwareentwicklung kann auch ein Vertreter des Fachbereichs (z.B. der Product Owner) an der Definition der Daten beteiligt sein, insbesondere für Abnahmetests. Diese Arbeitsteilung ermöglicht es den für die Entwicklung von Testskripten

© German Testing Board e.V.

zuständigen Personen (z.B. dem Technical Test Analysten), sich auf die Implementierung der Automatisierungsskripte zu konzentrieren, während der Test Analyst Autor des eigentlichen Tests bleibt. In den meisten Fällen wird der Test Analyst für die Ausführung der Testskripte verantwortlich sein, nachdem die Automatisierung implementiert und getestet wurde.

Verwendung eines schlüsselwortgetriebenen Automatisierungsansatzes

Der sogenannte schlüsselwortgetriebene oder aktionswortgetriebene Ansatz geht einen Schritt weiter und trennt auch die Aktion, die mit den gelieferten Testdaten durchgeführt werden soll, vom Testskript [Buwalda01]. Um die durchzuführenden Aktionen vom Testskript zu trennen, wird eine abstrakte Sprache erstellt, die die auszuführenden Aktionen beschreibt, sie aber nicht direkt ausführt. Schlüsselwörter können sowohl für die abstrakten als auch für die konkreten Aktionen definiert werden. Beispiel: Schlüsselwörter für Geschäftsprozesse könnten sein: „Anmelden“, „Benutzer_anlegen“ oder „Benutzer_löschen“. Solche Schlüsselwörter beschreiben die abstrakten Aktionen, die in der Anwendungsdomäne ausgeführt werden. Konkrete Aktionen bezeichnen die Interaktion mit der Softwareschnittstelle selbst. Schlüsselwörter wie beispielsweise „Schaltfläche_betätigen“, „Aus_Liste_auswählen“ oder „Baum_durchlaufen“ können zum Testen jener Funktionen der grafischen Benutzungsoberfläche verwendet werden, die nicht genau zu den vorhandenen Schlüsselwörtern des Geschäftsprozesses passen. Schlüsselwörter können Parameter enthalten, z.B. könnte das Schlüsselwort "Anmelden" zwei Parameter haben: Benutzername und Passwort.

Sobald die zu verwendenden Schlüsselwörter und Daten für die Testskripte definiert sind, setzt der Testautomatisierer (z.B. Technical Test Analyst oder Testautomatisierungsentwickler) die geschäftsprozessbezogenen Schlüsselwörter und untergeordneten Aktionen in ausführbaren Testautomatisierungscode um. Die Schlüsselwörter und Aktionen können zusammen mit den vorgesehenen Daten in Tabellenkalkulationsprogrammen gespeichert oder direkt in spezielle Werkzeuge eingegeben werden, die die schlüsselwortgetriebene Testautomatisierung unterstützen. Das Testautomatisierungsframework implementiert die Schlüsselwörter als eine Menge von einer oder mehreren ausführbaren Funktionen oder Skripten. Werkzeuge lesen mit Schlüsselwörtern erstellte Testfälle und rufen die entsprechenden Testfunktionen oder Testskripte auf, die diese implementieren. Die ausführbaren Testskripte sind sehr modular implementiert, damit sie leicht einzelnen Schlüsselwörtern zugeordnet werden können. Für die Implementierung der modularen Testskripte sind Programmierkenntnisse erforderlich.

Diese klare Aufteilung der Kenntnisse über Geschäftslogik und der eigentlichen Programmierung zur Implementierung der Testautomatisierungsskripte sorgt für die effektivste Nutzung der Testressourcen. Der Technical Test Analyst kann in seiner Rolle als Testautomatisierer seine Programmierfähigkeiten effektiv einbringen, ohne

© German Testing Board e.V.

dass er dazu spezifische Fachkenntnisse verschiedenster Geschäftsbereiche benötigt.

Durch die Trennung des Codes von den sich veränderbaren Daten wird die Testautomatisierung von den Änderungen isoliert. Dies wiederum verbessert die Wartbarkeit des Codes insgesamt und steigert die Investitionsrendite der Testautomatisierung.

Umgang mit Fehlerwirkungen

Beim Entwerfen der Testautomatisierung ist es wichtig, Fehlerwirkungen der Software unter Test zu antizipieren und zu behandeln. Der Testautomatisierer muss festlegen, was die Testausführungssoftware tun sollte, wenn eine Fehlerwirkung auftritt. Sollte die Fehlerwirkung aufgezeichnet werden und die Tests fortgesetzt werden? Sollten die Tests abgebrochen werden? Kann die Fehlerwirkung durch eine bestimmte Aktion (z.B. Betätigen einer Schaltfläche in einem Dialogfenster) oder vielleicht durch eine Verzögerung im Test behandelt werden? Nicht behandelte Fehlerwirkungen in der Software unter Test können die Ergebnisse der nachfolgenden Tests unbrauchbar machen und Probleme mit dem Test verursachen, der zum Zeitpunkt des Auftretens der Fehlerwirkung ausgeführt wurde.

Berücksichtigen des Systemzustands

Es ist auch wichtig, den Zustand des Systems zu Beginn und am Ende jedes Tests zu berücksichtigen. Es kann erforderlich sein, dass das System nach Abschluss der Testausführung in einen vordefinierten Zustand zurückkehrt. Dadurch wird ermöglicht, dass eine automatisierte Testsuite wiederholt ausgeführt werden kann, ohne das System manuell zurückzusetzen. Dazu muss die Testautomatisierung beispielsweise Daten, die erstellt wurden, wieder löschen, oder den Status von Datensätzen in einer Datenbank ändern. Das Testautomatisierungsframework sollte sicherstellen, dass am Ende der Tests ein korrekter Abschluss erfolgt ist (d.h. abmelden, nachdem die Tests beendet sind).

6.1.2 Geschäftsprozesse für die Automatisierung modellieren

Um einen schlüsselwortgetriebenen Ansatz für die Testautomatisierung zu implementieren, müssen die zu testenden Geschäftsprozesse in der abstrakten schlüsselwortbasierten Sprache modelliert bzw. spezifiziert werden. Es ist wichtig, dass diese so gestaltet ist, dass die Benutzer (z.B. die Test Analysten im Projekt, beziehungsweise, bei einer agilen Softwareentwicklung, die Product Owner als Vertreter des Fachbereichs) intuitiv damit arbeiten können.

Schlüsselwörter werden in der Regel verwendet, um abstrakte Geschäftsinteraktionen mit einem System unter Test abzubilden. Beispiel: Das Schlüsselwort „Auftrag_stornieren” umfasst die Überprüfung, ob der Auftrag existiert, die

© German Testing Board e.V.

Verifizierung der Zugriffsrechte der Person, die die Stornierung veranlasst, die Anzeige des Auftrags, der storniert werden soll, und das Anfordern der Stornierungsbestätigung. Der Test Analyst verwendet Sequenzen von Schlüsselwörtern (z.B. „Anmelden", „Auftrag_auswählen", „Auftrag_stornieren") sowie die relevanten Testdaten, um die Testfälle zu spezifizieren.

Zu den Faktoren, die berücksichtigt werden sollten, gehören die folgenden:

- Je detaillierter die Schlüsselwörter, desto spezifischer sind die Szenarien, die abgedeckt werden können. Allerdings kann durch die abstrakte Sprache die Wartung komplexer werden.
- Wenn Test Analysten auch die konkreten Aktionen ("Schaltfläche_betätigen", "Aus_Liste_auswählen" usw.) spezifizieren, können die schlüsselwortgetriebenen Tests besser mit unterschiedlichen Situationen umgehen. Da diese Aktionen jedoch direkt mit der grafischen Benutzungsschnittstelle verbunden sind, kann für die Tests auch mehr Wartungsaufwand erforderlich werden, wenn es zu Änderungen kommt.
- Die Verwendung von Sammelbegriffen als Schlüsselwörter kann die Entwicklung einfacher, die Wartung aber komplizierter machen. Es kann beispielsweise sechs Schlüsselwörter geben, die zusammen einen Datensatz erzeugen. Die Erstellung eines abstrakten Schlüsselworts, das alle sechs Schlüsselwörter aufruft, ist jedoch nicht unbedingt der effizienteste Ansatz.
- Ganz gleich wieviel Analyse für die Schlüsselwortsprache aufgewendet wird, es wird immer wieder vorkommen, dass neue oder geänderte Schlüsselwörter benötigt werden. Ein Schlüsselwort hat zwei unterschiedliche Aspekte: die Geschäftslogik, die darin zum Ausdruck kommt, und die Automatisierungsfunktionalität, die es ausführt. Es muss daher ein Prozess gefunden werden, der beide Aspekte berücksichtigt.

Die schlüsselwortbasierte Testautomatisierung kann die Wartungskosten für die Testautomatisierung erheblich reduzieren. Sie aufzusetzen kann zwar anfangs kostspieliger sein, der Ansatz wird aber bei einer ausreichend langen Projektlaufzeit wahrscheinlich insgesamt kostengünstiger sein.

Der Lehrplan Testautomatisierungsentwickler [CT_TAE_SYL] enthält weitere Informationen über die Modellierung von Geschäftsprozessen für die Automatisierung.

6.2 Spezifische Testwerkzeuge

Dieser Abschnitt enthält eine Übersicht über die Werkzeuge, die von Technical Test Analysten zusätzlich zu den im Foundation Level-Lehrplan [CTFL_SYL] beschriebenen Werkzeugen mitunter eingesetzt werden.

© German Testing Board e.V.

Anmerkung: Detaillierte Informationen über Testwerkzeuge sind in den folgenden ISTQB®-Lehrplänen enthalten:

- Mobile Application Testing [CT_MAT_SYL]
- Performanztest [CT_PT_SYL]
- Model-Based Tester [CT_MBT_SYL]
- Testautomatisierungsentwickler [CT_TAE_SYL]

6.2.1 Fehlereinpflanzungswerkzeuge

Fehlereinpflanzungswerkzeuge modifizieren den zu testenden Code (möglicherweise unter Verwendung vordefinierter Algorithmen), um die Überdeckung zu prüfen, die durch bestimmte Tests erreicht wird. Bei systematischer Anwendung kann so die Qualität der Tests (d.h. ihre Fähigkeit, die hinzugefügten Fehler zu erkennen) bewertet und gegebenenfalls verbessert werden.

Werkzeuge zur Fehlereinpflanzung werden vor allem von Technical Test Analysten verwendet; sie können jedoch auch von Entwicklern eingesetzt werden, um neu entwickelten Code zu testen.

6.2.2 Fehlereinfügungswerkzeuge

Fehlereinfügungswerkzeuge liefern absichtlich falsche Eingaben an das Testobjekt, um sicherzustellen, dass die Software mit dem Fehlerzustand umgehen kann. Die eingefügten Eingaben verursachen negative Bedingungen, die eine Fehlerbehandlung auslösen (und testen) sollten. DieUnterbrechung des normalen Kontrollflusses während der Ausführung erhöht auch die Codeüberdeckung.

Werkzeuge zur Fehlereinfügung werden vor allem von Technical Test Analysten verwendet; sie können jedoch auch von Entwicklern eingesetzt werden, um neu entwickelten Code zu testen.

6.2.3 Performanztestwerkzeuge

Performanztestwerkzeuge haben die folgenden Hauptfunktionen:

- Lastgenerierung
- Messung, Überwachung, Visualisierung und Analyse des Antwortverhaltens des Systems unter bestimmter Last, um Einblicke in den Umgang von System- und Netzwerkkomponenten mit den Ressourcen zu liefern

Zur Lastgenerierung wird ein vordefiniertes Nutzungsprofil (siehe Abschnitt 4.9) als Skript implementiert. Das Skript kann zunächst für einen einzelnen Benutzer erfasst

© German Testing Board e.V.

(möglicherweise mit einem Mitschnittwerkzeug) und dann mittels des Performanz-testwerkzeugs für das spezifizierte Nutzungsprofil implementiert werden. Die Implementierung muss die Variationen der Daten pro Transaktion (oder Menge von Transaktionen) berücksichtigen

Performanztestwerkzeuge generieren Last, indem sie eine große Zahl von Benutzern ("virtuelle" Benutzer) simulieren, die gemäß ihrer festgelegten Nutzungsprofile Aufgaben ausführen und eine bestimmte Menge an Eingabedaten erzeugen. Im Gegensatz zu einzelnen automatisierten Skripten zur Testausführung erzeugen viele Performanztestskripte die Interaktionen mit dem System auf der Ebene des Kommunikationsprotokolls und nicht durch die Simulation von Benutzerinteraktion über eine grafische Benutzungsschnittstelle. Dadurch werden in der Regel weniger separate Testsitzungen benötigt. Einige Lastgenerierungswerkzeuge können die Anwendung auch über Benutzungsschnittstelle steuern, um so die Antwortzeiten des Systems unter Last genauer zu messen.

Performanztestwerkzeuge liefern eine Vielzahl von Messungen für eine weitergehende Analyse während oder nach Ausführung des Tests. Typische Metriken und Berichte dieser Testwerkzeuge sind:

- Anzahl simulierter Benutzer im Test
- Anzahl und Art der von den simulierten Anwendern erzeugten Transaktionen sowie Eingangsrate der Transaktionen
- Antwortzeiten für bestimmte, von den Benutzern angeforderte Transaktionen
- Berichte und Graphen, die Systemlast und Antwortzeiten gegenüberstellen
- Berichte über Ressourcennutzung (z.B. Auslastung im Zeitverlauf mit Minmal- und Maximalwerten)

Wichtige Faktoren beim Implementieren von Performanztestwerkzeugen sind:

- Die für die Lastgenerierung benötigte Hardware und Netzwerkbandbreiten
- Kompatibilität des Werkzeugs mit dem Kommunikationsprotokoll des Systems unter Test
- Ausreichende Flexibilität des Werkzeugs für die einfache Implementierung unterschiedlicher Nutzungsprofile
- Bereitstellung benötigter Überwachungs-, Analyse- und Berichtsfunktionen

Performanztestwerkzeuge werden in der Regel erworben und nicht selbst entwickelt, da ihre Entwicklung aufwändig ist. Es kann jedoch sinnvoll sein, ein spezifisches Performanztestwerkzeug selbst zu entwickeln, wenn technische Einschränkungen den Einsatz eines verfügbaren Produkts nicht zulassen, oder wenn das Lastprofil und die bereitzustellenden Einrichtungen im Vergleich zu den von kommerziellen Werkzeugen bereitgestellten Lastprofilen und Einrichtungen einfach sind.

© German Testing Board e.V.

Weitere Informationen über Performanztestwerkzeuge sind im Lehrplan Performanztest [CT_PT_SYL] enthalten.

6.2.4 Werkzeuge zum Testen von Webseiten

Es gibt eine Vielzahl von „Open Source“- und kommerziellen Spezialwerkzeugen für den Test von Webseiten. Die nachfolgende Liste enthält die Verwendungszwecke einiger gebräuchlicher webbasierter Testwerkzeuge:

- Hyperlink-Testwerkzeuge werden verwendet, um Webseiten zu scannen und zu prüfen, dass keine Hyperlinks ungültig sind
- HTML- und XML-Prüfwerkzeuge überprüfen, ob die HTML- und XML-Standards in den von der Webseite erstellten Seiten eingehalten werden
- Performanztestwerkzeuge testen das Serververhalten, wenn viele Benutzer eine Serververbindung herstellen
- Einfache Testausführungswerkzeuge, die mit unterschiedlichen Browsern funktionieren
- Werkzeuge zum Scannen des Servercodes, um verwaiste (nicht verlinkte) Dateien zu identifizieren, auf die zuvor von der Webseite zugegriffen wurde
- HTML-Syntaxprüfprogramme
- Cascading Style Sheet (CSS)-Prüfprogramme
- Werkzeuge zur Prüfung von Standardverletzungen (z.B. Abschnitt 508 des US-amerikanischen Zugänglichkeitsstandards, bzw. M/376 in Europa)
- Werkzeuge, die eine Reihe von IT-Sicherheitsproblemen identifizieren

Gute Quellen für „Open Source“-Werkzeuge für den webbasierten Test sind:

- Das World Wide Web Consortium (W3C) [Web-3]: Diese Organisation legt Standards für das Internet fest und stellt eine Vielzahl von Werkzeugen zur Verfügung, um Fehlerzustände zu identifizieren, die diese Standards verletzen.
- Die Web Hypertext Application Technology Working Group (WHATWG) [Web-5]: Diese Organisation legt HTML-Standards fest und verfügt über ein Werkzeug, das die HTML-Validierung durchführt [Web-6].

Einige Werkzeuge, die mit web crawler ausgerüstet sind, können auch Informationen über die Größe der Seiten, über die Dauer des Herunterladens, sowie über das Vorhandensein/Fehlen von Seiten liefern (z.B. HTTP-Fehler 404). Dies sind nützliche Informationen für Entwickler, Webmaster und Tester.

Test Analysten und Technical Test Analysten verwenden diese Werkzeuge vorwiegend im Systemtest.

© German Testing Board e.V.

6.2.5 Werkzeugunterstützung für modellbasiertes Testen

Modellbasiertes Testen (MBT) ist ein Testverfahren, bei dem ein Modell (wie z.B. endliche oder erweiterte Zustandsmaschinen) eingesetzt wird, um das erwartete Verhalten eines software-gesteuerten Systems während der Ausführung zu beschreiben. Kommerzielle MBT-Werkzeuge (siehe [Utting07]) liefern oft eine Funktion, der es dem Benutzer ermöglicht, das Modell "auszuführen", um interessante Ausführungssequenzen zu speichern und diese als Testfälle zu verwenden. Auch andere ausführbare Modelle, wie beispielsweise Petri-Netze und Zustandsautomaten, unterstützen MBT.

MBT-Modelle (und -werkzeuge) können eingesetzt werden, um große Mengen unterschiedlicher Ausführungssequenzen zu generieren. Ebenfalls kann mithilfe von MBT-Werkzeugen die potenziell große Menge möglicher generierter Pfade reduziert werden. Das Testen mit diesen Werkzeugen kann eine andere Sichtweise auf die zu testende Software bieten. Dies kann dazu führen, dass Fehlerzustände gefunden werden, die im funktionalen Test vielleicht übersehen wurden.

Weitere Informationen über Testwerkzeuge für modellbasiertes Testen sind im Lehrplan Model-Based Tester [CT_MBT_SYL] enthalten.

6.2.6 Komponententest- und Build-Werkzeuge

Auch wenn Komponententest- und Build-Automatisierungswerkzeuge im Grunde Entwicklungswerkzeuge sind, werden sie vielfach von Technical Test Analysten verwendet und gewartet. Dies trifft insbesondere im Kontext einer agilen Entwicklung zu.

Komponententestwerkzeuge sind häufig spezifisch für die Programmiersprache, die zur Programmierung der Komponente verwendet wird. Wenn beispielsweise Java als Programmiersprache verwendet wurde, dann könnte JUnit für die automatisierten Komponententests eingesetzt werden. Auch für viele andere Programmiersprachen gibt es spezifische Testwerkzeuge; diese werden unter dem Oberbegriff „xUnit-Test-Framework“ zusammengefasst. Diese Testrahmen können zum Beispiel Testobjekte für jede erzeugte Klasse generieren, was die Aufgaben der Programmierer beim Automatisieren von Komponententests wesentlich vereinfacht.

Einige Build-Automatisierungswerkzeuge ermöglichen es, nach jeder Änderung einer Softwarekomponente automatisch einen neuen Build-Lauf auszulösen. Nach Abschluss des Builds führen andere Werkzeuge die Komponententests automatisch aus. Dieser Grad der Automatisierung, bei der der Build-Prozess eingeschlossen ist, ist normalerweise Bestandteil kontinuierlicher Integrationsumgebungen.

Wenn sie korrekt aufgesetzt und konfiguriert werden, können sich diese Arten von Werkzeugen positiv auf die Qualität der Builds auswirken, die zum Testen freigegeben

© German Testing Board e.V.

werden. Falls durch eine Änderung eines Programmierers Regressionsfehler in den Build eingefügt werden, führt das meist dazu, dass einige der automatisierten Tests nicht mehr erfolgreich ausgeführt werden. Dies ermöglicht eine sofortige Untersuchung der Ursache der Fehlerwirkungen, bevor der Build für die Testumgebung freigegeben wird.

6.2.7 Werkzeugunterstützung für das Testen mobiler Applikationen

Emulatoren und Simulatoren werden zur Unterstützung des Testens von Anwendungen häufig eingesetzt.

Simulatoren

Ein Simulator einer mobilen Plattform modelliert die Laufzeitumgebung des mobilen Endgerätes. Auf einem Simulator getestete Applikationen werden zu einer eigenen (dedizierten) Version kompiliert, die im Simulator, jedoch nicht auf einem realen Gerät funktioniert. Beim Testen werden Simulatoren als Ersatz für echte Geräte verwendet, beschränken sich aber in der Regel auf anfängliche Funktionstests und die Simulation vieler virtueller Benutzer bei Lasttests. Simulatoren sind relativ einfach gehalten (im Vergleich zu Emulatoren) und können Tests schneller durchführen als ein Emulator. Die auf einem Simulator getestete Anwendung unterscheidet sich jedoch von derjenigen, die an die echten mobilen Endgeräte verteilt wird.

Emulatoren

Ein Emulator modelliert die Hardware und verwendet dieselbe Laufzeitumgebung wie die physische Hardware. Applikationen, die kompiliert wurden, um auf einem Emulator bereitgestellt und getestet zu werden, könnten auch auf dem echten mobilen Endgerät verwendet werden.

Ein Emulator kann ein Gerät allerdings nicht vollständig ersetzen, da sich der Emulator möglicherweise anders verhält als das Mobilgerät, das er versucht nachzuahmen. Darüber hinaus werden einige Funktionen wie (Multi-)Touchfunktionen, Beschleunigungsmesser usw. möglicherweise nicht unterstützt. Dies liegt zum Teil an den Einschränkungen der Plattform, auf der der Emulator ausgeführt wird.

Gemeinsame Aspekte

Simulatoren und Emulatoren werden häufig eingesetzt, um die Kosten von Testumgebungen zu reduzieren, indem sie echte Geräte ersetzen. Simulatoren und Emulatoren sind in der frühen Entwicklungsphase sehr nützlich, da sie normalerweise in die Entwicklungsumgebungen integriert sind und eine frühzeitige Bereitstellung, Testen und Überwachung von Applikationen ermöglichen. Um den Emulator oder Simulator zu verwenden, muss dieser gestartet werden, die erforderliche App muss auf ihm installiert werden und wird dann so getestet, als ob sie sich auf dem echten

© German Testing Board e.V.

Gerät befände. Jede Entwicklungsumgebung für mobile Betriebssysteme wird normalerweise mit einem eigenen Emulator oder Simulator geliefert. Es stehen auch Emulatoren und Simulatoren von Drittanbietern zur Verfügung.

In der Regel lassen sich verschiedene Nutzungsparameter auf den Emulatoren und Simulatoren einstellen. Diese Einstellungen können die Netzwerkemulation mit unterschiedlichen Geschwindigkeiten, Signalstärken und Paketverlusten, das Ändern der Ausrichtung, das Generieren von Unterbrechungen und die GPS-Standortdaten umfassen. Für einige dieser Einstellungen, beispielsweise für GPS-Standortdaten oder Signalstärken, kann dies sehr nützlich sein, da diese mit echten Geräten schwierig oder kostspielig zu replizieren sind.

Weitere Informationen sind im Lehrplan Mobile Application Testing [CT_MAT_SYL] enthalten.

© German Testing Board e.V.

7. Literaturhinweise

7.1 Normen und Standards

Referenz	Titel
[DO-178C]	DO-178C - Software Considerations in Airborne Systems and Equipment Certification, RTCA, 2011 Kapitel 2
[ISO 9126]	ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality Kapitel 4 und Anhang A
[ISO 25010]	ISO/IEC 25010: 2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models Kapitel 1, 4 und Anhang A
[ISO 29119]	ISO/IEC/IEEE 29119-4:2015, Software and systems engineering - Software testing - Part 4: Test techniques Kapitel 2
[ISO 42010]	ISO/IEC/IEEE 42010:2011, Systems and software engineering - Architecture description Kapitel 5
[IEC 61508]	IEC 61508-5:2010, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Part 5: Examples of methods for the determination of safety integrity levels Kapitel 2
[ISO 26262]	ISO 26262-1:2018, Road vehicles — Functional safety, Parts 1 to 12. Kapitel 2
[IEC 62443-3-2]	IEC 62443-3-2:2020, Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design Kapitel 4

7.2 ISTQB®-Dokumente

Referenz	Titel

© German Testing Board e.V.

[CTAL_TTA_OVIEW]	ISTQB® Technical Test Analyst Advanced Level Overview v4.0
[CT_SEC_SYL]	Security Testing Syllabus, Version 2016
[CT_TAE_SYL]	Test Automation Engineer Syllabus, Version 2017
[CTFL_SYL]	Foundation Level Syllabus, Version 2018
[CT_PT_SYL]	Performance Testing Syllabus, Version 2018
[CT_MBT_SYL]	Model-Based Testing Syllabus, Version 2015
[CT_TM_SYL]	Test Manager Syllabus, Version 2012
[CT_MAT_SYL]	Mobile Application Testing Syllabus, 2019
[ISTQB_GLOSSARY]	Glossary of Terms used in Software Testing, Version 3.5, 2020
[CT_AuT_SYL]	Automotive Software Tester, Version 2018

7.3 Fachliteratur

Referenz	**Titel**
[Anderson01]	Lorin W. Anderson, David R. Krathwohl (Hrsg.) “A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives”, Allyn & Bacon, 2001, ISBN 978-0801319037
[Andrist20]	Björn Andrist and Viktor Sehr, C++ High Performance: Master the art of optimizing the functioning of your C++ code, 2nd Edition, Packt Publishing, 2020, ISBN 978-1839216541
[Beizer90]	Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3
[Buwalda01]	Hans Buwalda, “Integrated Test Design and Automation”, Addison-Wesley Longman, 2001, ISBN 0-201-73725-6
[Kaner02]	Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4
[McCabe76]	Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976, pp. 308-320
[Utting07]	Mark Utting, Bruno Legeard, “Practical Model-Based Testing: A Tools Approach”, Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1
[Whittaker04]	James Whittaker and Herbert Thompson, “How to Break Software Security”, Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0
[Wiegers02]	Karl Wiegers, "Peer Reviews in Software: A Practical Guide", Addison-Wesley, 2002, ISBN 0-201-73485-0

© German Testing Board e.V.

7.4 Internetquellen

Die folgenden Referenzen verweisen auf Informationen im Internet. Diese Referenzen wurden zum Zeitpunkt der Veröffentlichung dieses Advanced Level Lehrplans überprüft. Das ISTQB® übernimmt keine Verantwortung dafür, wenn diese Referenzen nicht mehr verfügbar sind.

Referenz	**URL**
[GTB-GLOSSAR]	https://www.german-testing-board.info/lehrplaene/istqbr-certified-tester-schema/glossar/
[Web-1]	http://www.nist.gov (NIST National Institute of Standards and Technology)
[Web-2]	http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx
[Web-3]	http://www.W3C.org
[Web-4]	https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[Web-5]	https://whatwg.org
[Web-6]	https://validator.w3.org/
[Web-7]	https://dl.acm.org/doi/abs/10.1145/3340433.3342822

Kapitel 2:	[Web-7]
Kapitel 4:	[Web-1] [Web-4]
Kapitel 5:	[Web-2]
Kapitel 6:	[Web-3] [Web-5] [Web-6]

© German Testing Board e.V.

8. Anhang A: Übersicht über die Qualitätsmerkmale

Die folgende Tabelle vergleicht die Haupt- und Untermerkmale der Produktqualität, die im mittlerweile abgelösten ISO 9126-1 (und im Technical Test Analyst-Lehrplan 2012) verwendet wurden, mit denjenigen des neueren ISO 25010 (wie im vorliegenden Lehrplan verwendet).

Anmerkung: Funktionale Eignung und Gebrauchstauglichkeit sind Bestandteil des Test Analyst-Lehrplans.

ISO/IEC 25010	ISO/IEC 9126-1	Bemerkungen
Funktionale Eignung	**Funktionalität**	Neue Bezeichnung ist genauer und vermeidet Verwechslungen mit anderen Bedeutungen von “Funktionalität”
Funktionale Vollständigkeit		Überdeckung der angegebenen Bedarfe
Funktionale Richtigkeit	Richtigkeit	Allgemeiner als die Richtigkeit
Funktionale Angemessenheit	Angemessenheit	Überdeckung der impliziten Bedarfe
	Interoperabilität	Verschoben zu Kompatibilität
	Sicherheit	Jetzt ein eigenes Hauptmerkmal
Performanz	**Effizienz**	Umbenannt, um einen Konflikt mit der Definition von Effizienz in ISO/IEC 25062 zu vermeiden
Zeitverhalten	Zeitverhalten	
Ressourcennutzung	Ressourcennutzung	
Kapazität		Neues Untermerkmal
Kompatibilität		Neues Hauptmerkmal
Koexistenz	Koexistenz	Verschoben von Übertragbarkeit
Interoperabilität		Verschoben von Funktionalität

© German Testing Board e.V.

Gebrauchstauglich-keit	**Gebrauchstauglich-keit**	
Erkennbare Angemessenheit	Verständlichkeit	Neue Bezeichnung ist genauer
Erlernbarkeit	Erlernbarkeit	
Operabilität	Operabilität	
Benutzerfehlerschutz		Neues Untermerkmal
Ästhetik der Benutzungsschnitt-stelle	Attraktivität	Neue Bezeichnung ist genauer
Zugänglichkeit		Neues Untermerkmal
Zuverlässigkeit	**Zuverlässigkeit**	
Reife	Reife	
Verfügbarkeit		Neues Untermerkmal
Fehlertoleranz	Fehlertoleranz	
Wiederherstellbarkeit	Wiederherstellbarkeit	
IT-Sicherheit	**Sicherheit**	War bisher ein Untermerkmal
Vertraulichkeit		Neues Untermerkmal
Integrität		Neues Untermerkmal
Nichtabstreitbarkeit		Neues Untermerkmal
Zurechenbarkeit		Neues Untermerkmal
Authentizität		Neues Untermerkmal
Wartbarkeit	**Wartbarkeit**	
Modularität		Neues Untermerkmal
Wiederverwendbar-keit		Neues Untermerkmal
Analysierbarkeit	Analysierbarkeit	

© German Testing Board e.V.

Modifizierbarkeit	Stabilität	Genauere Bezeichnung, kombiniert Stabilität und Änderbarkeit
Testbarkeit	Testbarkeit	
Übertragbarkeit	**Portabilität**	
Anpassbarkeit	Anpassbarkeit	
Installierbarkeit	Installierbarkeit	
	Koexistenz	Verschoben zu Kompatibilität
Austauschbarkeit	Austauschbarkeit	

© German Testing Board e.V.

9. Index

© German Testing Board e.V.

© German Testing Board e.V.

© German Testing Board e.V.

© German Testing Board e.V.