ISTQB® Certified Tester

Lehrplan

Foundation Level

4.0.1

Deutschsprachige Ausgabe. Herausgegeben durch

Austrian Testing Board, German Testing Board e. V. &
Swiss Testing Board

Übersetzung des englischsprachigen Lehrplans des International Software Testing Qualifications Board (ISTQB®), Originaltitel: Certified Tester, Foundation Level Syllabus, Version 4.0

Urheberschutzvermerk

Änderungsübersicht der deutschsprachigen Ausgabe

Version	Datum	Bemerkung
CTFL V4.0.1 D	16.08.2023	Update Urheberschutzvermerk
CTFL V4.0 D	04.08.2023	Deutschsprachige Fassung des ISTQB-Release 4.0
2018 V3.1D	20.01.2020	Deutschsprachiges Wartungsrelease
2018 D	03.09.2018	Deutschsprachige Fassung des ISTQB-Release 2018
2011 1-0.2	01.07.2017	Deutschsprachiges Wartungsrelease
2011 1.0.1	19.04.2013	Deutschsprachiges Wartungsrelease
2011	01.08.2011	Wartungsrelease
2010 1.0.1	29.11.2010	Deutschsprachiges Wartungsrelease
2010	01.10.2010	Wartungsrelease
2007	01.12.2007	Wartungsrelease
2005	01.10.2005	Erstfreigabe der deutschsprachigen Fassung des ISTQB®-Lehrplans „Certified Tester Foundation Level“
ASQF V2.2	Juli 2003	ASQF Syllabus Foundation Level Version 2.2 „Lehrplan Grundlagen des Softwaretestens“

Änderungsübersicht Originalausgabe

Version	Gültig ab	Bemerkungen
CTFL v4.0	21.04.2023	CTFL v4.0 – Generelle Release Version
CTFL v3.1.1	01.07.2021	CTFL v3.1.1 – Update von Copyright und Logo
CTFL v3.1	11.11.2019	CTFL v3.1 – Wartungsrelease mit kleineren Updates
ISTQB 2018	27.04.2018	CTFL v3.0 – Generelle Release Version
ISTQB 2011	01.04.2011	Lehrplan Certified Tester Foundation Level, Wartungsrelease
ISTQB 2010	30.03.2010	Lehrplan Certified Tester Foundation Level, Wartungsrelease
ISTQB 2007	01.05.2007	Lehrplan Certified Tester Foundation Level, Wartungsrelease
ISTQB 2005	01.07.2005	Lehrplan Certified Tester Foundation Level v1.0
ASQF V2.2	07.2003	ASQF Syllabus Foundation Level, Version 2.2 “Lehrplan Grundlagen des Softwaretestens“
ISEB V2.0	25.02.1999	ISEB Lehrplan Software Testing Foundation v2.0

Zur besseren Lesbarkeit wird im gesamten Dokument auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Es wird das generische Maskulinum verwendet, wobei unterschiedliche Geschlechter gleichermaßen gemeint sind.

Inhaltsverzeichnis

Danksagung

Das englischsprachige Dokument wurde von der Generalversammlung des ISTQB® am 21. April 2023 formell freigegeben.

Es wurde gemeinsam von den Teams der ISTQB Foundation Level und Agile Arbeitsgruppe erstellt: Laura Albert, Renzo Cerquozzi (stellvertretende Leitung), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (geteilte Leitung), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (geteilte Leitung), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (stellvertretende Leitung), Yaron Tsubery, Eshraka Zakaria.

Das Team dankt Stuart Reid, Patricia McQuaid und Leanne Howard für ihr Technisches Review sowie dem Review-Team und den nationalen Mitgliedboards für ihre Anregungen und Beiträge.

Die folgenden Personen waren am Review, der Kommentierung und der Abstimmung zu diesem Lehrplan beteiligt: Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Thomas Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo und Zsolt Hargitai.

Die D.A.CH Arbeitsgruppe ‘Lokalisierung CTFL 4.0’ dankt den Reviewern für ihre Kommentare und Beiträge: Arne Becher, Florian Fieber, Jan Giesen, Sabine Gschwandtner, Andreas Hetz, Dr. Matthias Hamburg, Tobias Horn, Thomas Letzkus, Dr. Seyed Mohsen Ekssir Monfared,

Jörn Münzel, Paul Müller, Reto Müller, Manfred Oberlerchner, Horst Pohlmann, Nishan Portoyan, Prof. Dr. Ralf Reißing, Maud Schlich, Emmi Schuhmacher, Dr. Andreas Spillner, Dominik Weber, Stephan Weißleder, Marc-Florian Wendland, Carsten Weise, Yu Zou.

ISTQB-Arbeitsgruppe Foundation Level (Ausgabe 2018): Klaus Olsen (Leitung), Tauhida Parveen (stellvertretende Leitung), Rex Black (Projektleitung), Eshraka Zakaria, Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radoslaw Smilgin, Stephanie Ulrich, Steve Toms, Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher, Yaron Tsubery und allen nationalen Mitgliedboards für ihre Vorschläge.

ISTQB-Arbeitsgruppe Foundation Level (Ausgabe 2011): Thomas Müller (Leitung), Debra Friedenberg. Das Kernteam dankt dem Review-Team (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) und allen nationalen Mitgliedboards für die Anregungen zur aktuellen Version des Lehrplans.

ISTQB-Arbeitsgruppe Foundation Level (Ausgabe 2010): Thomas Müller (Leitung), Rahul Verma, Martin Klonk und Armin Beer. Das Kernteam dankt dem Review-Team (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) und allen nationalen Mitgliedboards für ihre Anregungen.

ISTQB-Arbeitsgruppe Foundation Level (Ausgabe 2007): Thomas Müller (Leitung), Dorothy Graham, Debra Friedenberg, und Erik van Veenendaal. Das Kernteam dankt dem Review-Team (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson und Wonil Kwon) und allen nationalen Mitgliedboards für ihre Anregungen.

ISTQB-Arbeitsgruppe Foundation Level (Ausgabe 2005): Thomas Müller (Leitung), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson und Erik van Veenendaal. Das Kernteam dankt dem Review-Team und allen nationalen Mitgliedboards für ihre Vorschläge.

0. Einführung in diesen Lehrplan

0.1 Zweck dieses Dokuments

Dieser Lehrplan bildet die Grundlage der internationalen Qualifikation für Softwaretester. Das German Testing Board e.V. (im Folgenden GTB genannt) hat diesen Lehrplan in Zusammenarbeit mit dem Austrian Testing Board (ATB) und dem Swiss Testing Board (STB) in die deutsche Sprache übersetzt. Das ISTQB® stellt den Lehrplan folgenden Adressaten zur Verfügung:

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

ATB, GTB, STB 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 Foundation Level im Softwaretest

Die Foundation-Level-Qualifikation richtet sich an alle, die im Bereich des Softwaretestens tätig sind. Dazu gehören Personen in Rollen wie Tester, Testanalysten, Testengineer, Testberater, Testmanager, Softwareentwickler und Mitglieder von Entwicklungsteams. Diese Foundation-Level-Qualifikation eignet sich auch für alle, die ein grundlegendes Verständnis für das Testen von Software erwerben möchten, wie z. B. Projektmanager, Qualitätsmanager, Product Owner, Softwareentwicklungsmanager, Systemanalytiker (Businessanalysten), IT-Leiter und Unternehmensberater. Inhaber des Foundation-Zertifikats können höhere Qualifikationen im Bereich Softwaretest erwerben.

0.3 Karriereweg für Tester

Das ISTQB®-Schema unterstützt Testexperten in allen Stufen ihrer Karriere und bietet ihnen sowohl eine breite als auch eine tiefe Wissensbasis. Personen, die die ISTQB® Foundation-Level-Zertifizierung erlangt haben, sind möglicherweise auch an den Core Advanced Levels

(Test Analyst, Technical Test Analyst und Test Manager) und den nachfolgenden Expert Levels (Test Management oder Improving the Test Process) interessiert. Wer Fähigkeiten in der Testtätigkeit in einer agilen Umgebung entwickeln möchte, könnte die Zertifizierungen Agile Technical Tester oder Agile Test Leadership at Scale in Betracht ziehen.

Der Spezialistenstrang bietet einen tiefen Einblick in Bereiche, die spezifischen Testansätze und Testaktivitäten beinhalten (z. B. Testautomatisierung, KI-Tests, modellbasiertes Testen, Testen mobiler Apps), die sich auf bestimmte Testbereiche beziehen (z. B. Performanztests, Gebrauchstauglichkeitstests, Abnahmetests, Sicherheitstests) oder die das Test-Know-how für bestimmte Branchendomänen bündeln (z. B. Automotive oder Gaming).

Aktuelle Informationen über das ISTQB Certified Tester Schema finden Sie unter www.istqb.org oder auf den Seiten der nationalen Boards, wie z. B. https://www.gtb.de/ (Deutschland), www.austriantestingboard.at (Österreich) oder swisstestingboard.org (Schweiz).

0.4 Geschäftlicher Nutzen

In diesem Abschnitt werden 14 geschäftliche Nutzen (Business Outcomes, BO) aufgeführt, die von einer Person erwartet werden, die die Foundation Level Zertifizierung bestanden hat.

Ein Foundation Level zertifizierter Tester kann.

FL-BO1 Verstehen, was Testen ist und warum es nützlich ist

FL-BO2 Die grundlegenden Konzepte des Testens von Software verstehen

FL-BO3 Den Testansatz und die anzuwendenden Aktivitäten in Abhängigkeit vom Kontext des Testens identifizieren

FL-BO4 Die Qualität der Dokumentation bewerten und verbessern

FL-BO5 Die Effektivität und Effizienz des Testens steigern

FL-BO6 Den Testprozess an den Softwareentwicklungslebenszyklus anpassen

FL-BO7 Grundsätze des Testmanagements verstehen

FL-BO8 Klare und verständliche Fehlerberichte schreiben und kommunizieren

FL-BO9 Die Faktoren, die die Prioritäten und den Aufwand für das Testen beeinflussen, verstehen

FL-BO10 Als Teil eines funktionsübergreifenden Teams arbeiten

FL-BO11 Risiken und Vorteile der Testautomatisierung kennen

FL-BO12 Wesentliche Fähigkeiten, die für das Testen erforderlich sind, erkennen

FL-BO13 Die Auswirkungen von Risiken auf das Testen verstehen

FL-BO14 Über den Testfortschritt und die Qualität effektiv berichten

0.5 Prüfbare Lernziele und kognitive Stufen des Wissens

Die Lernziele (LO vom Englischen Learning Objectives) unterstützen den geschäftlichen Nutzen und dienen zur Ausarbeitung der Prüfungen für die Zertifizierung als Certified Tester Foundation Level. Im Allgemeinen sind alle Inhalte der Kapitel 1-6 dieses Lehrplans auf K1-Stufe prüfbar. Das heißt, vom Prüfling kann gefordert werden, einen Schlüsselbegriff oder ein Konzept aus einem der sechs Kapitel wiederzuerkennen, sich daran zu erinnern oder wiedergeben zu können. Die Stufen der spezifischen Lernziele werden am Anfang jedes Kapitels genannt und wie folgt klassifiziert:

- K1: Sich erinnern
- K2: Verstehen
- K3: Anwenden

Weitere Einzelheiten und Beispiele für Lernziele werden in Anhang 8 aufgezeigt. Alle Begriffe, die als Schlüsselbegriffe direkt unter den Kapitelüberschriften aufgelistet sind, müssen bekannt sein (K1), auch wenn sie nicht ausdrücklich in den Lernzielen erwähnt werden.

0.6 Die Foundation-Level-Zertifizierungsprüfung

Die Foundation-Level-Zertifizierungsprüfung basiert auf diesem Lehrplan. Die Beantwortung der Prüfungsfragen kann die Nutzung von Inhalten aus mehr als einem Abschnitt dieses Lehrplans erfordern. Alle Abschnitte des Lehrplans sind prüfungsrelevant, mit Ausnahme der Einführung und der Anhänge. Standards und Bücher sind als Referenzen genannt (Kapitel 7), ihr Inhalt ist jedoch nicht prüfungsrelevant, abgesehen von dem, was im Lehrplan selbst aus diesen Standards und Büchern zusammengefasst ist. Siehe dazu das Dokument “Foundation Level Examination Structures and Rules”.

0.7 Akkreditierung

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

0.8 Umgang mit Standards

Im Foundation Lehrplan wird auf Normen verwiesen (z. B. IEEE- oder ISO-Normen). Diese Verweise dienen als Rahmen (wie die Verweise auf ISO 25010 bezüglich der

Qualitätsmerkmale) oder als Quelle für zusätzliche Informationen, falls der Leser dies wünscht. Die Inhalte der Standards sind nicht prüfungsrelevant. Weitere Informationen über Normen sind in Kapitel 7 nachlesbar.

0.9 Auf dem Laufenden bleiben

Die Softwarebranche verändert sich schnell. Um diesen Veränderungen Rechnung zu tragen und den Beteiligten Zugang zu relevanten und aktuellen Informationen zu verschaffen, haben die ISTQB-Arbeitsgruppen auf der Website www.istqb.org Links angelegt, die auf unterstützende Dokumentation und Änderungen von Standards verweisen. Diese Informationen sind im Rahmen des Foundation Level Lehrplans nicht prüfungsrelevant.

0.10 Detaillierungsgrad

Der Detaillierungsgrad dieses Lehrplans erlaubt international einheitliche Schulungen und Prüfungen. Um dieses Ziel zu erreichen, enthält der Lehrplan Folgendes:

- Allgemeine Lehrziele, die die Intention des Foundation Levels beschreiben
- Eine Liste von Begriffen (Schlüsselbegriffe), an die sich die Lernende erinnern müssen
- Lernziele für jeden Wissensbereich, die die zu erreichenden kognitiven Lernergebnisse beschreiben
- Eine Beschreibung der wichtigsten Konzepte, einschließlich Verweisen auf anerkannte Quellen

Der Inhalt des Lehrplans ist keine Beschreibung des gesamten Wissensgebiets „Softwaretesten“. Er spiegelt den Detaillierungsgrad wider, der in Foundation-Level-Schulungen abgedeckt wird. Der Schwerpunkt liegt auf Konzepten und Verfahren des Testens, die auf alle Softwareprojekte angewendet werden können, unabhängig vom verwendeten Softwareentwicklungslebenszyklus.

0.11 Gliederung des Lehrplans

Es gibt sechs Kapitel mit prüfungsrelevantem Inhalt. Die Hauptüberschrift eines jeden Kapitels gibt die Schulungszeit für das Kapitel an. Für die Unterkapitel wird keine Zeitangabe gemacht. Für akkreditierte Schulungen fordert der Lehrplan mindestens 1135 Minuten (18 Stunden und 55 Minuten) Unterricht, die sich wie folgt auf die sechs Kapitel verteilen:

- Kapitel 1: Grundlagen des Testens (180 Minuten)
  - Der Lernende eignet sich die grundlegenden Prinzipien des Testens, die Gründe warum Testen notwendig ist und was Ziele des Testens sind, an.
  - Der Lernende versteht den Testprozess, die wichtigsten Testaktivitäten und Testmittel.
  - Der Lernende versteht die wesentlichen Fähigkeiten zum Testen.
- Kapitel 2: Testen während des Softwareentwicklungslebenszyklus (130 Minuten)
  - Der Lernende eignet sich an, wie das Testen in verschiedene Entwicklungsvorgehensweisen integriert wird.
  - Der Lernende eignet sich die Konzepte von Test-First-Ansätzen und DevOps kennen.
  - Der Lernende lernt die verschiedenen Teststufen, Testarten und den Wartungstest kennen.
- Kapitel 3: Statisches Testen (80 Minuten)
  - Der Lernende eignet sich die Grundlagen des statischen Testens, den Feedback- und den Reviewprozess an.
- Kapitel 4: Testanalyse und -entwurf (390 Minuten)
  - Der Lernende lernt Black-Box-, White-Box- und erfahrungsbasierte Testverfahren anzuwenden, um Testfälle aus verschiedenen Arbeitsergebnissen der Softwareentwicklung abzuleiten.
  - Der Lernende lernt den auf Zusammenarbeit basierenden Testansatz kennen.
- Kapitel 5: Management der Testaktivitäten (335 Minuten)
  - Der Lernende eignet sich an, wie man Tests im Allgemeinen plant und wie man den Testaufwand schätzt.
  - Der Lernende eignet sich an, wie Risiken den Umfang des Testens beeinflussen können.
  - Der Lernende eignet sich an, wie man Testaktivitäten überwacht und steuert.
  - Der Lernende eignet sich an, wie das Konfigurationsmanagement das Testen unterstützt.
  - Der Lernende eignet sich an, wie man Fehlerzustände klar und verständlich berichtet.
- Kapitel 6: Testwerkzeuge (20 Minuten)
  - Der Lernende lernt, Testwerkzeuge zu klassifizieren und die Risiken und Nutzen von Testautomatisierung zu verstehen.

1. Grundlagen des Testens - 180 Minuten

Schlüsselbegriffe

Debugging, Fehlerwirkung, Fehlerzustand, Fehlhandlung, Grundursache, Qualität, Qualitätssicherung, Testablauf, Testabschluss, Testanalyse, Testbasis, Testbedingung, Testdaten, Testdurchführung, Testen, Testentwurf, Testergebnis, Testfall, Testmittel, Testobjekt, Testplanung, Testrealisierung, Teststeuerung, Testüberwachung, Testziel, Überdeckung, Validierung, Verifizierung

Lernziele für Kapitel 1: Der Lernende kann ...

1.1 Was ist Testen?

FL-1.1.1 (K1) ... typische Testziele identifizieren

FL-1.1.2 (K2) ... Testen von Debugging unterscheiden

1.2 Warum ist Testen notwendig?

FL-1.2.1 (K2) ... Beispiele geben, warum Testen notwendig ist

FL-1.2.2 (K1) ... die Beziehung zwischen Testen und Qualitätssicherung wiedergeben

FL-1.2.3 (K2) ... zwischen Grundursache, Fehlhandlung, Fehlerzustand und Fehlerwirkung unterscheiden

1.3 Grundsätze des Testens

FL-1.3.1 (K2) ... die sieben Grundsätze des Testens erklären

1.4 Testaktivitäten, Testmittel und Rollen des Testens

FL-1.4.1 (K2) ... die verschiedenen Testaktivitäten und -aufgaben zusammenfassen

FL-1.4.2 (K2) ... die Auswirkungen des Kontexts auf den Testprozess erklären

FL-1.4.3 (K2) ... Testmittel, die die Testaktivitäten unterstützen, unterscheiden

FL-1.4.4 (K2) ... die Bedeutung der Pflege der Verfolgbarkeit erklären

FL-1.4.5 (K2) ... die verschiedenen Rollen beim Testen vergleichen

1.5 Grundlegende Kompetenzen und gute Praktiken beim Testen

FL-1.5.1 (K2) ... Beispiele für die allgemeinen Kompetenzen, die für das Testen erforderlich sind, geben

FL-1.5.2 (K1) ... die Vorteile des Whole-Team-Ansatzes wiedergeben

FL-1.5.3 (K2) ... die Vor- und Nachteile des unabhängigen Testens unterscheiden

1.1 Was ist Testen?

Softwaresysteme sind ein integraler Bestandteil unseres täglichen Lebens. Die meisten Menschen haben bereits Erfahrung mit Software gemacht, die nicht wie erwartet funktioniert. Software, die nicht ordnungsgemäß funktioniert, kann zu vielen Problemen führen, unter anderem zu Geld-, Zeit- oder Ansehensverlust, und in Extremfällen sogar zu Verletzung oder Tod. Softwaretests bewerten die Qualität der Software und helfen, das Risiko einer Fehlerwirkung im Betrieb zu verringern.

Das Testen von Software ist eine Reihe von Aktivitäten zur Entdeckung von Fehlerzuständen und zur Bewertung der Qualität von Softwareartefakten. Werden diese Artefakte getestet, werden sie als Testobjekte bezeichnet. Ein weit verbreitetes Missverständnis über das Testen ist, dass es nur aus dem Ausführen von Tests besteht (d. h. dem Ausführen der Software und der Prüfung der Testergebnisse). Das Testen von Software umfasst jedoch auch andere Aktivitäten und muss auf den Softwareentwicklungslebenszyklus (Software Development Life Cycle, SDLC) abgestimmt sein (siehe Kapitel 2).

Ein weiteres verbreitetes Missverständnis über das Testen ist, dass sich das Testen ausschließlich auf das Verifizieren des Testobjekts konzentriert. Zwar beinhaltet Testen das Verifizieren, d. h. das Prüfen, ob das System die spezifizierten Anforderungen erfüllt, aber auch das Validieren, d. h. das Prüfen, ob das System die Bedürfnisse der Benutzer und anderer Stakeholder in seiner Betriebsumgebung erfüllt.

Testen kann dynamisch oder statisch sein. Beim dynamischen Test wird die Software ausgeführt, beim statischen Test hingegen nicht. Zum statischen Test gehören Reviews (siehe Kapitel 3) und statische Analysen. Beim dynamischen Test werden verschiedene Testverfahren und Testansätzen verwendet, um Testfälle abzuleiten (siehe Kapitel 4).

Testen ist nicht nur eine technische Aktivität. Es muss auch richtig geplant, verwaltet, geschätzt, überwacht und gesteuert werden (siehe Kapitel 5).

Tester verwenden Werkzeuge (siehe Kapitel 6), aber es ist wichtig, sich daran zu erinnern, dass Testen eine weitgehend intellektuelle Aktivität ist. Das erfordert von den Testern Fachwissen, die Anwendung analytischer Fähigkeiten und den Einsatz kritischen Denkens sowie Systemdenken (Myers 2011, Roman 2018).

Die Norm ISO/IEC/IEEE 29119-1 liefert weitere Informationen über Konzepte des Softwaretestens.

1.1.1 Testziele

Typische Testziele sind:

- Evaluieren von Arbeitsergebnissen wie Anforderungen, User Storys, Entwürfe und Code
- Auslösen von Fehlerwirkungen und Finden von Fehlerzuständen
- Sicherstellen der erforderlichen Überdeckung eines Testobjekts

- Verringern des Risikos einer unzureichenden Softwarequalität
- Verifizieren, ob spezifizierte Anforderungen erfüllt wurden
- Verifizieren, ob ein Testobjekt den vertraglichen, rechtlichen und regulatorischen Anforderungen entspricht
- Bereitstellen von Informationen für die Stakeholder, damit diese fundierten Entscheidungen treffen können
- Aufbauen von Vertrauen in die Qualität des Testobjekts
- Validieren, ob das Testobjekt vollständig ist und aus Sicht der Stakeholder wie erwartet funktioniert.

Ziele des Testens können je nach Kontext variieren. Zum Kontext gehören das zu testende Arbeitsergebnis, die Teststufe, Risiken, der Softwareentwicklungslebenszyklus und Faktoren im Zusammenhang mit dem geschäftlichen Kontext, z. B. die Unternehmensstruktur, Wettbewerbserwägungen oder die Zeit bis zur Markteinführung.

1.1.2 Testen und Debugging

Testen und Debugging sind getrennte Aktivitäten. Testen kann Fehlerwirkungen auslösen, die durch Fehlerzustände in der Software verursacht werden (dynamischer Test), oder kann direkt Fehlerzustände im Testobjekt finden (statischer Test).

Wenn ein dynamischer Test (siehe Kapitel 4) eine Fehlerwirkung auslöst, geht es beim Debugging darum, die Ursachen für diese Fehlerwirkung (die Fehlerzustände) zu finden, diese zu analysieren und zu beseitigen. Der typische Debugging-Prozess umfasst in diesem Fall:

- Reproduzieren einer Fehlerwirkung
- Diagnose (Befund der Grundursache)
- Behebung der Ursache

Anschließende Fehlernachtests prüfen, ob das Problem durch die Korrekturen behoben wurde. Vorzugsweise wird der Fehlernachtest von derselben Person durchgeführt, die auch den ersten Test durchgeführt hat. Anschließende Regressionstests können ebenfalls durchgeführt werden, um zu prüfen, ob die Korrekturen in anderen Teilen des Testobjekts Fehlerwirkungen verursachen (für Informationen über Fehlernachtests und Regressionstests siehe Abschnitt 2.2.3).

Wenn beim statischen Test ein Fehlerzustand festgestellt wird, geht es beim Debugging darum, diesen zu beseitigen. Reproduktion oder Diagnose sind nicht erforderlich, da statische Tests direkt Fehlerzustände finden und keine Fehlerwirkungen auslösen können (siehe Kapitel 3).

1.2 Warum ist Testen notwendig?

Testen, als eine Form der Qualitätssteuerung, trägt dazu bei, die vereinbarten Ziele innerhalb des festgelegten Umfangs sowie der Zeit-, Qualitäts- und Budgetvorgaben zu erreichen. Der Beitrag des Testens zum Erfolg sollte nicht auf die Aktivitäten des Testteams beschränkt sein. Jeder Stakeholder kann seine Testkompetenzen einsetzen, um das Projekt dem Erfolg näher zu bringen. Das Testen von Komponenten, Systemen und der zugehörigen Dokumentation hilft bei der Ermittlung von Fehlerzuständen in der Software.

1.2.1 Der Beitrag des Testens zum Erfolg

Testen ist ein kosteneffizientes Mittel zur Erkennung von Fehlerzuständen. Diese Fehlerzustände können dann beseitigt werden (durch Debugging – eine Aktivität, die nicht zum Testen gehört), so dass das Testen indirekt zu einer höheren Qualität der Testobjekte beiträgt.

Testen bietet ein Mittel zur direkten Bewertung der Qualität eines Testobjekts in verschiedenen Phasen des SDLC. Diese Messgrößen werden als Teil einer größeren Projektmanagementaktivität verwendet und tragen zu Entscheidungen für den Übergang zur nächsten Phase des SDLC bei, z. B. zur Freigabeentscheidung.

Testen bietet den Benutzern eine indirekte Darstellung des Entwicklungsprojekts. Tester stellen sicher, dass ihr Verständnis für die Bedürfnisse der Benutzer während des gesamten Entwicklungszyklus berücksichtigt wird. Die Alternative besteht darin, eine repräsentative Gruppe von Benutzern in das Entwicklungsprojekt einzubeziehen, was in der Regel aufgrund der hohen Kosten und der mangelnden Verfügbarkeit geeigneter Benutzer nicht möglich ist.

Testen kann auch erforderlich sein, um vertragliche oder gesetzliche Anforderungen zu erfüllen oder um regulatorischen Standards zu entsprechen.

1.2.2 Testen und Qualitätssicherung

Obwohl die Begriffe "Testen" und "Qualitätssicherung" (oder kurz QS) häufig synonym verwendet werden, sind Testen und Qualitätssicherung nicht dasselbe. Testen ist eine Form der Qualitätssteuerung.

Qualitätssteuerung ist ein produktorientierter, korrigierender Ansatz, der sich auf jene Aktivitäten konzentriert, die das Erreichen eines angemessenen Qualitätsniveaus unterstützen. Testen ist eine der wichtigsten Formen der Qualitätssteuerung, andere sind formale Methoden (Modellprüfung und Korrektheitsnachweis), Simulation und Prototyping.

Qualitätssicherung ist ein prozessorientierter, präventiver Ansatz, der sich auf die Implementierung und Verbesserung von Prozessen konzentriert. Sie geht davon aus, dass ein guter Prozess, wenn er korrekt durchgeführt wird, ein gutes Produkt hervorbringt. Qualitätssicherung bezieht sich sowohl auf den Entwicklungs- als auch auf den Testprozess und liegt in der Verantwortung aller Projektbeteiligten.

Testergebnisse werden von Qualitätssicherung und Qualitätssteuerung verwendet. In der Qualitätssteuerung werden sie zur Behebung von Fehlerzuständen verwendet, während sie in

der Qualitätssicherung Rückmeldungen darüber liefern, wie gut die Entwicklungs- und Testprozesse funktionieren.

1.2.3 Fehlhandlungen, Fehlerzustände, Fehlerwirkungen und Grundursachen

Menschen begehen Fehlhandlungen (Irrtümer), die zu Fehlerzuständen (Defekten) führen, was wiederum zu Fehlerwirkungen führen kann. Menschen machen aus verschiedenen Gründen Fehlhandlungen, wie z. B. durch Zeitdruck, Komplexität von Arbeitsergebnissen, Prozessen, Infrastruktur oder Interaktionen, oder einfach, weil sie erschöpft sind oder nicht ausreichend geschult wurden.

Fehlerzustände können in der Dokumentation, z. B. in einer Anforderungsspezifikation oder einem Testskript, im Quellcode oder in einem unterstützenden Artefakt, z. B. einer Build-Datei, gefunden werden. Fehlerzustände in Artefakten, die zu einem früheren Zeitpunkt im SDLC erstellt wurden, führen, wenn sie unentdeckt bleiben, häufig zu fehlerhaften Artefakten im späteren Verlauf des Lebenszyklus. Wenn ein Fehlerzustand im Code ausgeführt wird, kann es sein, dass das System nicht das tut, was es tun sollte, oder etwas tut, was es nicht tun sollte, was zu einer Fehlerwirkung führt. Einige Fehlerzustände führen, wenn sie ausgeführt werden, immer zu einer Fehlerwirkung, während andere nur unter bestimmten Umständen zu einer Fehlerwirkung führen, und wieder andere führen nie zu einer Fehlerwirkung.

Fehlhandlungen und Fehlerzustände sind nicht die einzige Ursache von Fehlerwirkungen. Fehlerwirkungen können auch durch Umweltbedingungen verursacht werden, z. B. wenn Strahlung oder elektromagnetische Felder Fehlerzustände in der Firmware verursachen.

Eine Grundursache (root cause) ist ein wesentlicher Grund für das Auftreten eines Problems (z. B. eine Situation, die zu einer Fehlhandlung führt). Grundursachen werden durch eine Grundursachenanalyse ermittelt, die normalerweise durchgeführt wird, wenn eine Fehlerwirkung auftritt oder ein Fehlerzustand festgestellt wird. Es wird davon ausgegangen, dass weitere ähnliche Fehlerwirkungen oder Fehlerzustände verhindert oder ihre Häufigkeit verringert werden können, wenn die Grundursache angegangen wird, z. B. durch ihre Beseitigung.

1.3 Grundsätze des Testens

Im Laufe der Jahre wurde eine Reihe von Grundsätzen des Testens angeregt, die allgemeine Richtlinien für alle Tests bieten. Dieser Lehrplan beschreibt sieben solcher Grundsätze.

1. Testen zeigt das Vorhandensein, nicht die Abwesenheit von Fehlerzuständen. Testen kann zeigen, dass Fehlerzustände im Testobjekt vorhanden sind, kann aber nicht beweisen, dass es keine Fehlerzustände gibt (Buxton 1970). Testen verringert die Wahrscheinlichkeit, dass Fehlerzustände im Testobjekt unentdeckt bleiben, aber selbst, wenn keine Fehlerzustände gefunden werden, kann Testen nicht die Korrektheit des Testobjekts beweisen.

2. Vollständiges Testen ist unmöglich. Es ist nicht möglich, alles zu testen, außer in trivialen Fällen (Manna 1978). Anstatt zu versuchen, vollständig zu testen, sollten Testverfahren (siehe

Kapitel 4), Priorisierung von Testfällen (siehe Abschnitt 5.1.5) und risikobasiertes Testen (siehe Abschnitt 5.2) angewendet werden, um den Testaufwand gezielt einzusetzen.

3. Frühes Testen spart Zeit und Geld. Fehlerzustände, die in einem frühen Stadium des Prozesses beseitigt werden, verursachen keine späteren Fehlerzustände in abgeleiteten Arbeitsergebnissen. Die Qualitätskosten werden gesenkt, da später im SDLC weniger Fehlerwirkungen auftreten (Boehm 1981). Um Fehlerzustände frühzeitig zu finden, sollte sowohl mit statischen Tests (siehe Kapitel 3) als auch mit dynamischen Tests (siehe Kapitel 4) so früh wie möglich begonnen werden.

4. Fehlerzustände treten gehäuft auf. Eine kleine Anzahl von Komponenten eines Systems enthält in der Regel die meisten der entdeckten Fehlerzustände oder ist für die meisten Fehlerwirkungen im Betrieb verantwortlich (Enders 1975). Dieses Phänomen ist eine Veranschaulichung des Pareto-Prinzips. Vorausgesagte Anhäufungen von Fehlerzuständen und die tatsächlich beobachteten Fehlerzustände im Test oder im Betrieb sind ein wichtiger Beitrag für den risikobasierten Test (siehe Abschnitt 5.2).

5. Tests nutzen sich ab. Wenn dieselben Tests viele Male wiederholt werden, werden sie bei der Erkennung neuer Fehlerzustände zunehmend ineffektiv (Beizer 1990). Um diesen Effekt zu überwinden, müssen bestehende Tests und Testdaten möglicherweise modifiziert und neue Tests geschrieben werden. In einigen Fällen kann die Wiederholung der gleichen Tests jedoch zu einem positiven Ergebnis führen, z. B. bei automatisierten Regressionstests (siehe Abschnitt 2.2.3).

6. Testen ist kontextabhängig. Es gibt keinen universell anwendbaren Ansatz für das Testen. Das Testen wird in verschiedenen Kontexten unterschiedlich praktiziert (Kaner 2011).

7. Trugschluss: „Keine Fehler“ bedeutet ein brauchbares System. Es ist ein Irrtum (d. h. ein Trugschluss) zu erwarten, dass das Verifizieren von Software den Erfolg eines Systems sicherstellt. Das gründliche Testen aller spezifizierten Anforderungen und das Beheben aller gefundenen Fehlerzustände könnte immer noch ein System hervorbringen, das die Bedürfnisse und Erwartungen der Benutzer nicht erfüllt, das nicht dazu beiträgt, die Geschäftsziele des Kunden zu erreichen, und das im Vergleich zu anderen konkurrierenden Systemen minderwertig ist. Neben der Verifizierung sollte auch eine Validierung durchgeführt werden (Boehm 1981).

1.4 Testaktivitäten, Testmittel und Rollen des Testens

Testen ist kontextabhängig, aber auf einem hohen Abstraktionsniveau gibt es Gruppen von Testaktivitäten, ohne die die Wahrscheinlichkeit, dass die Testziele erreicht werden können, geringer ist. Diese Gruppen von Testaktivitäten bilden einen Testprozess. Der Testprozess kann auf der Grundlage verschiedener Faktoren auf eine bestimmte Situation zugeschnitten werden. Welche Testaktivitäten zu diesem Testprozess gehören, wie sie durchgeführt werden und wann sie stattfinden, wird normalerweise im Rahmen der Testplanung für die jeweilige Situation entschieden (siehe Abschnitt 5.1).

In den folgenden Abschnitten werden die allgemeinen Aspekte dieses Testprozesses in Bezug auf die Testaktivitäten und -aufgaben, der Einfluss des Kontexts, die Testmittel, die Verfolgbarkeit zwischen Testbasis und Testmitteln sowie die Rollen im Testen beschrieben.

Die Norm ISO/IEC/IEEE 29119-2 enthält weitere Informationen über Testprozesse.

1.4.1 Testaktivitäten und -aufgaben

Ein Testprozess besteht in der Regel aus den unten beschriebenen Hauptgruppen von Aktivitäten. Obwohl viele dieser Aktivitäten einer logischen Abfolge zu folgen scheinen, werden sie oft iterativ oder parallel durchgeführt. Diese Testaktivitäten müssen in der Regel auf das System und das Projekt zugeschnitten werden.

Die **Testplanung** besteht darin, die Testziele zu definieren und dann eine Testvorgehensweise auszuwählen, mit der die Testziele innerhalb der durch den Gesamtkontext auferlegten Randbedingungen am besten erreicht werden können. Die Testplanung wird in Abschnitt 5.1 näher erläutert.

Testüberwachung und -steuerung. Die Testüberwachung umfasst die laufende Überprüfung aller Testaktivitäten und den Vergleich des tatsächlichen Fortschritts mit dem Plan. Bei der Teststeuerung werden die erforderlichen Korrekturmaßnahmen ergriffen, um die Testziele zu erreichen. Testüberwachung und -steuerung werden in Abschnitt 5.3 näher erläutert.

Die **Testanalyse** umfasst die Analyse der Testbasis, um testbare Merkmale zu identifizieren und die zugehörigen Testbedingungen zu bestimmen und zu priorisieren, zusammen mit den damit verbundenen Risiken und Risikostufen (siehe Abschnitt 5.2). Die Testbasis und die Testobjekte werden auch geprüft, um darin enthaltene Fehlerzustände zu identifizieren und ihre Testbarkeit zu beurteilen. Die Testanalyse wird häufig durch den Einsatz von Testverfahren unterstützt (siehe Kapitel 4). Die Testanalyse beantwortet die Frage "Was soll getestet werden?" in Form von messbaren Überdeckungskriterien.

Der **Testentwurf** umfasst die Ausarbeitung der Testbedingungen zu Testfällen und anderen Testmitteln (z. B. Test-Chartas). Dabei werden häufig Überdeckungselemente identifiziert, die als Leitfaden für die Spezifizierung der Testfalleingaben dienen. Testverfahren (siehe Kapitel 4) können zur Unterstützung dieser Aktivität eingesetzt werden. Zum Testentwurf gehören auch die Definition von Anforderungen an die Testdaten, der Entwurf der Testumgebung und die Identifizierung von sonstiger benötigter Infrastruktur und Werkzeuge. Der Testentwurf beantwortet die Frage "Wie soll getestet werden?".

Die **Testrealisierung** umfasst die Erstellung oder Beschaffung der für die Testdurchführung erforderlichen Testmittel (z. B. Testdaten). Testfälle können in Testabläufen organisiert werden, die wiederum oft zu Testsuiten zusammengestellt werden. Es werden manuelle und automatisierte Testskripte erstellt. Die Testabläufe werden priorisiert und in einem Testausführungsplan angeordnet, um eine effiziente Testdurchführung zu gewährleisten (siehe Abschnitt 5.1.5). Die Testumgebung wird aufgebaut und ihre korrekte Einrichtung verifiziert.

Die **Testdurchführung** umfasst die Ausführung der Tests gemäß dem Testausführungsplan (Testläufe). Tests können dabei manuell oder automatisiert ausgeführt werden. Die Testdurchführung kann viele Formen annehmen, wie kontinuierlichen Test oder Testsitzungen in Paaren. Die Istergebnisse des Tests werden mit den erwarteten Ergebnissen verglichen. Die Testergebnisse werden protokolliert. Abweichungen zwischen tatsächlichem und erwartetem Ergebnis werden analysiert, um ihre wahrscheinlichen Ursachen zu ermitteln.

Diese Analyse ermöglicht eine Berichterstattung über die Abweichung auf der Grundlage der beobachteten Fehlerwirkungen (siehe Abschnitt 5.5).

Der **Testabschluss** findet in der Regel zu Projektmeilensteinen statt (z. B. Freigabe, Ende der Iteration, Abschluss der Teststufe). Für alle nicht behobenen Fehlerzustände werden Änderungsanträge (Change Requests) oder Produkt-Backlog-Elemente erstellt. Alle Testmittel, die für die Zukunft nützlich sein könnten, werden identifiziert und archiviert oder an die entsprechenden Teams übergeben. Die Testumgebung wird in einen vereinbarten Zustand gebracht. Die Testaktivitäten werden analysiert, um Lessons Learned und Verbesserungen für zukünftige Iterationen, Releases oder Projekte zu ermitteln (siehe Abschnitt 2.1.6). Es wird ein Testabschlussbericht erstellt und an die Stakeholder kommuniziert.

1.4.2 Testprozess im Kontext

Testen wird nicht isoliert durchgeführt, sondern alle Testaktivitäten sind ein integraler Bestandteil der Entwicklungsprozesse innerhalb einer Organisation. Das Testen wird auch von den Stakeholdern finanziert und soll letztendlich dazu beitragen, die Geschäftsanforderungen der Stakeholder zu erfüllen. Daher hängt die Art und Weise, wie das Testen durchgeführt wird, von einer Reihe von Kontextfaktoren ab, darunter:

- Stakeholder (Bedürfnisse, Erwartungen, Anforderungen, Bereitschaft zur Zusammenarbeit, usw.)
- Teammitglieder (Kompetenz, Wissen, Erfahrungsstand, Verfügbarkeit, Schulungsbedarf, usw.)
- Unternehmensbereich (Kritikalität des Testobjekts, identifizierte Risiken, Marktbedürfnisse, spezifische gesetzliche Vorschriften, usw.)
- Technische Faktoren (Art der Software, Produktarchitektur, verwendete Technologie, usw.)
- Projektbedingte Randbedingungen (Umfang, Zeit, Budget, Ressourcen, usw.)
- Organisatorische Faktoren (Organisationsstruktur, bestehende Richtlinien, angewandte Praktiken, usw.)
- Softwareentwicklungslebenszyklus (technologische Praktiken, Entwicklungsmethoden, usw.)
- Werkzeuge (Verfügbarkeit, Gebrauchstauglichkeit, Konformität, usw.)

Diese Faktoren wirken sich auf viele testbezogene Aspekte aus, darunter: Teststrategie, verwendete Testverfahren, Grad der Testautomatisierung, geforderte Überdeckung, Detaillierungsgrad der Testdokumentation, Berichterstattung, usw.

1.4.3 Testmittel

Testmittel werden als Arbeitsergebnisse aus den in Abschnitt 1.4.1 beschriebenen Testaktivitäten erstellt. Es gibt erhebliche Unterschiede in der Art und Weise, wie verschiedene Organisationen ihre Arbeitsergebnisse erstellen, gestalten, benennen, organisieren und

verwalten. Ein ordnungsgemäßes Konfigurationsmanagement (siehe Abschnitt 5.4) gewährleistet die Konsistenz und Integrität der Arbeitsergebnisse. Die folgende Liste der Arbeitsergebnisse erhebt keinen Anspruch auf Vollständigkeit:

- Zu den **Arbeitsergebnissen der Testplanung** gehören: Testkonzept, Testzeitplan, Risikoverzeichnis sowie Eingangs- und Endekriterien (siehe Abschnitt 5.1). Das Risikoverzeichnis ist eine Liste von Risiken mit ihrer jeweiligen Eintrittswahrscheinlichkeit, ihrem Schadensausmaß und Informationen zur Risikominderung (siehe Abschnitt 5.2). Testzeitplan, Risikoverzeichnis sowie Eingangs- und Endekriterien sind häufig Teil des Testkonzepts.
- Zu den **Arbeitsergebnissen der Testüberwachung und -steuerung** gehören: Testfortschrittsberichte (siehe Abschnitt 5.3.2), Dokumentation der Steuerungsmaßnahmen (siehe Abschnitt 5.3) und Risikoinformationen (siehe Abschnitt 5.2).
- Zu den **Arbeitsergebnissen der Testanalyse** gehören: (priorisierte) Testbedingungen (z. B. Abnahmekriterien, siehe Abschnitt 4.5.2) und Fehlerberichte über Fehlerzustände in der Testbasis (falls nicht direkt behoben).
- Zu den **Arbeitsergebnissen des Testentwurfs** gehören: (priorisierte) Testfälle, Test-Chartas, Überdeckungselemente, Anforderungen an Testdaten und an Testumgebungen.
- Zu den **Arbeitsergebnissen der Testrealisierung** gehören: Testabläufe, automatisierte Testskripte, Testsuiten, Testdaten, Testausführungspläne und Bestandteile der Testumgebung. Beispiele für Bestandteile der Testumgebung sind: Platzhalter, Treiber, Simulatoren und Dienst-Virtualisierungen (service virtualizations).
- Zu den **Arbeitsergebnissen der Testdurchführung** gehören: Testprotokolle und Fehlerberichte (siehe Abschnitt 5.5).
- Zu den **Arbeitsergebnissen des Testabschlusses** gehören: Testabschlussberichte (siehe Abschnitt 5.3.2), Maßnahmen zur Verbesserung nachfolgender Projekte oder Iterationen, dokumentierte Lessons Learned und Änderungsanträge (z. B. als Elemente des Produkt-Backlogs).

1.4.4 Verfolgbarkeit zwischen der Testbasis und den Testmitteln

Für eine effektive Testüberwachung und -steuerung ist es wichtig, während des gesamten Testprozesses eine Verfolgbarkeit zwischen den Bestandteilen der Testbasis, den mit diesen Bestandteilen verbundenen Testmitteln (z. B. Testbedingungen, Risiken, Testfälle), den Testergebnissen und den festgestellten Fehlerzuständen herzustellen und zu pflegen.

Eine genaue Verfolgbarkeit unterstützt die Bewertung der Überdeckung, daher ist es sehr nützlich, wenn in der Testbasis messbare Überdeckungskriterien definiert sind. Die Überdeckungskriterien können als wichtige Key-Performance-Indicator (KPI) dienen, um die Aktivitäten zu steuern, die zeigen, inwieweit die Testziele erreicht wurden (siehe Abschnitt 1.1.1). Zum Beispiel:

- Durch die Verfolgbarkeit von Testfällen zu Anforderungen kann überprüft werden, ob die Anforderungen durch Testfälle überdeckt werden.
- Durch die Verfolgbarkeit von Testergebnissen zu Risiken kann das Ausmaß des Restrisikos eines Testobjekts bewertet werden.

Neben der Bewertung der Überdeckung ermöglicht eine gute Verfolgbarkeit die Ermittlung der Auswirkungen von Änderungen, erleichtert Audits der Testaktivitäten und hilft bei der Erfüllung von IT-Governance-Kriterien. Eine gute Verfolgbarkeit macht auch Testfortschritts- und Testabschlussberichte leichter verständlich, indem sie den Status der Bestandteile der Testbasis enthält. Dies kann auch dabei helfen, den Stakeholdern die technischen Aspekte des Testens auf verständliche Weise zu vermitteln. Die Verfolgbarkeit liefert Informationen zur Bewertung der Produktqualität, der Prozessfähigkeit und des Projektfortschritts im Vergleich zu den Unternehmenszielen.

1.4.5 Rollen des Testens

In diesem Lehrplan werden zwei Hauptrollen des Testens behandelt: eine Rolle des Testmanagements und eine Rolle des Testens. Die Aktivitäten und Aufgaben, die diesen beiden Rollen zugewiesen werden, hängen von Faktoren wie dem Projekt- und Produktkontext, den Kompetenzen der Personen, die diese Rollen innehaben, und der Organisation ab.

Die Rolle des Testmanagements übernimmt die Gesamtverantwortung für den Testprozess, das Testteam und die Leitung der Testaktivitäten. Die Rolle des Testmanagements konzentriert sich hauptsächlich auf die Aktivitäten der Testplanung, Testüberwachung und -steuerung sowie des Testabschlusses. Die Art und Weise, wie die Rolle des Testmanagements ausgeübt wird, variiert je nach Kontext. Bei der agilen Softwareentwicklung beispielsweise können einige der Aufgaben des Testmanagements vom agilen Team übernommen werden. Aufgaben, die sich über mehrere Teams oder die gesamte Organisation erstrecken, können von Testmanagern außerhalb des Entwicklungsteams übernommen werden.

Die Rolle des Testens übernimmt die Gesamtverantwortung für den operativen Aspekt des Testens. Die Rolle des Testens konzentriert sich hauptsächlich auf die Aktivitäten der Testanalyse, des Testentwurfs, der Testrealisierung und der Testdurchführung.

Diese Rollen können von verschiedenen Personen zu verschiedenen Zeiten übernommen werden. Die Rolle des Testmanagements kann zum Beispiel von einem Teamleiter, einem Testmanager, einem Entwicklungsleiter usw. übernommen werden. Es ist auch möglich, dass eine Person gleichzeitig die Rollen des Testens und des Testmanagements übernimmt.

1.5 Wesentliche Kompetenzen und bewährte Praktiken beim Testen

Kompetenz ist die Fähigkeit, etwas gut zu machen, die sich aus dem Wissen, der Übung und der Eignung einer Person ergibt. Gute Tester sollten über einige wesentliche Kompetenzen verfügen, um ihre Arbeit gut zu machen. Gute Tester sollten effektive Teamplayer sein und in der Lage sein, Tests mit verschiedenen Graden an Unabhängigkeit durchzuführen.

1.5.1 Allgemeine Kompetenzen, die für das Testen erforderlich sind

Die folgenden Kompetenzen sind zwar überwiegend allgemeiner Art, aber für Tester besonders wichtig:

- Testwissen (zur Steigerung der Effektivität des Testens, z. B. durch den Einsatz von Testverfahren)
- Gründlichkeit, Sorgfalt, Neugier, Detailgenauigkeit, methodisches Vorgehen (um Fehlerzustände zu erkennen, insbesondere solche, die schwer zu finden sind)
- Gute Kommunikationsfähigkeit, aktives Zuhören, Teamfähigkeit (um mit allen Stakeholdern effektiv zu interagieren, Informationen an andere weiterzugeben, verstanden zu werden und Fehlerzustände zu berichten und zu diskutieren)
- Analytisches Denken, kritisches Denken, Kreativität (zur Steigerung der Effektivität des Testens)
- Technische Kenntnisse (um die Effizienz des Testens zu steigern, z. B. durch den Einsatz geeigneter Testwerkzeuge)
- Wissen in der Anwendungsdomäne (um Endanwender/Fachbereichsvertreter verstehen und mit ihnen kommunizieren zu können)

Tester sind oft die Überbringer schlechter Nachrichten. Es ist ein allgemeiner menschlicher Charakterzug, den Überbringer schlechter Nachrichten zu verurteilen. Daher ist Kommunikationsfähigkeit für Tester von entscheidender Bedeutung. Die Kommunikation von Testergebnissen kann als Kritik an dem Produkt und seinem Autor aufgefasst werden. Bestätigungsfehler (Voreingenommenheit) können dazu führen, dass es schwierig ist, Informationen zu akzeptieren, die nicht mit den bereits bestehenden Überzeugungen übereinstimmen. Manche Menschen empfinden das Testen als eine destruktive Aktivität, obwohl es in hohem Maße zum Projekterfolg und zur Qualität des Produkts beiträgt. Um diese Sichtweise zu verbessern, sollten Informationen über Fehlerzustände und Fehlerwirkungen auf konstruktive Weise kommuniziert werden.

1.5.2 Whole-Team-Ansatz (Whole Team Approach)

Eine der wichtigsten Kompetenzen eines Testers ist die Fähigkeit, effektiv im Team zu arbeiten und einen positiven Beitrag zu den Teamzielen zu leisten. Der Whole-Team-Ansatz – eine aus dem Extreme Programming stammende Praxis (siehe Abschnitt 2.1) – baut auf dieser Fähigkeit auf.

Beim Whole-Team-Ansatz kann jedes Teammitglied, das über die erforderlichen Kompetenzen verfügt, jede Aufgabe ausführen, und jeder ist für die Qualität verantwortlich. Die Teammitglieder teilen sich einen gemeinsamen Arbeitsbereich (physisch oder virtuell), da der gemeinsame Standort die Kommunikation und Interaktion erleichtert. Der Whole-Team-Ansatz verbessert die Teamdynamik, fördert die Kommunikation und Zusammenarbeit innerhalb des Teams und schafft Synergien, da die verschiedenen Kompetenzen innerhalb des Teams zum Nutzen des Projekts eingesetzt werden.

Tester arbeiten eng mit anderen Teammitgliedern zusammen, um sicherzustellen, dass die gewünschte Qualität erreicht wird. Dazu gehört die Zusammenarbeit mit Fachbereichsvertretern, um sie bei der Erstellung geeigneter Abnahmetests zu unterstützen, und die Zusammenarbeit mit Entwicklern, um die Teststrategie abzustimmen und über Ansätze der Testautomatisierung zu entscheiden. So können Tester ihr Wissen über das Testen an andere Teammitglieder weitergeben und die Entwicklung des Produkts positiv beeinflussen.

Je nach Kontext ist der Whole-Team-Ansatz nicht immer angemessen. In einigen Situationen, wie z. B. in sicherheitskritischen Bereichen, kann ein hohes Maß an Unabhängigkeit des Testens erforderlich sein.

1.5.3 Unabhängigkeit des Testens

Ein gewisser Grad an Unabhängigkeit macht den Tester effektiver bei der Fehlerfindung, da sich die Voreingenommenheit (kognitive Verzerrungen) zwischen Autor und Tester unterscheidet (vgl. Salman 1995). Unabhängigkeit ist jedoch kein Ersatz für Nähe zum System, z. B. können Entwickler viele Fehlerzustände in ihrem eigenen Code effizient finden.

Arbeitsergebnisse können von ihrem Autor (keine Unabhängigkeit), von den Kollegen des Autors aus demselben Team (etwas Unabhängigkeit), von Testern außerhalb des Teams des Autors, aber innerhalb der Organisation (hohe Unabhängigkeit), oder von Testern außerhalb der Organisation (sehr hohe Unabhängigkeit) getestet werden. Bei den meisten Projekten ist es in der Regel am besten, das Testen mit mehreren Unabhängigkeitsstufen durchzuführen (z. B. Entwickler, die Komponenten- und Komponentenintegrationstests durchführen, ein Testteam, das System- und Systemintegrationstests durchführt, und Fachbereichsvertreter, die Abnahmetests durchführen).

Der Hauptvorteil des unabhängigen Testens besteht darin, dass unabhängige Tester wahrscheinlich andere Arten von Fehlerwirkungen und Fehlerzuständen erkennen als Entwickler, aufgrund ihres unterschiedlichen Hintergrunds, ihrer technischen Perspektive und der Voreingenommenheit der Entwickler. Außerdem kann ein unabhängiger Tester die Annahmen, die von den Stakeholdern während der Spezifikation und Implementierung des Systems gemacht wurden, überprüfen, in Frage stellen oder widerlegen.

Allerdings gibt es auch einige Nachteile. Unabhängige Tester können vom Entwicklungsteam isoliert sein, was zu mangelnder Zusammenarbeit, Kommunikationsproblemen oder einer gegnerischen Beziehung mit dem Entwicklungsteam führen kann. Die Entwickler verlieren möglicherweise das Gefühl der Verantwortung für die Qualität. Unabhängige Tester können als Engpass angesehen oder für Verzögerungen bei der Freigabe verantwortlich gemacht werden.

2. Testen während des Softwareentwicklungslebenszyklus – 130 Minuten

Schlüsselbegriffe

Abnahmetest, Black-Box-Test, Fehlernachtest, funktionaler Test, Integrationstest, Komponententest, Komponentenintegrationstest, nicht-funktionaler Test, Regressionstest, Shift-Left, Systemintegrationstest, Systemtest, Testart, Testobjekt, Teststufe, Wartungstest, White-Box-Test

Lernziele für Kapitel 2: Der Lernende kann ...

2.1 Testen im Kontext eines Softwareentwicklungslebenszyklus

FL-2.1.1 (K2) ... die Auswirkungen des gewählten Softwareentwicklungslebenszyklus auf das Testen erklären

FL-2.1.2 (K1) ... gute Praktiken für das Testen, die für alle Softwareentwicklungslebenszyklen gelten, wiedergeben

FL-2.1.3 (K1) ... die Beispiele für Test-First-Ansätze in der Entwicklung wiedergeben

FL-2.1.4 (K2) ... die möglichen Auswirkungen von DevOps auf das Testen zusammenfassen

FL-2.1.5 (K2) ... den Shift-Left-Ansatz erklären

FL-2.1.6 (K2) ... den Einsatz von Retrospektiven als Mechanismus zur Prozessverbesserung erklären

2.2 Teststufen und Testarten

FL-2.2.1 (K2) ... die verschiedenen Teststufen unterscheiden

FL-2.2.2 (K2) ... die verschiedenen Testarten unterscheiden

FL-2.2.3 (K2) ... Fehlernachtests von Regressionstests unterscheiden

2.3 Wartungstest

FL-2.3.1 (K2) ... Wartungstest und dessen Auslöser zusammenfassen

2.1 Testen im Kontext eines Softwareentwicklungslebenszyklus

Ein Modell des Softwareentwicklungslebenszyklus (SDLC) ist eine abstrakte, übergeordnete Darstellung des Softwareentwicklungsprozesses. Ein SDLC-Modell definiert, wie innerhalb des Prozesses verschiedene Entwicklungsphasen und Arten von Aktivitäten logisch und chronologisch zueinander in Beziehung stehen. Beispiele für SDLC-Modelle sind: sequenzielle Entwicklungsmodelle (z. B. Wasserfallmodell, V-Modell), iterative Entwicklungsmodelle (z. B. Spiralmodell, Prototyping) und inkrementelle Entwicklungsmodelle (z. B. Unified Process).

Einige Aktivitäten innerhalb von Softwareentwicklungsprozessen können auch durch detaillier-tere Softwareentwicklungsmethoden und agile Praktiken beschrieben werden. Beispiele hierfür sind: abnahmetestgetriebene Entwicklung (acceptance test-driven development, ATDD), verhaltensgetriebene Entwicklung (behavior-driven development BDD), domänengesteuertes Design (domain-driven design, DDD), Extreme Programming (XP), Feature-getriebene Entwicklung (feature-driven development, FDD), Kanban, Lean IT, Scrum und testgetriebene Entwicklung (test-driven development, TDD).

2.1.1 Auswirkungen des Softwareentwicklungslebenszyklus auf das Testen

Testen muss an den SDLC angepasst werden, um erfolgreich zu sein. Die Auswahl des SDLC hat Auswirkungen auf:

- Umfang und Zeitpunkt der Testaktivitäten (z. B. Teststufen und Testarten)
- Detaillierungsgrad der Testdokumentation
- Wahl der Testverfahren und des Testansatzes
- Umfang der Testautomatisierung
- Rolle und Aufgaben eines Testers

In sequenziellen Entwicklungsmodellen sind Tester in den Anfangsphasen in der Regel an den Reviews der Anforderungen, der Testanalyse und dem Testentwurf beteiligt. Der ausführbare Code wird normalerweise in den späteren Phasen erstellt, so dass dynamische Tests nicht in den frühen Phasen des SDLC durchgeführt werden können.

Bei einigen iterativen und inkrementellen Entwicklungsmodellen wird davon ausgegangen, dass jede Iteration einen funktionierenden Prototyp oder ein Inkrement des Produkts liefert. Dies impliziert, dass in jeder Iteration sowohl statische als auch dynamische Tests auf allen Teststufen durchgeführt werden können. Die häufige Lieferung von Inkrementen erfordert eine schnelle Rückmeldung und umfangreiche Regressionstests.

Bei der agilen Softwareentwicklung wird davon ausgegangen, dass sich während des gesamten Projekts Änderungen ergeben können. Daher werden in agilen Projekten eine schlanke Dokumentation der Arbeitsergebnisse und eine umfassende Testautomatisierung bevorzugt, um Regressionstests zu erleichtern. Außerdem kann der Großteil der manuellen Tests mit erfahrungsbasierten Testverfahren durchgeführt werden (siehe Abschnitt 4.4), die keine umfangreiche Testanalyse und keinen ausführlichen Testentwurf erfordern.

2.1.2 Softwareentwicklungslebenszyklus und gute Praktiken für das Testen

Zu den guten Testpraktiken gehören, unabhängig vom gewählten SDLC-Modell, die Folgenden:

- Für jede Softwareentwicklungsaktivität gibt es eine entsprechende Testaktivität, so dass alle Entwicklungsaktivitäten der Qualitätssteuerung unterliegen.
- Unterschiedliche Teststufen (siehe Kapitel 2.2.1) haben spezifische und unterschiedliche Testziele, so dass der jeweilige Test angemessen und entsprechend umfassend ist und Redundanzen vermieden werden.
- Die Testanalyse und der Testentwurf für eine bestimmte Teststufe beginnen bereits in der entsprechenden Entwicklungsphase des SDLC, so dass der Test den Grundsatz des frühen Testens (siehe Abschnitt 1.3) einhalten kann.
- Tester werden in das Review von Arbeitsergebnissen einbezogen, sobald Entwürfe dieser Dokumentation verfügbar sind, so dass frühes Testen und die Fehlerentdeckung den Shift-Left-Ansatz unterstützen können (siehe Abschnitt 2.1.5).

2.1.3 Testen als Treiber für die Softwareentwicklung

TDD, ATDD und BDD sind ähnliche Entwicklungsansätze, bei denen Tests als Mittel zur Lenkung der Entwicklung definiert werden. Jeder dieser Ansätze setzt das Prinzip des frühen Testens um (siehe Abschnitt 1.3) und folgt einem Shift-Left-Ansatz (siehe Abschnitt 2.1.5), da die Tests definiert werden, bevor der Code geschrieben wird. Sie unterstützen ein iteratives Entwicklungsmodell. Diese Ansätze werden wie folgt charakterisiert:

Testgetriebene Entwicklung (TDD):

- Lenkt die Codierung durch Testfälle (unter Verzicht auf einen umfangreichen Softwareentwurf) (Beck 2003).
- Zuerst werden Tests geschrieben, dann wird der Code geschrieben, um die Tests zu erfüllen, und dann werden Tests und Code überarbeitet (Refactoring).

Abnahmetestgetriebene Entwicklung (ATDD) (siehe Abschnitt 4.5.3):

- Leitet Tests aus Abnahmekriterien als Teil des Systementwurfs ab (Gärtner 2011).
- Tests werden geschrieben, bevor der Teil der Anwendung entwickelt wird, der die Tests erfüllt.

Verhaltensgetriebene Entwicklung (Behavior-Driven Development, BDD):

- Drückt das gewünschte Verhalten einer Anwendung mit Testfällen aus, die in einer einfachen, natürlichsprachlichen Form geschrieben und die von Stakeholdern leicht zu verstehen sind – üblicherweise unter Verwendung des Gegeben/Wenn/Dann- (Given/When/Then-) Formats (Chelimsky 2010).
- Die Testfälle werden dann automatisch in ausführbare Tests übersetzt.

Um die Codequalität bei zukünftigen Anpassungen / Umgestaltungen (Refactoring) sicherzustellen, können bei allen oben genannten Ansätzen die Tests als automatisierte Tests weiterverwendet werden.

2.1.4 DevOps und Testen

DevOps ist ein organisatorischer Ansatz, der darauf abzielt, Synergien zu schaffen, indem Entwicklung (einschließlich Testen) und Betrieb zusammenarbeiten, um eine Reihe von gemeinsamen Zielen zu erreichen. DevOps erfordert einen Kulturwandel innerhalb eines Unternehmens, um die Kluft zwischen Entwicklung (einschließlich Testen) und Betrieb zu überbrücken und gleichzeitig ihre jeweilige Aufgabe gleichwertig zu behandeln. DevOps fördert Teamautonomie, schnelle Rückmeldungen, integrierte Werkzeugketten und technische Praktiken wie kontinuierliche Integration (Continuous Integration, CI) und kontinuierliche Auslieferung (Continuous Delivery, CD). Dies ermöglicht es den Teams, qualitativ hochwertigen Code über eine DevOps-Auslieferungskette (Delivery Pipeline) schneller zu erstellen, zu testen und freizugeben (Kim 2016).

Aus Sicht des Testens sind einige der Vorteile von DevOps:

- Schnelle Rückmeldung über die Codequalität und ob sich Änderungen nachteilig auf den bestehenden Code auswirken.
- CI fördert einen Shift-Left-Ansatz beim Testen (siehe Abschnitt 2.1.5), indem Entwickler dazu angehalten werden, qualitativ hochwertigen Code zusammen mit Komponententests und statischer Analyse bereitzustellen.
- Förderung von automatisierten Prozessen wie CI/CD, die den Aufbau stabiler Testumgebungen erleichtern.
- Steigert den Blick auf nicht-funktionale Qualitätsmerkmale (z. B. Performanz, Zuverlässigkeit).
- Automatisierung durch eine Auslieferungskette reduziert den Bedarf an sich wiederholenden manuellen Tests.
- Das Risiko einer zu aufwendigen Regression wird durch den Umfang und die Bandbreite der automatisierten Regressionstests minimiert.

DevOps ist nicht ohne Risiken und Herausforderungen, dazu gehören:

- Die DevOps-Auslieferungskette muss definiert und etabliert werden.
- CI/CD-Werkzeuge müssen eingeführt und gewartet werden.
- Die Testautomatisierung erfordert zusätzliche Ressourcen und kann schwierig einzurichten und zu warten sein.

Obwohl DevOps ein hohes Maß an automatisierten Tests mit sich bringt, sind manuelle Tests – insbesondere aus Benutzerperspektive – weiterhin erforderlich.

2.1.5 Shift-Left-Ansatz

Das Prinzip des frühen Testens (siehe Abschnitt 1.3) wird manchmal auch als Shift-Left bezeichnet, da es sich um einen Ansatz handelt, bei dem Testen zu einem früheren Zeitpunkt im SDLC durchgeführt wird. Shift-Left bedeutet normalerweise, dass das Testen früher beginnen sollte z. B. nicht erst, wenn der Code implementiert oder die Komponenten integriert sind, aber nicht, dass das Testen später im SDLC vernachlässigt werden sollte.

Es gibt einige bewährte Verfahren, die veranschaulichen, wie ein "Shift-Left" beim Testen erreicht werden kann, dazu gehören:

- Review der Spezifikation aus der Sicht des Testens. Diese Review-Aktivitäten zu Spezifikationen finden oft potenzielle Fehlerzustände, wie Mehrdeutigkeiten, Unvollständigkeit und Inkonsistenzen.
- Schreiben von Testfällen, bevor der Code geschrieben wird, und Ausführen des Codes in einem Testrahmen während der Coderealisierung.
- Verwendung von CI und noch besser auch CD, da dies schnelle Rückmeldungen und automatisierte Tests für Komponenten bietet, die zusammen mit dem Quellcode in das Code-Repository eingefügt wird.
- Abschluss der statischen Analyse des Quellcodes vor dem dynamischen Testen oder als Teil eines automatisierten Prozesses.
- Durchführung von nicht-funktionalen Tests, wenn möglich, beginnend auf der Ebene der Komponententests. Dies ist eine Form von Shift-Left, da die nicht-funktionalen Testarten meist erst spät im SDLC durchgeführt werden, wenn ein vollständiges System und eine repräsentative Testumgebung zur Verfügung stehen.

Ein Shift-Left-Ansatz kann zu Beginn des Prozesses zu zusätzlichen Schulungen, Aufwand und/oder Kosten führen. Jedoch gilt die Erwartung später im Prozess Aufwand und/oder Kosten sparen.

Für den Shift-Left-Ansatz ist es wichtig, dass die Stakeholder von dem Konzept überzeugt sind und es annehmen.

2.1.6 Retrospektiven und Prozessverbesserung

Retrospektiven (auch bekannt als Projekt-Abschluss-Sitzungen oder Bewertungssitzungen und Projekt-Retrospektiven) werden häufig am Ende eines Projekts oder einer Iteration, bei einem Releasemeilenstein oder bei Bedarf abgehalten. Zeitpunkt und Organisation der Retrospektiven hängen von dem jeweiligen SDLC-Modell ab. In diesen Sitzungen diskutieren die Teilnehmer (nicht nur Tester, sondern z. B. auch Entwickler, Architekten, Product Owner, Businessanalysten):

- Was war erfolgreich und sollte beibehalten werden?
- Was war nicht erfolgreich und könnte verbessert werden?
- Wie können die Verbesserungen eingearbeitet und die Erfolge in Zukunft beibehalten werden?

Die Ergebnisse sollten festgehalten werden und sind normalerweise Teil des Testabschlussberichts (siehe Abschnitt 5.3.2). Retrospektiven sind entscheidend für die erfolgreiche Umsetzung der kontinuierlichen Verbesserung, und es ist wichtig, dass die empfohlenen Verbesserungen weiterverfolgt werden.

Typische Vorteile für das Testen sind:

- Erhöhte Effektivität/Effizienz des Testens, z. B. durch die Umsetzung von Vorschlägen zur Prozessverbesserung
- Höhere Qualität der Testmittel, z. B. durch gemeinsames Review der Testprozesse
- Teamzusammenhalt und Lernen, z. B. durch die Möglichkeit, Probleme anzusprechen und Verbesserungspunkte vorzuschlagen
- Verbesserte Qualität der Testbasis, z. B. weil Mängel im Umfang und in der Qualität der Anforderungen angesprochen und behoben werden konnten
- Bessere Zusammenarbeit zwischen Entwicklung und Test, z. B. weil die Zusammenarbeit regelmäßig überprüft und optimiert wird

2.2 Teststufen und Testarten

Teststufen sind Gruppen von Testaktivitäten, die gemeinsam organisiert und verwaltet werden. Jede Teststufe ist eine Instanz des Testprozesses, die in Bezug auf Software in einem bestimmten Entwicklungsstadium durchgeführt wird, von einzelnen Komponenten bis hin zu kompletten Systemen oder gegebenenfalls Systemen von Systemen.

Teststufen stehen in Beziehung zu anderen Aktivitäten innerhalb des SDLC. In sequenziellen SDLC-Modellen sind die Teststufen oft so definiert, dass die Endekriterien einer Stufe Teil der Eingangskriterien für die nächste Stufe sind. In einigen iterativen Modellen trifft dies nicht zu. Entwicklungsaktivitäten können sich über mehrere Teststufen erstrecken. Teststufen können sich zeitlich überschneiden.

Testarten sind Gruppen von Testaktivitäten, die sich auf bestimmte Qualitätsmerkmale beziehen, und die meisten dieser Testaktivitäten können in jeder Teststufe durchgeführt werden.

2.2.1 Teststufen

In diesem Lehrplan werden die folgenden fünf Teststufen beschrieben:

- **Der Komponententest** (auch Unittest genannt) konzentriert sich auf das Testen von isolierten Komponenten. Dies erfordert oft spezifische Unterstützung, wie Testrahmen oder Unittest-Frameworks. Komponententests werden normalerweise von Entwicklern in ihrer Entwicklungsumgebung durchgeführt.
- **Der Komponentenintegrationstest** konzentriert sich auf das Testen der Schnittstellen und Interaktionen zwischen Komponenten.

Komponentenintegrationstests sind stark abhängig von der Integrationsstrategie, wie Bottom-up, Top-down oder Big-Bang.

- **Der Systemtest** konzentriert sich auf das Gesamtverhalten und die Leistungsfähigkeiten eines gesamten Systems oder Produkts und umfassen häufig funktionale Tests von End-To-End-Aufgaben und nicht-funktionale Tests von Qualitätsmerkmalen. Bei einigen nicht-funktionalen Qualitätsmerkmalen ist es besser, sie an einem vollständigen System in einer repräsentativen Testumgebung zu testen (z. B. Gebrauchstauglichkeit). Die Verwendung von Simulationen von Teilsystemen ist ebenfalls möglich. Der Systemtest kann von einem unabhängigen Testteam durchgeführt werden und bezieht sich auf Anforderungsspezifikationen für das System.
- **Der Systemintegrationstest** konzentriert sich auf das Testen der Schnittstellen zwischen dem System unter Test und anderen Systemen und externen Diensten. Für Systemintegrationstests sind geeignete Testumgebungen erforderlich, die vorzugsweise der Betriebsumgebung entsprechen.
- **Der Abnahmetest** konzentriert sich auf die Validierung und den Nachweis der Einsatzfähigkeit, d. h., dass das System die Geschäftsanforderungen des Benutzers erfüllt. Idealerweise sollten Abnahmetests von den vorgesehenen Benutzern durchgeführt werden. Die wichtigsten Formen des Abnahmetests sind: der Benutzerabnahmetest (user acceptance testing, UAT), der betriebliche Abnahmetest, der vertragliche und regulatorische Abnahmetest, der Alpha-Test und der Beta-Test.

Die Teststufen werden in diesem Kapitel durch die folgende Liste von Attributen unterschieden, um Überschneidungen von Testaktivitäten zu vermeiden:

- Testobjekt
- Testziele
- Testbasis
- Fehlerzustände und Fehlerwirkungen
- Vorgehensweise und Verantwortlichkeiten

2.2.2 Testarten

Es gibt eine Vielzahl von Testarten, die in Projekten eingesetzt werden können. In diesem Lehrplan werden die folgenden vier Testarten behandelt:

Beim **funktionalen Test** werden die Funktionen bewertet, die eine Komponente oder ein System erfüllen soll. Die Funktionen sind, „was" das Testobjekt tun soll. Das Hauptziel der funktionalen Tests ist die Überprüfung der funktionalen Vollständigkeit, der funktionalen Korrektheit und der funktionalen Angemessenheit.

Beim **nicht-funktionalen Test** werden andere als die funktionalen Eigenschaften einer Komponente oder eines Systems bewertet. Beim nicht-funktionalen Test wird geprüft, "wie gut sich das System verhält". Das Hauptziel des nicht-funktionalen Tests ist die Überprüfung der nicht-funktionalen Qualitätsmerkmale der Software. Die Norm ISO/IEC 25010 bietet die folgende Klassifizierung der nicht-funktionalen Qualitätsmerkmale von Software:

- Performanz
- Kompatibilität
- Gebrauchstauglichkeit
- Zuverlässigkeit
- IT-Sicherheit
- Wartbarkeit
- Übertragbarkeit

Manchmal ist es sinnvoll, dass nicht-funktionale Tests schon früh im Lebenszyklus beginnen (z. B. im Rahmen von Reviews und Komponententest oder Systemtest). Viele nicht-funktionale Tests leiten sich von funktionalen Tests ab, da sie dieselben funktionalen Tests verwenden, aber prüfen, ob bei der Ausführung der Funktion eine nicht-funktionale Bedingung erfüllt ist (z. B. die Prüfung, ob eine Funktion innerhalb einer bestimmten Zeit ausgeführt wird oder ob eine Funktion auf eine neue Plattform portiert werden kann). Die späte Entdeckung von nicht-funktionalen Fehlerzuständen kann den Erfolg eines Projekts ernsthaft gefährden. Nicht-funktionale Tests erfordern manchmal eine sehr spezielle Testumgebung, wie z. B. ein Gebrauchstauglichkeitslabor für Gebrauchstauglichkeitstests.

Der **Black-Box-Test** (siehe Abschnitt 4.2) basiert auf Spezifikationen und leitet die Tests aus der Dokumentation außerhalb des Testobjekts ab. Das Hauptziel des Black-Box-Tests besteht darin, das Verhalten des Systems gegen seine Spezifikationen zu überprüfen.

Der **White-Box-Test** (siehe Abschnitt 4.3) ist strukturbasiert und leitet Tests aus der Implementierung oder der internen Struktur des Systems ab (z. B. Code, Architektur, Arbeitsabläufe und Datenflüsse). Das Hauptziel des White-Box-Tests besteht darin, die zugrunde liegende Struktur durch die Tests bis zu einer akzeptablen Stufe zu überdecken.

Alle vier oben genannten Testarten können auf allen Teststufen angewandt werden, auch wenn der Schwerpunkt auf jeder Stufe anders ist. Für alle genannten Testarten können unterschiedliche Testverfahren zur Ableitung von Testbedingungen und Testfällen verwendet werden.

2.2.3 Fehlernachtest und Regressionstest

Änderungen werden in der Regel an einer Komponente oder einem System vorgenommen, um entweder durch Hinzufügen eines neuen Features eine Verbesserung oder durch Beseitigung eines Fehlerzustands eine Korrektur zu erreichen. Das Testen sollte dann auch Fehlernachtests und Regressionstests beinhalten.

Der Fehlernachtest bestätigt, dass ein ursprünglicher Fehlerzustand erfolgreich behoben wurde. Je nach Risiko kann man die behobene Version der Software auf verschiedene Arten testen, z. B:

- Ausführen aller Testfälle, die zuvor aufgrund des Fehlerzustands fehlgeschlagen sind, oder auch durch

- Hinzufügen neuer Testfälle, um alle Änderungen zu überdecken, die zur Behebung des Fehlerzustands erforderlich waren.

Wenn jedoch die Zeit oder das Geld für die Behebung von Fehlern knapp ist, können sich Fehlernachtests darauf beschränken, lediglich die Testschritte auszuführen, die die durch den Fehlerzustand verursachte Fehlerwirkung produziert haben, und zu prüfen, ob die Fehlerwirkung nicht mehr auftritt.

Der Regressionstest bestätigt, dass eine Änderung, einschließlich einer bereits getesteten Fehlerbehebung, keine nachteiligen Folgen hat. Die nachteiligen Folgen könnten die Komponente betreffen, an der die Änderung vorgenommen wurde, andere Komponenten desselben Systems oder sogar andere verbundene Systeme. Der Regressionstest muss sich nicht auf das Testobjekt selbst beschränken, sondern kann sich auch auf die Umgebung beziehen. Es ist ratsam, zunächst eine Auswirkungsanalyse durchzuführen, um den Umfang der Regressionstests zu optimieren. Die Auswirkungsanalyse zeigt, welche Teile der Software betroffen sein könnten.

Regressionstestsuiten werden viele Male durchlaufen, und im Allgemeinen nimmt die Anzahl der Testfälle mit jeder Iteration oder jedem Release zu, so dass sich Regressionstests sehr gut für eine Automatisierung eignen. Die Automatisierung dieser Tests sollte bereits in einem frühen Stadium des Projekts beginnen. Wenn CI eingesetzt wird, wie z. B. bei DevOps (siehe Abschnitt 2.1.4), ist es gute Praxis, auch automatisierte Regressionstests einzubeziehen. Je nach Situation kann dies Regressionstests auf verschiedenen Teststufen umfassen.

Fehlernachtests und/oder Regressionstests für das Testobjekt sind auf allen Teststufen erforderlich, wenn Fehlerzustände behoben und/oder Änderungen für diese Teststufen vorgenommen wurden.

2.3 Wartungstest

Es gibt verschiedene Kategorien von Wartung, sie kann korrigierend sein, sich an Änderungen in der Umgebung anpassen oder die Leistung oder Wartbarkeit verbessern (Einzelheiten siehe ISO/IEC 14764), so dass die Wartung geplante Releases/Bereitstellungen und ungeplante Releases/Bereitstellungen (Hotfixes) umfassen kann. Vor einer Änderung kann eine Auswirkungsanalyse durchgeführt werden, um auf der Grundlage der potenziellen Auswirkungen auf andere Bereiche des Systems zu entscheiden, ob die Änderung durchgeführt werden sollte. Das Testen der Änderungen an einem System in Produktion umfasst sowohl die Bewertung des Erfolgs der Implementierung der Änderung als auch die Überprüfung auf mögliche nachteilige Folgen (Regressionstest) in Teilen des Systems, die unverändert bleiben (was in der Regel der größte Teil des Systems ist).

Der Umfang des Wartungstests hängt in der Regel ab von:

- dem Grad des Risikos der Änderung
- der Größe des bestehenden Systems
- dem Umfang der Änderung

Die Auslöser für Wartung und Wartungstest können wie folgt klassifiziert werden:

- Änderungen, wie z. B. geplante Erweiterungen (d. h. release-basiert), korrigierende Änderungen oder Hotfixes.
- Upgrades oder Migrationen der Betriebsumgebung, z. B. von einer Plattform auf eine andere, was Tests der neuen Umgebung sowie der geänderten Software erfordern kann, oder Tests der Datenkonvertierung, wenn Daten aus einer anderen Anwendung in das zu wartende System migriert werden.
- Außerbetriebnahme, z. B. wenn eine Anwendung das Ende ihres Lebens erreicht. Wenn ein System außer Betrieb genommen wird, kann dies Tests der Datenarchivierung erfordern, falls lange Datenaufbewahrungsfristen erforderlich sind. Das Testen von Wiederherstellungsverfahren nach der Archivierung kann ebenfalls erforderlich sein, wenn bestimmte Daten während der Archivierungszeit benötigt werden.

3. Statischer Test – 80 Minuten

Schlüsselbegriffe

Anomalie, dynamischer Test, formales Review, informelles Review, Inspektion, Review, statische Analyse, statischer Test, Technisches Review, Walkthrough

Lernziele für Kapitel 3: Der Lernende kann ...

3.1 Grundlagen des statischen Tests

FL-3.1.1 (K1) ... Produktarten, die mit den verschiedenen statischen Testverfahren geprüft werden können, erkennen

FL-3.1.2 (K2) ... den Wert statischer Tests erklären

FL-3.1.3 (K2) ... statischen und dynamischen Test vergleichen und gegenüberstellen

3.2 Feedback- und Reviewprozess

FL-3.2.1 (K1) ... Vorteile eines frühzeitigen und häufigen Stakeholder-Feedbacks erkennen

FL-3.2.2 (K2) ... die Aktivitäten des Reviewprozesses zusammenfassen

FL-3.2.3 (K1) ... die bei der Durchführung von Reviews den Hauptrollen zugewiesenen Verantwortlichkeiten wiedergeben

FL-3.2.4 (K2) ... verschiedene Arten von Reviews vergleichen und gegenüberstellen

FL-3.2.5 (K1) ... die Faktoren, die zu einem erfolgreichen Review beitragen, wiedergeben

3.1 Grundlagen des statischen Tests

Im Gegensatz zum dynamischen Test muss beim statischen Test die zu testende Software nicht ausgeführt werden. Code, Prozessspezifikation, Systemarchitekturspezifikation oder andere Arbeitsergebnisse werden durch manuelle Prüfung (z. B. Review) oder mit Hilfe eines Werkzeugs (z. B. statische Analyse) bewertet. Zu den Testzielen gehören die Verbesserung der Qualität, die Aufdeckung von Fehlerzuständen und die Bewertung von Merkmalen wie Lesbarkeit, Vollständigkeit, Korrektheit, Testbarkeit und Konsistenz. Statische Tests können sowohl zur Verifizierung als auch zur Validierung eingesetzt werden.

Tester, Fachbereichsvertreter und Entwickler arbeiten beim Example-Mapping, beim gemeinsamen Schreiben von User Storys und bei der Verfeinerung (Refinement) des Backlogs zusammen, um sicherzustellen, dass die User Storys und die zugehörigen Arbeitsergebnisse definierten Kriterien entsprechen, z. B. der Definition-of-Ready (siehe Abschnitt 5.1.3). Reviewverfahren können angewendet werden, um sicherzustellen, dass die User Storys vollständig und verständlich sind und testbare Abnahmekriterien enthalten. Indem sie die richtigen Fragen stellen, können Tester die vorgeschlagenen User Storys analysieren, hinterfragen und verbessern.

Die statische Analyse kann Probleme vor dem dynamischen Testen aufdecken und ist oft mit weniger Aufwand verbunden, da keine Testfälle erforderlich sind und in der Regel Werkzeuge (siehe Kapitel 6) verwendet werden. Die statische Analyse wird häufig in CI-Frameworks integriert (siehe Abschnitt 2.1.4). Die statische Analyse wird zwar hauptsächlich zur Erkennung spezifischer Fehlerzustände im Code eingesetzt, dient aber auch zur Bewertung der Wartbarkeit und IT-Sicherheit. Rechtschreibprüfung und Werkzeuge zur Prüfung der Lesbarkeit sind weitere Beispiele für statische Analysewerkzeuge.

3.1.1 Arbeitsergebnisse, die durch statische Tests untersucht werden können

Fast jedes Arbeitsergebnis kann mit statischen Tests untersucht werden. Beispiele hierfür sind Dokumente zur Spezifikation von Anforderungen, Quellcode, Testkonzepte, Testfälle, Produkt-Backlog-Elemente, Test-Chartas, Projektdokumentation, Verträge und Modelle.

Jedes Arbeitsergebnis, das gelesen und verstanden werden kann, kann Gegenstand eines Reviews sein. Für die statische Analyse benötigen Arbeitsergebnisse jedoch eine Struktur, anhand derer sie überprüft werden können, z. B. Modelle, Code oder Text mit einer formalen Syntax.

Zu den Arbeitsergebnissen, die sich nicht für statische Tests eignen, gehören solche, die für den Menschen schwer zu interpretieren sind und die nicht mit Hilfe von Werkzeugen analysiert werden sollten, z. B. ausführbarer Code von Drittanbietern, welcher aus rechtlichen Gründen nicht untersucht werden darf.

3.1.2 Wert des statischen Tests

Der Statische Test kann Fehlerzustände in den frühesten Phasen des SDLC aufdecken und erfüllt damit den Grundsatz des frühen Testens (siehe Abschnitt 1.3). Es können auch Fehlerzustände aufgedeckt werden, die durch dynamische Tests nicht erkannt werden

können, z. B. nicht erreichbarer Code, nicht wie gewünscht implementierte Entwurfsmuster, Fehlerzustände in nicht ausführbaren Arbeitsergebnissen.

Der Statische Test bietet die Möglichkeit, die Qualität von Arbeitsergebnissen zu bewerten und Vertrauen in sie aufzubauen. Durch die Überprüfung der dokumentierten Anforderungen können die Stakeholder auch sicherstellen, dass diese Anforderungen ihre tatsächlichen Bedürfnisse beschreiben. Da der Statische Test bereits in einer frühen Phase des SDLC durchgeführt werden kann, kann ein gemeinsames Verständnis zwischen den beteiligten Stakeholdern geschaffen werden. Auch die Kommunikation zwischen den beteiligten Stakeholdern wird verbessert. Aus diesem Grund ist es empfehlenswert, eine Vielzahl von Stakeholdern in statische Tests einzubeziehen.

Auch wenn die Durchführung von Reviews Kosten verursacht, sind die Gesamtkosten des Projekts in der Regel wesentlich geringer, als wenn keine Reviews durchgeführt werden. Das liegt daran, dass weniger Zeit und Aufwand für die Behebung von Fehlerzuständen im späteren Verlauf des Projekts aufgewendet werden müssen.

Fehlerzustände im Code können durch statische Analyse effizienter aufgedeckt werden als durch dynamische Tests, was in der Regel sowohl zu weniger Codefehlern als auch zu einem geringeren Gesamtentwicklungsaufwand führt.

3.1.3 Unterschiede zwischen statischem Test und dynamischem Test

Statischer Test und dynamischer Test ergänzen sich gegenseitig. Sie haben ähnliche Ziele, wie z. B. die Unterstützung bei der Erkennung von Fehlerzuständen in Arbeitsergebnissen (siehe Abschnitt 1.1.1), aber es gibt auch einige Unterschiede, wie z. B.:

- Sowohl statischer als auch dynamischer Test (mit Analyse der Fehlerwirkungen) können zur Entdeckung von Fehlerzuständen führen, allerdings gibt es einige Fehlerzustände, die nur durch statischen oder dynamischen Test gefunden werden können.
- Beim statischen Test werden Fehlerzustände direkt gefunden, während beim dynamischen Test Fehlerwirkungen auftreten, aus denen durch eine anschließende Analyse die zugehörigen Fehlerzustände ermittelt werden.
- Statischer Test kann leichter Fehlerzustände aufdecken, die auf Pfaden durch den Code liegen, die selten ausgeführt werden oder die schwer durch dynamische Tests zu erreichen sind.
- Statischer Test kann auf nicht ausführbare Arbeitsergebnisse angewandt werden, während dynamischer Test nur auf ausführbare Arbeitsergebnisse angewandt werden kann.
- Statischer Test kann zur Messung von Qualitätsmerkmalen verwendet werden, die nicht von der Ausführung des Codes abhängen (z. B. Wartbarkeit), während dynamischer Test zur Messung von Qualitätsmerkmalen verwendet werden kann, die von der Ausführung des Codes abhängen (z. B. Performanz).

Typische Fehlerzustände, die durch statische Tests leichter und/oder kostengünstiger zu finden sind, sind:

- Fehlerzustände in Anforderungen, z. B. Inkonsistenzen, Mehrdeutigkeiten, Widersprüche, Auslassungen, Ungenauigkeiten, Duplikationen
- Fehlerzustände im Entwurf, z. B. ineffiziente Datenbankstrukturen, schlechte Modularität
- Bestimmte Arten von Fehlerzuständen im Code, z. B. Variablen mit undefinierten Werten, nicht deklarierte Variablen, unerreichbarer oder duplizierter Code, übermäßige Komplexität des Codes
- Abweichungen von Standards, z. B. mangelnde Einhaltung von Namenskonventionen in Programmierstandards
- Falsche Spezifikation von Schnittstellen, z. B. nicht übereinstimmende Anzahl, Art oder Reihenfolge von Parametern
- Spezifische Arten von Schwachstellen in der IT-Sicherheit, z. B. Pufferüberläufe
- Lücken oder Ungenauigkeiten in der Überdeckung der Testbasis, z. B. fehlende Tests für ein Abnahmekriterium

3.2 Feedback- und Reviewprozess

3.2.1 Vorteile eines frühzeitigen und häufigen Stakeholder-Feedbacks

Ein frühzeitiges und häufiges Feedback ermöglicht die frühzeitige Kommunikation von potenziellen Qualitätsproblemen. Wenn die Stakeholder während des SDLC nur wenig einbezogen werden, entspricht das zu entwickelnde Produkt möglicherweise nicht den ursprünglichen oder aktuellen Vorstellungen der Stakeholder. Wenn die Wünsche der Stakeholder nicht erfüllt werden, kann dies zu kostspieligen Nacharbeiten, verpassten Terminen, Schuldzuweisungen und sogar zu einem kompletten Scheitern des Projekts führen.

Häufiges Feedback der Stakeholder während des SDLC kann Missverständnisse über Anforderungen vorbeugen und sicherstellen, dass Änderungen an den Anforderungen verstanden und früher umgesetzt werden. Dies hilft dem Entwicklungsteam dabei, besser zu verstehen, was es entwickelt. Es ermöglicht ihm, sich auf die Features zu konzentrieren, die für die Stakeholder den größten Nutzen bringen und die sich am positivsten auf die identifizierten Risiken haben.

3.2.2 Aktivitäten des Reviewprozesses

Die Norm ISO/IEC 20246 definiert einen generischen Reviewprozess, der einen strukturierten, aber flexiblen Rahmen bietet, auf dessen Grundlage ein spezifischer Reviewprozess auf eine bestimmte Situation zugeschnitten werden kann. Wenn das geforderte Review eher formal ist, werden mehr der beschriebenen Aufgaben für die verschiedenen Aktivitäten benötigt.

Viele Arbeitsergebnisse sind zu umfangreich, als dass sie in einem einzigen Review behandelt werden könnten. Der Reviewprozess kann daher mehrfach durchgeführt werden, um das Review für das gesamte Arbeitsergebnis zu vervollständigen.

Die Aktivitäten des Reviewprozesses sind:

- **Planung.** In der Planungsphase wird der Umfang des Reviews festgelegt, der den Zweck, das zu überprüfende Arbeitsergebnis, die zu bewertenden Qualitätsmerkmale, die zu berücksichtigenden Bereiche, die Endekriterien, unterstützende Informationen wie Normen, den Aufwand und den Zeitrahmen für das Review umfasst.
- **Reviewbeginn.** Während des Reviewbeginns geht es darum, sicherzustellen, dass jeder und alles, der oder was benötigt wird, vorbereitet ist, um mit dem Review zu starten. Dazu gehört auch, dass jeder Teilnehmer Zugang zu dem zu prüfenden Arbeitsergebnis hat, seine Rolle und Verantwortlichkeiten versteht und alles erhält, was er für die Durchführung des Reviews benötigt.
- **Individuelles Review**. Jeder Gutachter führt ein individuelles Review durch, um die Qualität des zu prüfenden Arbeitsergebnisses zu bewerten und Anomalien, Empfehlungen und Fragen zu identifizieren, indem er ein oder mehrere Reviewverfahren anwendet (z. B. checklistenbasiertes Review, szenariobasiertes Review). Die Norm ISO/IEC 20246 geht näher auf die verschiedenen Reviewverfahren ein. Die Gutachter protokollieren alle von ihnen identifizierten Anomalien, Empfehlungen und Fragen.
- **Kommunikation und Analyse.** Da es sich bei den während eines Reviews festgestellten Anomalien nicht unbedingt um Fehlerzustände handelt, müssen alle diese Anomalien analysiert und diskutiert werden. Für jede Anomalie sollte eine Entscheidung über ihren Status, ihre Verantwortlichkeit und die erforderlichen Maßnahmen getroffen werden. Dies geschieht in der Regel in einer Reviewsitzung, in der die Teilnehmer auch über die Qualität des geprüften Arbeitsergebnisses und über die erforderlichen Folgemaßnahmen entscheiden. Nach Abschluss der Maßnahmen kann ein Folgereview erforderlich sein.
- **Behebung und Berichterstattung.** Für jeden Fehlerzustand sollte ein Fehlerbericht erstellt werden, damit die Korrekturmaßnahmen nachverfolgt werden können. Wenn die Endekriterien erreicht sind, kann das Arbeitsergebnis abgenommen werden. Über die Ergebnisse des Reviews wird berichtet.

3.2.3 Rollen und Verantwortlichkeiten bei Reviews

An Reviews sind verschiedene Stakeholder beteiligt, die mehrere Rollen einnehmen können. Die wichtigsten Rollen und ihre Verantwortlichkeiten sind:

- **Manager**: entscheidet, was geprüft werden soll, und stellt Ressourcen wie Personal und Zeit für das Review zur Verfügung.
- **Autor**: erstellt und korrigiert das Arbeitsergebnis des Reviews.

- **Moderator** (auch Facilitator genannt): sorgt für einen effektiven Ablauf der Reviewsitzungen, einschließlich Mediation, Zeitmanagement und einer geschützten Reviewumgebung, in der jeder frei sprechen kann.
- **Protokollant**: sammelt Anomalien von Gutachtern und zeichnet Reviewinformationen auf, z. B. Entscheidungen und neue Anomalien, die während der Reviewsitzung gefunden werden.
- **Gutachter**: führt Reviews durch. Ein Gutachter (auch Reviewer genannt) kann ein Projektmitarbeiter, ein Fachexperte oder ein anderer Stakeholder sein.
- **Reviewleiter**: übernimmt die Gesamtverantwortung für das Review, z. B. die Entscheidung, wer daran teilnimmt, und die Organisation, wann und wo das Review stattfindet.

Andere, detailliertere Rollen sind möglich, wie in der Norm ISO/IEC 20246 beschrieben.

3.2.4 Arten von Reviews

Es gibt viele Arten von Reviews, die von informellen Reviews bis zu formalen Reviews reichen. Der erforderliche Grad an Formalität hängt von Faktoren wie dem angewandten SDLC, der Reife des Entwicklungsprozesses, der Kritikalität und Komplexität des zu prüfenden Arbeitsergebnisses, gesetzlichen oder regulatorischen Anforderungen und dem Bedarf an einem Prüfnachweis ab. Ein und dasselbe Arbeitsergebnis kann mit verschiedenen Reviewarten geprüft werden, z. B. zunächst mit einem informellen und später mit einem formaleren Review.

Die Auswahl der richtigen Reviewart ist der Schlüssel zum Erreichen der geforderten Reviewziele (siehe Abschnitt 3.2.5). Die Auswahl richtet sich nicht nur nach den Zielen, sondern auch nach Faktoren wie dem Projektbedarf, den verfügbaren Ressourcen, der Art des Arbeitsergebnisses und seinen Risiken, dem Unternehmensbereich und der Unternehmenskultur.

Einige häufig verwendete Arten von Reviews sind:

- **Informelles Review.** Ein informelles Review folgt keinem definierten Prozess und erfordert keine formalen, dokumentierten Ergebnisse. Das Hauptziel ist die Aufdeckung von Anomalien.
- **Walkthrough.** Ein Walkthrough, das vom Autor geleitet wird, kann vielen Zielen dienen, z. B. der Bewertung der Qualität und dem Aufbau von Vertrauen in das Arbeitsergebnis, der Schulung von Gutachtern, der Erzielung eines Konsenses, der Generierung neuer Ideen, der Motivation und Befähigung von Autoren zur Verbesserung und der Aufdeckung von Anomalien. Gutachter können vor dem Walkthrough ein individuelles Review durchführen, dies ist jedoch nicht verpflichtend.
- **Technisches Review.** Ein Technisches Review wird von technisch qualifizierten Gutachtern durchgeführt und von einem Moderator geleitet. Die Ziele eines Technischen Reviews sind die Erzielung eines Konsenses und die Entscheidungsfindung in Bezug auf ein technisches Problem, aber auch die Aufdeckung von Anomalien, die Bewertung der Qualität und der Aufbau von Vertrauen

in das Arbeitsergebnis, die Entwicklung neuer Ideen sowie die Motivation und Befähigung der Autoren zur Verbesserung.

- **Inspektion.** Da die Inspektion die formalste Art der Reviews ist, folgt sie dem vollständigen allgemeinen Prozess (siehe Abschnitt 3.2.2). Das Hauptziel besteht darin, die maximale Anzahl von Anomalien zu finden. Weitere Ziele sind die Bewertung der Qualität, der Aufbau von Vertrauen in das Arbeitsergebnis und die Motivation und Befähigung der Autoren zur Verbesserung. Es werden Metriken gesammelt und zur Verbesserung des SDLC, einschließlich des Inspektionsprozesses, verwendet. Bei Inspektionen kann der Autor nicht als Reviewleiter oder Protokollant agieren.

3.2.5 Erfolgsfaktoren für Reviews

Es gibt mehrere Faktoren, die den Erfolg von Reviews bestimmen, dazu gehören u. a.:

- Die Festlegung klarer Ziele und messbarer Endekriterien. Die Bewertung der Teilnehmer sollte niemals ein Ziel sein.
- Auswahl der geeigneten Reviewart, um die vorgegebenen Ziele zu erreichen und um der Art des Arbeitsergebnisses, den Reviewteilnehmern, den Projektanforderungen und dem Kontext gerecht zu werden.
- Durchführung von Reviews in kleinen Einheiten, damit die Gutachter während eines individuellen Reviews und/oder der Reviewsitzung (sofern diese stattfindet) nicht die Konzentration verlieren.
- Lieferung von Feedback aus Reviews an die Stakeholder und Autoren, damit diese das Produkt und ihre Aktivitäten verbessern können (siehe Abschnitt 3.2.1).
- Bereitstellung von ausreichend Zeit für die Teilnehmer zur Vorbereitung auf das Review.
- Unterstützung des Reviewprozesses durch das Management.
- Einbeziehung der Reviews in die Unternehmenskultur, um Lernen und Prozessverbesserung zu fördern.
- Angebot geeigneter Schulungen für alle Teilnehmer, damit sie wissen, wie sie ihre Rolle erfüllen können.
- Moderation der Sitzungen.

4. Testanalyse und -entwurf – 390 Minuten

Schlüsselbegriffe

Abnahmekriterien, abnahmetestgetriebene Entwicklung, Äquivalenzklassenbildung, Anweisungsüberdeckung, auf Zusammenarbeit basierender Testansatz, Black-Box-Testverfahren, checklistenbasierter Test, Entscheidungstabellentest, erfahrungsbasierte Testverfahren, explorativer Test, Grenzwertanalyse, intuitive Testfallermittlung, Testverfahren, Überdeckung, Überdeckungselement, White-Box-Testverfahren, Zweigüberdeckung, Zustandsübergangstests

Lernziele für Kapitel 4: Der Lernende kann ...

4.1 Testverfahren im Überblick

FL-4.1.1 (K2) ... Black-Box-Testverfahren, White-Box-Testverfahren und erfahrungsbasierte Testverfahren unterscheiden

4.2 Black-Box-Testverfahren

FL-4.2.1 (K3) ... Äquivalenzklassenbildung zur Ableitung von Testfällen anwenden

FL-4.2.2 (K3) ... Grenzwertanalyse zur Ableitung von Testfällen anwenden

FL-4.2.3 (K3) ... Entscheidungstabellentest zur Ableitung von Testfällen anwenden

FL-4.2.4 (K3) ... Zustandsübergangstest zur Ableitung von Testfällen anwenden

4.3 White-Box-Testverfahren

FL-4.3.1 (K2) ... Anweisungstest erklären

FL-4.3.2 (K2) ... Zweigtest erklären

FL-4.3.3 (K2) ... den Wert des White-Box-Tests erklären

4.4 Erfahrungsbasierter Test

FL-4.4.1 (K2) ... intuitive Testfallermittlung erklären

FL-4.4.2 (K2) ... explorativen Test erklären

FL-4.4.3 (K2) ... checklistenbasierten Test erklären

4.4 Auf Zusammenarbeit basierende Testansätze

FL-4.5.1 (K2) ... das Schreiben von User Storys in Zusammenarbeit mit Entwicklern und Fachvertretern erklären

FL-4.5.2 (K2) ... die verschiedenen Möglichkeiten zum Schreiben von Abnahmekriterien einordnen

FL-4.5.3 (K3) ... abnahmetestgetriebene Entwicklung (ATDD) zur Ableitung von Testfällen anwenden

4.1 Testverfahren im Überblick

Testverfahren unterstützen den Tester bei der Testanalyse (was soll getestet werden) und beim Testentwurf (wie soll getestet werden). Testverfahren helfen dabei, eine relativ kleine, aber ausreichende Menge von Testfällen systematisch zu entwickeln. Testverfahren helfen dem Tester auch bei der Definition von Testbedingungen, der Identifizierung von Überdeckungselementen und der Identifizierung von Testdaten während der Testanalyse und des Testentwurfs. Weitere Informationen zu Testverfahren und ihren entsprechenden Messgrößen finden sich in der Norm ISO/IEC/IEEE 29119-4 sowie in (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019).

In diesem Lehrplan werden Testverfahren als Black-Box-, White-Box- und erfahrungsbasiert klassifiziert.

Black-Box-Testverfahren (auch spezifikationsbasierte Verfahren genannt) basieren auf einer Analyse des spezifizierten Verhaltens des Testobjekts ohne Kenntnis der internen Struktur. Daher werden die Testfälle unabhängig von der Implementierung der Software erstellt. Folglich sind die Testfälle auch dann noch nützlich, wenn sich die Implementierung ändert, das geforderte Verhalten aber gleichbleibt.

White-Box-Testverfahren (auch als strukturbasierte Verfahren bekannt) basieren auf einer Analyse der internen Struktur und Verarbeitung des Testobjekts. Da die Testfälle vom Entwurf der Software abhängig sind, können sie erst nach dem Entwurf oder der Implementierung des Testobjekts erstellt werden.

Erfahrungsbasierte Testverfahren nutzen das Wissen und die Erfahrung von Testern effektiv für den Entwurf und die Implementierung von Testfällen. Die Effektivität dieser Verfahren hängt stark von den Kenntnissen des Testers ab. Mit erfahrungsbasierten Testverfahren können Fehlerzustände aufgedeckt werden, die bei Black-Box- und White-Box-Testverfahren möglicherweise übersehen werden. Daher ergänzen erfahrungsbasierte Testverfahren Black-Box- und White-Box-Testverfahren.

4.2 Black-Box-Testverfahren

Die folgenden Abschnitte behandeln die üblichen Black-Box-Testverfahren:

- Äquivalenzklassenbildung
- Grenzwertanalyse
- Entscheidungstabellentest
- Zustandsübergangstest

4.2.1 Äquivalenzklassenbildung

Bei der Äquivalenzklassenbildung werden Daten in Klassen (so genannte Äquivalenzklassen) unterteilt, wobei davon ausgegangen wird, dass alle Elemente einer bestimmten Klasse vom Testobjekt auf die gleiche Weise verarbeitet werden. Die Theorie hinter diesem Verfahren ist,

dass, wenn ein Testfall, der einen Wert aus einer Äquivalenzklasse testet, einen Fehlerzustand entdeckt, dieser Fehlerzustand auch von Testfällen entdeckt worden wäre, die einen beliebigen anderen Wert aus derselben Klasse testen. Daher reicht ein Test je Klasse aus.

Äquivalenzklassen können für jedes dem Testobjekt zugehörigem Datenelement ermittelt werden, einschließlich Eingaben, Ausgaben, Konfigurationselementen, internen Werten, zeitbezogenen Werten und Schnittstellenparametern. Die Klassen können zusammenhängend oder einzeln, geordnet oder ungeordnet, endlich oder unendlich sein. Die Klassen dürfen sich nicht überschneiden und müssen nicht-leere Mengen sein.

Bei einfachen Testobjekten kann die Äquivalenzklassenbildung leicht sein, aber in der Praxis ist es oft kompliziert zu verstehen, wie das Testobjekt verschiedene Werte verarbeitet. Daher sollte die Klassenbildung mit Sorgfalt vorgenommen werden.

Eine Klasse, die gültige Werte enthält, wird als gültige Klasse bezeichnet. Eine Klasse, die ungültige Werte enthält, wird als ungültige Klasse bezeichnet. Die Definitionen von gültigen und ungültigen Werten können je nach Team und Unternehmen variieren. So können beispielsweise gültige Werte als solche interpretiert werden, die vom Testobjekt verarbeitet werden sollten, oder als solche, für die die Spezifikation ihre Verarbeitung definiert. Ungültige Werte können als solche interpretiert werden, die vom Testobjekt ignoriert oder zurückgewiesen werden sollen, oder als solche, für die in der Spezifikation des Testobjekts keine Verarbeitung festgelegt ist.

In der Äquivalenzklassenbildung bilden die Äquivalenzklassen die Überdeckungselemente. Um mit diesem Verfahren eine 100%ige Überdeckung zu erreichen, müssen die Testfälle alle identifizierten Klassen (einschließlich ungültiger Klassen) mindestens einmal ausführen. Die Überdeckung wird gemessen als die Anzahl der Klassen, die von mindestens einem Testfall ausgeführt wurden, geteilt durch die Gesamtzahl der identifizierten Klassen, und wird in Prozent ausgedrückt.

Viele Testobjekte umfassen mehrere Gruppen von Klassen (z. B. Testobjekte mit mehr als einem Eingabeparameter), was bedeutet, dass ein Testfall Klassen aus verschiedenen Gruppen von Klassen abdeckt. Das einfachste Überdeckungskriterium für den Fall mehrerer Klassensätze ist die Each-Choice-Überdeckung (Ammann 2016). Diese verlangt, dass jede Klasse aus jeder Gruppe von Klassen durch Testfälle mindestens einmal ausgeführt werden. Die Each-Choice-Überdeckung berücksichtigt keine gezielten Kombinationen von Klassen.

4.2.2 Grenzwertanalyse

Die Grenzwertanalyse ist ein Verfahren, das auf der Überprüfung der Grenzen von Äquivalenzklassen basiert. Daher kann die Grenzwertanalyse nur für geordnete Klassen verwendet werden. Die Minimum- und Maximumwerte einer Klasse sind ihre Grenzwerte. Wenn zwei Elemente zur gleichen Klasse gehören, müssen bei der Grenzwertanalyse alle Elemente zwischen ihnen ebenfalls zu dieser Klasse gehören.

Grenzwertanalyse konzentriert sich auf die Grenzwerte der Klassen, weil Entwickler bei diesen Grenzwerten eher Fehlhandlungen unterlaufen. Typische Fehlerzustände, die durch Grenzwertanalyse gefunden werden, liegen dort, wo implementierte Grenzen an Positionen oberhalb oder unterhalb ihrer beabsichtigten Positionen verschoben oder ganz ausgelassen werden.

In diesem Lehrplan werden zwei Versionen der Grenzwertanalyse behandelt: 2-Wert-Grenzwertanalyse und 3-Wert-Grenzwertanalyse. Sie unterscheiden sich in der Anzahl der Überdeckungselemente pro Grenzwert, die ausgeführt werden müssen, um eine 100%ige Überdeckung zu erreichen.

Bei der **2-Wert-Grenzwertanalyse** (Craig 2002, Myers 2011) gibt es für jeden Grenzwert zwei Überdeckungselemente: den Grenzwert und seinen engsten Nachbarn, der zur angrenzenden Klasse gehört. Um bei der 2-Wert-Grenzwertanalyse eine 100%ige Überdeckung zu erreichen, müssen die Testfälle alle Überdeckungselemente, d. h. alle identifizierten Grenzwerte, ausführen. Die Überdeckung wird gemessen als die Anzahl der ausgeführten Grenzwerte, geteilt durch die Gesamtzahl der identifizierten Grenzwerte, und wird in Prozent ausgedrückt.

In der **3-Wert-Grenzwertanalyse** (Koomen 2006, O'Regan 2019) gibt es für jeden Grenzwert drei Überdeckungselemente: den Grenzwert und seine beiden Nachbarn. Daher können bei der 3-Wert-Grenzwertanalyse einige der Überdeckungselemente keine Grenzwerte sein. Um bei der 3-Wert-Grenzwertanalyse eine 100%ige Überdeckung zu erreichen, müssen die Testfälle alle Überdeckungselemente, d. h. die identifizierten Grenzwerte und deren Nachbarn, ausführen. Die Überdeckung wird gemessen als die Anzahl der ausgeführten Grenzwerte und ihrer Nachbarn, geteilt durch die Gesamtzahl der identifizierten Grenzwerte und ihrer Nachbarn, und wird in Prozent ausgedrückt.

Die 3-Wert-Grenzwertanalyse ist strenger als die 2-Wert-Grenzwertanalyse, da sie Fehlerzustände aufdecken kann, die bei der 2-Wert-Grenzwertanalyse übersehen wurden. Wenn beispielsweise die Entscheidung "wenn (x ≤ 10) ..." fälschlicherweise als "wenn (x = 10) ..." implementiert wird, kann keiner der aus der 2-Wert-Grenzwertanalyse abgeleiteten Testdaten (x = 10, x = 11) den Fehlerzustand aufdecken. Mit x = 9, abgeleitet aus der 3-Wert-Grenzwertanalyse, wird der Fehlerzustand jedoch mit hoher Wahrscheinlichkeit entdeckt.

4.2.3 Entscheidungstabellentest

Entscheidungstabellen werden zum Testen der Umsetzung von Systemanforderungen verwendet, die angeben, wie verschiedene Kombinationen von Bedingungen zu unterschiedlichen Ergebnissen führen. Entscheidungstabellen sind ein effektives Mittel zur Erfassung komplexer Logik, wie z. B. Geschäftsregeln.

Bei der Erstellung von Entscheidungstabellen werden die Bedingungen und die daraus resultierenden Aktionen des Systems definiert. Diese bilden die Zeilen der Tabelle. Jede Spalte entspricht einer Entscheidungsregel, die eine eindeutige Kombination von Bedingungen zusammen mit den zugehörigen Aktionen definiert. In Entscheidungstabellen mit eingeschränkter Eingabe werden alle Werte der Bedingungen und Aktionen (mit Ausnahme der irrelevanten oder undurchführbaren, siehe unten) als boolesche Werte (wahr oder falsch) dargestellt. Alternativ können in Entscheidungstabellen mit erweiterter Eingabe einige oder alle Bedingungen und Aktionen auch mehrere Werte annehmen (z. B. Zahlenbereiche, Äquivalenzklassen, Einzelwerte).

Die Notation für Bedingungen ist wie folgt: "J" (wahr) bedeutet, dass die Bedingung erfüllt ist. "N" (falsch) bedeutet, dass die Bedingung nicht erfüllt ist. "-" bedeutet, dass der Wert der Bedingung für das Ergebnis der Aktion irrelevant ist. "N/A" bedeutet, dass die Bedingung für eine bestimmte Regel nicht durchführbar ist. Für Aktionen: "X" bedeutet, dass die Aktion

stattfinden sollte. Leer bedeutet, dass die Aktion nicht eintreten sollte. Es können auch andere Notationen verwendet werden.

Eine vollständige Entscheidungstabelle hat genügend Spalten, um jede Kombination von Bedingungen abzudecken. Die Tabelle kann vereinfacht werden, indem Spalten mit undurchführbaren Kombinationen von Bedingungen gelöscht werden. Die Tabelle kann auch minimiert werden, indem Spalten, in denen einige Bedingungen keinen Einfluss auf das Ergebnis haben, in einer einzigen Spalte zusammengefasst werden. Algorithmen zur Minimierung von Entscheidungstabellen sind nicht Gegenstand dieses Lehrplans.

Beim Entscheidungstabellentest sind die Überdeckungselemente die Spalten, die ausführbare Kombinationen von Bedingungen enthalten. Um mit diesem Verfahren eine 100%ige Überdeckung zu erreichen, müssen die Testfälle alle diese Spalten ausführen. Die Überdeckung wird gemessen als die Anzahl der ausgeführten Spalten, geteilt durch die Gesamtzahl der ausführbaren Spalten, und wird in Prozent ausgedrückt.

Die Stärke des Entscheidungstabellentests liegt darin, dass er einen systematischen Ansatz zur Identifizierung aller Kombinationen von Bedingungen bietet, von denen einige andernfalls übersehen werden könnten. Es hilft auch, Lücken oder Widersprüche in den Anforderungen zu finden. Wenn es viele Bedingungen gibt, kann die Anwendung aller Entscheidungsregeln sehr zeitaufwändig sein, da die Anzahl der Regeln exponentiell mit der Anzahl der Bedingungen wächst. In einem solchen Fall kann eine minimierte Entscheidungstabelle oder ein risikobasierter Ansatz verwendet werden, um die Anzahl der auszuführenden Regeln zu reduzieren.

4.2.4 Zustandsübergangstest

Ein Zustandsübergangsdiagramm modelliert das Verhaltes eines Systems, indem es seine möglichen Zustände und gültigen Übergänge aufzeigt. Ein Übergang wird durch ein Ereignis ausgelöst, das zusätzlich durch eine Wächterbedingung (guard condition) qualifiziert werden kann. Es wird davon ausgegangen, dass die Übergänge augenblicklich erfolgen und manchmal dazu führen, dass die Software eine Aktion ausführt. Die übliche Syntax zur Kennzeichnung von Übergängen lautet wie folgt: "Ereignis [Wächterbedingung] / Aktion". Wächterbedingungen und Aktionen können weggelassen werden, wenn sie nicht existieren oder für den Tester irrelevant sind.

Eine Zustandstabelle ist ein Modell, das einem Zustandsübergangsdiagramm entspricht. Ihre Zeilen stellen Zustände dar, ihre Spalten Ereignisse (zusammen mit Wächterbedingungen, falls vorhanden). Die Tabelleneinträge (Zellen) stellen Übergänge dar und enthalten den Zielzustand sowie die daraus resultierenden Aktionen, falls definiert. Im Gegensatz zum Zustandsübergangsdiagramm zeigt die Zustandstabelle explizit ungültige Übergänge an, die durch leere Zellen dargestellt werden.

Ein Testfall, der auf einem Zustandsübergangsdiagramm oder einer Zustandstabelle basiert, wird in der Regel als eine Folge von Ereignissen dargestellt, die zu einer Abfolge von Zustandsänderungen (und ggf. Aktionen) führt. Ein Testfall kann und wird in der Regel mehrere Übergänge zwischen den Zuständen abdecken.

Es gibt viele Überdeckungskriterien für Zustandsübergangstests. In diesem Lehrplan werden drei von ihnen behandelt.

Bei der **Überdeckung aller Zustände** sind die Überdeckungselemente die Zustände. Um eine 100%ige Überdeckung aller Zustände zu erreichen, müssen die Testfälle sicherstellen, dass alle Zustände besucht werden. Die Überdeckung wird als Anzahl der besuchten Zustände geteilt durch die Gesamtzahl der Zustände gemessen und in Prozent ausgedrückt.

Bei der **Überdeckung der gültigen Übergänge** (auch 0-Switch-Überdeckung genannt) handelt es sich bei den Überdeckungselementen um einzelne gültige Übergänge. Um eine 100%ige Überdeckung der gültigen Übergänge zu erreichen, müssen die Testfälle alle gültigen Übergänge ausführen. Die Überdeckung wird als Anzahl der ausgeführten gültigen Übergänge geteilt durch die Gesamtzahl der gültigen Übergänge gemessen und in Prozent ausgedrückt.

Bei der **Überdeckung aller Übergänge** handelt es sich bei den Überdeckungselementen um alle Übergänge, die in einer Zustandstabelle aufgeführt sind. Um eine 100%ige Überdeckung aller Übergänge zu erreichen, müssen die Testfälle alle gültigen Übergänge ausführen und versuchen, ungültige Übergänge auszuführen. Das Testen von nur einem ungültigen Übergang in einem einzigen Testfall hilft dabei, Fehlermaskierung zu vermeiden, d. h. eine Situation, in der ein Fehlerzustand die Entdeckung eines anderen verhindert. Die Überdeckung wird als Anzahl der gültigen und ungültigen Übergänge, die durch die Testfälle ausgeführt oder versucht wurden, auszuführen, geteilt durch die Gesamtzahl der gültigen und ungültigen Übergänge gemessen und in Prozent ausgedrückt.

Die Überdeckung aller Zustände ist schwächer als die Überdeckung aller Übergänge, da sie in der Regel erreicht werden kann, ohne alle Übergänge zu testen. Die Überdeckung der gültigen Übergänge ist das am häufigsten verwendete Überdeckungskriterium. Eine vollständige Überdeckung aller Übergänge garantiert eine vollständige Überdeckung aller Zustände. Das Erreichen der vollständigen Überdeckung aller Übergänge garantiert sowohl die vollständige Überdeckung aller Zustände als auch die vollständige Überdeckung der gültigen Übergänge und sollte eine Mindestanforderung für unternehmenskritische und sicherheitskritische Software sein.

4.3 White-Box-Test

Aufgrund ihrer Verbreitung und Einfachheit konzentriert sich dieser Abschnitt auf zwei codebezogene White-Box-Testverfahren:

- Anweisungstest
- Zweigtest

Es gibt gründlichere Verfahren, die in einigen sicherheitskritischen, unternehmenskritischen oder hochgradig integrierten Umgebungen eingesetzt werden, um eine gründlichere Codeüberdeckung zu erreichen. Es gibt auch White-Box-Testverfahren, die in höheren Teststufen eingesetzt werden (z. B. bei API-Test) oder die Überdeckungen verwenden, die sich nicht auf den Code beziehen (z. B. Neuronenüberdeckung beim Testen von neuronalen Netzen). Diese Verfahren werden in diesem Lehrplan nicht behandelt.

4.3.1 Anweisungstest und Anweisungsüberdeckung

Beim Anweisungstest sind die Überdeckungselemente ausführbare Anweisungen. Ziel ist es, Testfälle zu entwerfen, die Anweisungen im Code auszuführen, bis eine akzeptable Anweisungsüberdeckung erreicht ist. Die Überdeckung wird als Anzahl der durch die Testfälle ausgeführten Anweisungen geteilt durch die Gesamtzahl der ausführbaren Anweisungen im Code gemessen und wird in Prozent ausgedrückt.

Wenn eine Anweisungsüberdeckung von 100% erreicht wird, ist sichergestellt, dass alle ausführbaren Anweisungen im Code mindestens einmal ausgeführt worden sind. Dies bedeutet insbesondere, dass jede Anweisung mit einem Fehlerzustand ausgeführt wird, was zu einer Fehlerwirkung führen kann, die das Vorhandensein des Fehlerzustands beweist. Das Ausführen einer Anweisung mit einem Testfall wird jedoch nicht in allen Fällen Fehlerzustände aufdecken. So werden beispielsweise Fehlerzustände, die datenabhängig sind, nicht erkannt (z. B. eine Division durch Null, die nur fehlschlägt, wenn der Nenner auf Null gesetzt wird). Auch eine 100%ige Anweisungsüberdeckung stellt nicht sicher, dass die gesamte Entscheidungslogik getestet wurde, da z. B. nicht alle Verzweigungen (siehe Kapitel 4.3.2) des Codes ausgeführt werden können.

4.3.2 Zweigtest und Zweigüberdeckung

Ein Zweig ist ein Kontrollübergang zwischen zwei Knoten im Kontrollflussdiagramm, das die möglichen Sequenzen aufzeigt, in denen Quellcodeanweisungen im Testobjekt ausgeführt werden. Jeder Kontrollübergang kann entweder bedingungslos (d. h. geradliniger Code) oder bedingt (d. h. ein Entscheidungsergebnis) sein.

Beim Zweigtest sind die Überdeckungselemente Zweige, und das Ziel ist es, Testfälle zu entwerfen, um die Zweige im Code auszuführen, bis ein akzeptabler Überdeckungsgrad erreicht ist. Die Messgröße der Zweigüberdeckung erfolgt als Anzahl der durch die Testfälle ausgeführten Zweige geteilt durch die Gesamtzahl der Zweige und wird in Prozent ausgedrückt.

Wenn eine 100%ige Zweigüberdeckung erreicht ist, werden alle Zweige des Codes, unbedingte und bedingte, durch Testfälle ausgeführt. Bedingte Verzweigungen entsprechen typischerweise einem wahren oder falschen Ergebnis einer "if...then"-Entscheidung, einem Ergebnis einer switch/case-Anweisung oder einer Entscheidung über den Austritt oder die Fortsetzung einer Schleife. Das Ausführen eines Zweigs mit einem Testfall wird jedoch nicht in allen Fällen Fehlerzustände aufdecken. So werden beispielsweise Fehlerzustände, die die Ausführung eines bestimmten Pfades in einem Code erfordern, nicht erkannt.

Zweigüberdeckung schließt Anweisungsüberdeckung ein. Das bedeutet, dass jeder Satz von Testfällen, der eine 100%ige Zweigüberdeckung erreicht, auch eine 100%ige Anweisungsüberdeckung erreicht (aber nicht umgekehrt).

4.3.3 Der Wert des White-Box-Tests

Eine grundlegende Stärke, die allen White-Box-Testverfahren gemeinsam ist, besteht darin, dass beim Testen die gesamte Softwareimplementierung berücksichtigt wird, was die Erkennung von Fehlerzuständen auch dann erleichtert, wenn die Software-Spezifikation vage,

veraltet oder unvollständig ist. Ein entsprechender Schwachpunkt besteht darin, dass White-Box-Tests, wenn die Software eine oder mehrere Anforderungen nicht erfüllt, die daraus resultierenden Fehlerzustände möglicherweise nicht erkennen (Watson 1996).

White-Box-Testverfahren können beim statischen Testen eingesetzt werden (z. B. bei Dry Runs (Probeläufen) von Code). Sie eignen sich gut für das Review von Code, der noch nicht ausführbar ist (Hetzel 1988), sowie von Pseudocode und anderer High-Level- oder Top-Down-Logik, die mit einem Kontrollflussdiagramm modelliert werden kann.

Die Durchführung von Black-Box-Tests allein liefert keine Messgröße der tatsächlichen Code-Überdeckung. White-Box-Tests bieten eine objektive Messgröße der Überdeckung und liefern die notwendigen Informationen, um zusätzliche Tests zu generieren, die die Überdeckung erhöhen und somit das Vertrauen in den Code stärken.

4.4 Erfahrungsbasierter Test

Übliche erfahrungsbasierte Testverfahren werden in den folgenden Abschnitten besprochen:

- Intuitive Testfallermittlung
- Explorativer Test
- Checklistenbasierter Test

4.4.1 Intuitive Testfallermittlung

Die intuitive Testfallermittlung ist ein Verfahren zur Vorhersage des Auftretens von Fehlhandlungen, Fehlerzuständen und Fehlerwirkungen, die auf dem Wissen des Testers basiert, einschließlich:

- Wie die Anwendung in der Vergangenheit funktioniert hat
- Den Arten von Fehlhandlungen, zu denen die Entwickler neigen, und die Arten von Fehlerzuständen, die aus diesen Fehlhandlungen resultieren
- Den Arten von Fehlerwirkungen, die in anderen, ähnlichen Anwendungen aufgetreten sind

Im Allgemeinen können sich Fehlhandlungen, Fehlerzustände und Fehlerwirkungen auf Folgendes beziehen: Eingabe (z. B. korrekte Eingabe nicht akzeptiert, falsche oder fehlende Parameter), Ausgabe (z. B. falsches Format, falsches Ergebnis), Logik (z. B. fehlende Fälle, falscher Operator), Berechnung (z. B. falscher Operand, falsche Berechnung), Schnittstellen (z. B. falsche Parameterzuordnung, inkompatible Typen) oder Daten (z. B. falsche Initialisierung, falscher Typ).

Fehlerangriffe sind ein methodischer Ansatz für die Durchführung von intuitiven Testfallermittlungen. Bei diesem Verfahren muss der Tester eine Liste möglicher Fehlhandlungen, Fehlerzustände und Fehlerwirkungen erstellen oder übernehmen und Tests entwerfen, die die mit den Fehlhandlungen verbundenen Fehlerzustände identifizieren, die Fehlerzustände aufdecken oder die Fehlerwirkungen verursachen. Diese Listen können auf

der Grundlage von Erfahrungswerten, Daten über Fehlerzustände und Fehlerwirkungen oder auf der Grundlage des allgemeinen Wissens darüber, warum Software fehlschlägt, erstellt werden.

Siehe (Whittaker 2002, Whittaker 2003, Andrews 2006) für weitere Informationen über intuitive Testfallermittlungen und Fehlerangriffe.

4.4.2 Explorativer Test

Beim explorativen Test werden Tests gleichzeitig entworfen, ausgeführt und bewertet, während der Tester mehr über das Testobjekt erfährt. Neben dem genauen Kennenlernen des Testobjekts, wird das Testobjekt mit gezielten Tests gründlicher erforscht und weitere Tests für ungetestete Bereiche erstellt.

Exploratives Testen wird manchmal als sitzungsbasierter Test durchgeführt, um das Testen zu strukturieren. Bei einem sitzungsbasierten Ansatz wird der explorative Test innerhalb eines bestimmten Zeitrahmens durchgeführt. Der Tester verwendet eine Test-Charta mit Testzielen, um das Testen zu steuern. An die Testsitzung schließt sich in der Regel eine Nachbesprechung an, in der der Tester mit den an den Testergebnissen interessierten Beteiligten diskutiert. Bei dieser Testvorgehensweise können abstrakte Testbedingungen als Testziele behandelt werden. Überdeckungselemente werden während der Testsitzung identifiziert und ausgeführt. Der Tester kann Testsitzungsblätter (Session sheets) verwenden, um die durchgeführten Schritte und die gemachten Erkenntnisse zu dokumentieren.

Explorative Tests sind sinnvoll, wenn es nur wenige oder unzureichende Spezifikationen gibt oder der Zeitdruck beim Testen groß ist. Explorative Tests sind auch als Ergänzung zu anderen, eher formalen Testverfahren sinnvoll. Exploratives Testen ist effektiver, wenn der Tester erfahren ist, über Fachkenntnisse verfügt und ein hohes Maß an grundlegenden Kompetenzen wie analytische Fähigkeiten, Neugier und Kreativität besitzt (siehe Abschnitt 1.5.1).

Beim explorativen Test können auch andere Testverfahren zum Einsatz kommen (z. B. Äquivalenzklassenbildung). Weitere Informationen zum explorativen Testen finden sich in (Kaner 1999, Whittaker 2009, Hendrickson 2013).

4.4.3 Checklistenbasierter Test

Beim checklistenbasierten Test entwirft, implementiert und führt ein Tester Tests aus, um Testbedingungen aus einer Checkliste abzudecken. Checklisten können auf der Grundlage von Erfahrungen, dem Wissen darüber, was für den Benutzer wichtig ist, oder einem Verständnis darüber, warum und wie Software fehlgeschlagen ist, erstellt werden. Checklisten sollten keine Elemente enthalten, die automatisch geprüft werden können, Elemente, die sich besser als Eingangs-/Endekriterien eignen, oder Elemente, die zu allgemein sind (Brykczynski 1999).

Die Elemente der Checkliste sind häufig in Form von Fragen formuliert. Es sollte möglich sein, jedes Element einzeln und direkt zu prüfen. Diese Elemente können sich auf Anforderungen, grafische Oberflächeneigenschaften, Qualitätsmerkmale oder andere Formen von Testbedingungen beziehen. Checklisten können zur Unterstützung verschiedener Testarten,

einschließlich funktionaler und nicht-funktionaler Tests, erstellt werden (z. B. 10 Heuristiken für Gebrauchstauglichkeitstests (Nielsen 1994)).

Einige Checklisteneinträge können im Laufe der Zeit an Effektivität verlieren, weil die Entwickler lernen, dieselben Fehlhandlungen zu vermeiden. Neue Einträge müssen möglicherweise auch hinzugefügt werden, um neu gefundene Fehlerzustände mit hohem Fehlerschweregrad zu berücksichtigen. Daher sollten Checklisten regelmäßig auf der Grundlage von Fehlerzuständen aktualisiert werden. Es sollte jedoch darauf geachtet werden, dass die Checkliste nicht zu lang wird (Gawande 2009).

Beim Fehlen detaillierter Testfälle kann das checklistenbasierte Testen Richtlinien und ein gewisses Maß an Konsistenz für das Testen bieten. Wenn die Checklisten generisch sind, ist eine gewisse Variabilität beim tatsächlichen Testen wahrscheinlich, was zu einer potenziell größeren Überdeckung, aber weniger Wiederholbarkeit führt.

4.5 Auf Zusammenarbeit basierende Testansätze

Jedes der oben erwähnten Verfahren (siehe Abschnitte 4.2, 4.3, 4.4) verfolgt ein bestimmtes Ziel in Hinblick auf die Erkennung von Fehlerzuständen. Auf Zusammenarbeit basierende Ansätze hingegen konzentrieren sich auch auf die Vermeidung von Fehlerzuständen durch Zusammenarbeit und Kommunikation.

4.5.1 Gemeinsames Schreiben von User Storys

Eine User Story repräsentiert ein Feature, das für einen Benutzer oder Käufer eines Systems oder einer Software nützlich sein wird. User Storys haben drei kritische Aspekte (Jeffries 2000), die zusammen die "3 C's" genannt werden:

- **Karte** (Card) – das Medium, das eine User Story beschreibt (z. B. eine Karteikarte, ein Eintrag auf einem elektronischen Board)
- **Konversation** (Conversation) – erklärt, wie die Software genutzt werden soll (kann dokumentiert oder mündlich erfolgen)
- **Bestätigung** (Confirmation) – die Abnahmekriterien (siehe Abschnitt 4.5.2)

Das gängigste Format für eine User Story ist "Als [Rolle] möchte ich, dass [das zu erreichende Ziel], so dass ich [resultierender Nutzen für die Rolle]", gefolgt von den Abnahmekriterien.

Für die Zusammenarbeit bei der Erstellung der User Story können Verfahren wie Brainstorming und Mind-Mapping eingesetzt werden. Die Zusammenarbeit ermöglicht es dem Team, eine gemeinsame Vision von dem zu erhalten, was geliefert werden soll, indem drei Perspektiven berücksichtigt werden: Fachlichkeit, Entwicklung und Testen.

Gute User Storys sollten sein: Unabhängig (independent), verhandelbar (negotiable), nützlich (valuable), schätzbar (estimable), klein (small) und testbar (testable) (INVEST-Prinzip). Wenn ein Stakeholder nicht weiß, wie er eine User Story testen soll, kann dies darauf hindeuten, dass die User Story nicht klar genug ist, dass sie für ihn keinen erkennbaren Mehrwert darstellt oder dass der Stakeholder einfach Hilfe beim Testen benötigt (Wake 2003).

4.5.2 Abnahmekriterien

Abnahmekriterien für eine User Story (auch Akzeptanzkriterien; aus dem engl. Acceptance criteria) sind die Bedingungen, die eine Implementierung der User Story erfüllen muss, um von den Stakeholdern akzeptiert zu werden. Aus dieser Perspektive können Abnahmekriterien als die Testbedingungen betrachtet werden, die durch die Tests ausgeführt werden sollten. Abnahmekriterien sind in der Regel ein Ergebnis der Diskussion (siehe Abschnitt 4.5.1).

Abnahmekriterien werden verwendet, um:

- den Umfang der User Story zu definieren
- einen Konsens zwischen den Stakeholdern zu erreichen
- sowohl positive als auch negative Szenarien zu beschreiben
- als Basis für Abnahmetests der User Story zu dienen (siehe Abschnitt 5.3.3)
- eine genaue Planung und Schätzung zu ermöglichen

Es gibt mehrere Möglichkeiten, Abnahmekriterien für eine User Story zu formulieren. Die zwei gängigsten Formate sind:

- Szenario-orientiert (z. B. das Gegeben/Wenn/Dann-Format, das in der verhaltensgetriebenen Entwicklung (BDD) verwendet wird, siehe Abschnitt 2.1.3)
- Regelorientiert (z. B. Verifizierungsliste mit Aufzählungspunkten oder tabellarische Form der Input-Output-Zuordnung)

Die meisten Abnahmekriterien lassen sich in einem dieser beiden Formate dokumentieren. Das Team kann jedoch auch ein anderes, benutzerdefiniertes Format verwenden, solange die Abnahmekriterien klar definiert und eindeutig sind.

4.5.3 Abnahmetestgetriebene Entwicklung (ATDD)

ATDD ist ein Test-First-Ansatz (siehe Abschnitt 2.1.3). Testfälle werden vor der Implementierung der User Story erstellt. Die Testfälle werden von Teammitgliedern mit unterschiedlichen Perspektiven erstellt, z. B. von Kunden, Entwicklern und Testern (Adzic 2009). Die Testfälle können manuell oder automatisiert ausgeführt werden.

Der erste Schritt ist ein Spezifikationsworkshop, in dem die User Story und (falls noch nicht definiert) deren Abnahmekriterien von den Teammitgliedern analysiert, diskutiert und geschrieben werden. Unvollständigkeiten, Mehrdeutigkeiten oder Fehlerzustände in der User Story werden in diesem Prozess behoben. Der nächste Schritt ist die Erstellung der Testfälle. Dies kann durch das Team als Ganzes oder durch einen einzelnen Tester geschehen. Die Testfälle basieren auf den Abnahmekriterien und können als Beispiele für die Funktionsweise der Software angesehen werden. Dies hilft dem Team, die User Story korrekt umzusetzen.

Da Beispiele und Tests dasselbe sind, werden diese Begriffe oft synonym verwendet. Während des Testentwurfs können die in den Abschnitten 4.2, 4.3 und 4.4 beschriebenen Testverfahren angewandt werden.

Typischerweise sind die ersten Testfälle positiv, bestätigen das korrekte Verhalten ohne Ausnahmen oder Fehlerbedingungen und umfassen die Abfolge der Aktivitäten, die ausgeführt werden, wenn alles wie erwartet abläuft. Nachdem die positiven Testfälle abgeschlossen sind, sollte das Team Negativtests durchführen. Schließlich sollte das Team auch nicht-funktionale Qualitätsmerkmale abdecken (z. B. Performanz, Gebrauchstauglichkeit). Testfälle sollten so formuliert werden, dass sie für die Stakeholder verständlich sind. In der Regel bestehen Testfälle aus Sätzen in natürlicher Sprache, die die notwendigen Vorbedingungen (falls vorhanden), die Eingaben und die Nachbedingungen enthalten.

Die Testfälle müssen alle Merkmale der User Story abdecken und sollten nicht über sie hinausgehen. Die Abnahmekriterien können jedoch auf einige der in der User Story beschriebenen Probleme eingehen. Darüber hinaus sollten keine zwei Testfälle dieselben Merkmale der User Story beschreiben.

Wenn die Testfälle in einem Format erfasst werden, das von einem Testautomatisierungs-framework unterstützt wird, können die Entwickler die Testfälle automatisieren, indem sie den unterstützenden Code schreiben, während sie die in einer User Story beschriebenes Feature implementieren. Die Abnahmetests werden dann zu ausführbaren Anforderungen.

5. Management der Testaktivitäten – 335 Minuten

Schlüsselbegriffe

Eingangskriterien, Endekriterien, Fehlerbericht, Fehlermanagement, Produktrisiko, Projektrisiko, Risiko, Risikoanalyse, risikobasierter Test, Risikobewertung, Risikoidentifizierung, Risikomanagement, Risikominderung, Risikosteuerung, Risikostufe, Risikoüberwachung, Testabschlussbericht, Testansatz, Testfortschrittsbericht, Testkonzept, Testplanung, Testpyramide, Testquadranten, Teststeuerung, Testüberwachung

Lernziele für Kapitel 5: Der Lernende kann ...

5.1 Testplanung

FL-5.1.1 (K2) ... Beispiele zu Zweck und Inhalt eines Testkonzepts geben

FL-5.1.2 (K1) ... den möglichen Mehrwert, den ein Tester für die Iterations- und Releaseplanung schafft, erkennen

FL-5.1.3 (K2) ... Eingangskriterien und Endekriterien vergleichen und gegenüberstellen

FL-5.1.4 (K3) ... Schätzverfahren zur Berechnung des erforderlichen Testaufwands anwenden

FL-5.1.5 (K3) ... Priorisierung von Testfällen anwenden

FL-5.1.6 (K1) ... die Konzepte der Testpyramide wiedergeben

FL-5.1.7 (K2) ... die Testquadranten und ihre Beziehungen zu Teststufen und Testarten zusammenfassen

5.2 Risikomanagement

FL-5.2.1 (K1) ... die Risikostufe anhand der Eintrittswahrscheinlichkeit des Risikos und des Schadensausmaßes des Risikos identifizieren

FL-5.2.2 (K2) ... zwischen Projektrisiken und Produktrisiken unterscheiden

FL-5.2.3 (K2) ... den möglichen Einfluss der Produktrisikoanalyse auf Intensität und Umfang des Testens erklären

FL-5.2.4 (K2) ... mögliche Maßnahmen, die als Reaktion auf analysierte Produktrisiken ergriffen werden können, erklären

5.3 Testüberwachung, Teststeuerung und Testabschluss

FL-5.3.1 (K1) ... die beim Testen verwendeten Metriken wiedergeben

FL-5.3.2 (K2) ... Zweck, Inhalt und Zielgruppen von Testberichten zusammenfassen

FL-5.3.3 (K2) ... Beispiele geben, wie man den Teststatus kommunizieren kann

5.4 Konfigurationsmanagement

FL-5.4.1 (K2) ... mögliche Unterstützung des Testens durch das Konfigurationsmanagement zusammenfassen

5.5 Fehlermanagement

FL-5.5.1 (K3) ... einen Fehlerbericht erstellen

5.1 Testplanung

5.1.1 Zweck und Inhalt eines Testkonzepts

Ein Testkonzept beschreibt die Ziele, Ressourcen und Prozesse für ein Testprojekt. Ein Testkonzept:

- dokumentiert die Mittel und den Zeitplan zur Erreichung der Testziele,
- hilft sicherzustellen, dass die durchgeführten Testaktivitäten die festgelegten Kriterien erfüllen,
- dient als Mittel zur Kommunikation mit Teammitgliedern und anderen Stakeholdern,
- zeigt, dass sich das Testen an die bestehende Testrichtlinie und Teststrategie hält (oder erklärt, warum das Testen davon abweicht).

Die Testplanung gibt den Testern Denkanstöße und zwingt sie, sich mit den zukünftigen Herausforderungen in Bezug auf Risiken, Zeitpläne, Mitarbeiter, Werkzeuge, Kosten, Aufwand usw. auseinanderzusetzen. Der Prozess der Erstellung eines Testkonzepts ist eine nützliche Vorgehensweise, um die Maßnahmen zu überdenken, die zum Erreichen der Testziele des Projekts erforderlich sind.

Zu den typischen Inhalten eines Testkonzepts gehören:

- Kontext des Testens (z. B. Umfang, Testziele, Einschränkungen, Testbasis)
- Annahmen und Einschränkungen des Testprojekts
- Stakeholder (z. B. Rollen, Verantwortlichkeiten, Relevanz für das Testen, Einstellung und Schulungsbedarf)
- Kommunikation (z. B. Formen und Häufigkeit der Kommunikation, Dokumentationsvorlagen)
- Risikoverzeichnis (z. B. Produktrisiken, Projektrisiken)
- Testansatz (z. B. Teststufen, Testarten, Testverfahren, Testliefergegenstände, Eingangskriterien und Endekriterien, Unabhängigkeit des Testens, zu erhebende Metriken, Anforderungen an Testdaten, Anforderungen an die Testumgebung, Abweichungen von der organisationsweiten Testrichtlinie und Teststrategie)
- Budget und Zeitplan

Weitere Einzelheiten über das Testkonzept und seinen Inhalt sind in der Norm ISO/IEC/IEEE 29119-3 zu finden.

5.1.2 Der Beitrag des Testers zur Iterations- und Releaseplanung

In iterativen SDLCs gibt es typischerweise zwei Arten von Planung: Releaseplanung und Iterationsplanung.

Die Releaseplanung sieht die Bereitstellung eines Produkts vor, definiert das Produkt-Backlog bzw. passt es an und kann die Verfeinerung größerer User Storys in eine Reihe kleinerer User Storys beinhalten. Sie dient auch als Grundlage für den Testansatz und das Testkonzept über alle Iterationen. Tester, die an der Releaseplanung mitwirken, beteiligen sich an der Erstellung testbarer User Storys und Abnahmekriterien (siehe Abschnitt 4.5), beteiligen sich an Projekt- und Qualitätsrisikoanalysen (siehe Abschnitt 5.2), schätzen den mit den User Storys verbundenen Testaufwand (siehe Abschnitt 5.1.4), legen den Testansatz fest und planen die Tests für das Release.

Die Iterationsplanung sieht das Ende einer einzelnen Iteration voraus und befasst sich mit dem Iterations-Backlog. Die an der Iterationsplanung beteiligten Tester nehmen an der detaillierten Risikoanalyse der User Storys teil, bestimmen die Testbarkeit der User Storys, zerlegen die User Storys in Aufgaben (insbesondere Testaufgaben), schätzen den Testaufwand für alle Testaufgaben und identifizieren und verfeinern die funktionalen und nicht-funktionalen Aspekte des Testobjekts.

5.1.3 Eingangskriterien und Endekriterien

Eingangskriterien definieren die Vorbedingungen für die Durchführung einer bestimmten Aktivität. Wenn die Eingangskriterien nicht erfüllt sind, ist es wahrscheinlich, dass sich die Aktivität als schwieriger, zeitaufwendiger, kostspieliger und risikoreicher erweist. Die Endekriterien legen fest, was erreicht werden muss, um eine Aktivität für abgeschlossen zu erklären. Eingangskriterien und Endekriterien sollten für jede Teststufe definiert werden und unterscheiden sich je nach den Testzielen.

Typische Eingangskriterien sind: Verfügbarkeit von Ressourcen (z. B. Menschen, Werkzeuge, Umgebungen, Testdaten, Budget, Zeit), Verfügbarkeit von Testmitteln (z. B. Testbasis, testbare Anforderungen, User Storys, Testfälle) und die anfängliche Qualität eines Testobjekts (z. B. alle Smoke-Tests wurden bestanden).

Typische Endekriterien sind: Messungen der Gründlichkeit (z. B. erreichter Überdeckungsgrad, Anzahl der ungelösten Fehlerzustände, Fehlerdichte, Anzahl der fehlgeschlagenen Testfälle) und Testabschlusskriterien (z. B. geplante Tests wurden ausgeführt, statische Tests wurden ausgeführt, alle gefundenen Fehlerzustände werden berichtet, alle Regressionstests sind automatisiert).

Auch aufgebrauchte Zeit oder Budget, kann als gültige Endekriterien betrachtet werden. Auch wenn keine anderen Abnahmekriterien erfüllt sind, kann es akzeptabel sein, das Testen unter solchen Umständen zu beenden, wenn die Stakeholder das Risiko, ohne weitere Tests in Betrieb zu gehen, geprüft und akzeptiert haben.

In der agilen Softwareentwicklung werden die Endekriterien oft als Definition-of-Done bezeichnet, die die objektiven Metriken des Teams für ein freizugebendes Element definieren. Eingangskriterien, die eine User Story erfüllen muss, um mit der Entwicklung und/oder dem Testen zu beginnen, werden als Definition-of-Ready bezeichnet.

5.1.4 Schätzverfahren

Bei der Schätzung des Testaufwands geht es um die Vorhersage des Umfangs der testbezogenen Arbeit, die erforderlich ist, um die Ziele eines Testprojekts zu erreichen. Es ist wichtig, den Stakeholdern klarzumachen, dass die Schätzung auf einer Reihe von Annahmen beruht und immer mit Schätzfehlern behaftet ist. Die Schätzung für kleine Aufgaben ist in der Regel genauer als für große Aufgaben. Bei der Schätzung einer großen Aufgabe kann diese daher in eine Reihe kleinerer Aufgaben zerlegt werden, die dann ihrerseits geschätzt werden können.

In diesem Lehrplan werden die folgenden vier Verfahren zur Schätzung beschrieben.

Schätzung basierend auf Verhältniszahlen: Bei diesem metrikbasierten Verfahren werden Zahlen aus früheren Projekten innerhalb des Unternehmens gesammelt, was die Ableitung von "Standard"-Verhältniszahlen für ähnliche Projekte ermöglicht. Die Kennzahlen der eigenen Projekte eines Unternehmens (z. B. aus historischen Daten) sind im Allgemeinen die beste Quelle für den Schätzprozess. Diese Standard-Verhältniszahlen können dann zur Schätzung des Testaufwands für das neue Projekt verwendet werden. Wenn beispielsweise im vorherigen Projekt das Verhältnis von Entwicklungs- zu Testaufwand 3:2 war und im aktuellen Projekt ein Entwicklungsaufwand von 600 Personentagen erwartet wird, kann der Testaufwand auf 400 Personentage geschätzt werden.

Extrapolation: Bei diesem auf Metriken basierenden Verfahren werden Messungen so früh wie möglich im laufenden Projekt durchgeführt, um die Daten zu sammeln. Wenn genügend Beobachtungen vorliegen, kann der für die verbleibende Arbeit erforderliche Aufwand durch Extrapolation dieser Daten (in der Regel durch Anwendung eines mathematischen Modells) angenähert werden. Diese Methode eignet sich sehr gut für iterative SDLCs. Zum Beispiel kann das Team den Testaufwand in der nächsten Iteration als den durchschnittlichen Aufwand der letzten drei Iterationen extrapolieren.

Breitband-Delphi: Bei diesem iterativen, expertenbasierten Verfahren nehmen die Experten erfahrungsbasierte Schätzungen vor. Jeder Experte schätzt für sich allein den Aufwand. Die Ergebnisse werden gesammelt, und wenn es Abweichungen gibt, die außerhalb der vereinbarten Grenzen liegen, diskutieren die Experten ihre aktuellen Schätzungen. Jeder Experte wird dann gebeten, auf der Grundlage dieser Rückmeldungen eine neue Schätzung vorzunehmen, wiederum für sich allein. Dieser Prozess wird so lange wiederholt, bis ein Konsens erreicht ist. Planungspoker ist eine Variante von Breitband-Delphi, die häufig in der agilen Softwareentwicklung eingesetzt wird. Beim Planungspoker werden Schätzungen in der Regel mithilfe von Karten mit Zahlen vorgenommen, die die Höhe des Aufwands darstellen.

Drei-Punkt-Schätzung: Bei diesem expertenbasierten Verfahren werden drei Schätzungen von den Experten vorgenommen: die optimistischste Schätzung (a), die wahrscheinlichste Schätzung (m) und die pessimistischste Schätzung (b). Die finale Schätzung (E) ist ihr gewichtetes arithmetisches Mittel. In der am weitesten verbreiteten Version diesem Verfahren wird die Schätzung wie folgt berechnet: E = (a + 4*m + b) / 6. Der Vorteil dieses Verfahrens besteht darin, dass sie es den Experten ermöglicht, den Schätzfehler (Standardabweichung) zu berechnen: SD = (b - a) / 6. Wenn zum Beispiel die Schätzungen (in Personenstunden) a=6, m=9 und b=18 sind, dann liegt die endgültige Schätzung bei 10±2 Personenstunden

(d. h. zwischen 8 und 12 Personenstunden), weil E = (6 + 4*9 + 18) / 6 = 10 und SD = (18 - 6) / 6 = 2.

Siehe (Kan 2003, Koomen 2006, Westfall 2009) für diese und viele andere Testschätzverfahren.

5.1.5 Priorisierung von Testfällen

Sobald die Testfälle und Testabläufe spezifiziert und zu Testsuiten zusammengestellt sind, können diese Testsuiten in einem Testausführungsplan angeordnet werden, der die Reihenfolge ihrer Ausführung festlegt. Bei der Priorisierung von Testfällen können verschiedene Faktoren in Betracht gezogen werden. Die am häufigsten verwendeten Strategien zur Priorisierung von Testfällen sind die folgenden:

- **Risikobasierte Priorisierung**, bei der sich die Reihenfolge der Testdurchführung nach den Ergebnissen der Risikoanalyse richtet (siehe Abschnitt 5.2.3). Testfälle, die die wichtigsten Risiken überdecken, werden zuerst ausgeführt.
- **Überdeckungsbasierte Priorisierung**, bei der sich die Reihenfolge der Testdurchführung nach der Überdeckung richtet (z. B. Anweisungsüberdeckung). Testfälle, die die höchste Überdeckung erreichen, werden zuerst ausgeführt. Bei einer anderen Variante, der Priorisierung von zusätzlicher Überdeckung, wird der Testfall mit der höchsten Überdeckung zuerst ausgeführt; jeder nachfolgende Testfall ist derjenige, der die höchste zusätzliche Überdeckung erreicht.
- **Anforderungsbasierte Priorisierung**, bei der sich die Reihenfolge der Testdurchführung nach den Prioritäten der Anforderungen richtet, die auf die entsprechenden Testfälle übertragen werden. Die Prioritäten der Anforderungen werden von den Stakeholdern festgelegt. Testfälle, die sich auf die wichtigsten Anforderungen beziehen, werden zuerst ausgeführt.

Im Idealfall werden die Testfälle in der Reihenfolge ihrer Prioritäten ausgeführt, z. B. mit Hilfe einer der oben genannten Priorisierungsstrategien. Diese Praktik funktioniert jedoch möglicherweise nicht, wenn die Testfälle oder die zu testenden Features Abhängigkeiten aufweisen. Wenn ein Testfall mit einer höheren Priorität von einem Testfall mit einer niedrigeren Priorität abhängt, muss der Testfall mit der niedrigeren Priorität zuerst ausgeführt werden.

Bei der Reihenfolge der Testdurchführung muss auch die Verfügbarkeit von Ressourcen berücksichtigt werden. Zum Beispiel die erforderlichen Testwerkzeuge, Testumgebungen oder Personen, die möglicherweise nur für ein bestimmtes Zeitfenster zur Verfügung stehen.

5.1.6 Testpyramide

Die Testpyramide ist ein Modell, das zeigt, dass verschiedene Tests eine unterschiedliche Granularität haben können. Das Testpyramiden-Modell unterstützt das Team bei der Testautomatisierung und bei der Verteilung des Testaufwands, indem es zeigt, dass verschiedene Ziele durch verschiedene Stufen der Testautomatisierung unterstützt werden. Die Ebenen der Pyramide stellen Gruppen von Tests dar. Je höher die Ebene, desto geringer

ist die Testgranularität, das isolierte Testen und die Testdurchführungszeit der gesamten Tests der Ebene. Tests in der untersten Schicht sind klein, isoliert, schnell und prüfen einen kleinen Teil der Funktionalität, so dass normalerweise viele von ihnen benötigt werden, um eine angemessene Überdeckung zu erreichen. Die oberste Schicht repräsentiert komplexe, High-Level End-to-End-Tests. Diese High-Level Tests sind in der Regel langsamer als die Tests aus den unteren Schichten und prüfen in der Regel einen großen Teil der Funktionalität, so dass in der Regel nur wenige von ihnen erforderlich sind, um eine angemessene Überdeckung zu erreichen. Die Anzahl und Benennung der Schichten können variieren. Das ursprüngliche Testpyramiden-Modell (Cohn 2009) definiert zum Beispiel drei Schichten: "Unit-Tests", "Service-Tests" und "UI-Tests". Ein anderes beliebtes Modell definiert Unittests (Komponententests), Integrationstests (Komponentenintegrationstests) und End-To-End-Tests. Andere Teststufen (siehe Abschnitt 2.2.1) können ebenfalls verwendet werden.

5.1.7 Testquadranten

Die von Brian Marick definierten Testquadranten (Marick 2003, Crispin 2008) gruppieren die Teststufen mit den entsprechenden Testarten, Aktivitäten, Testverfahren und Arbeitsergebnissen in der agilen Softwareentwicklung. Das Modell unterstützt das Testmanagement dabei, diese zu visualisieren, um sicherzustellen, dass alle geeigneten Testarten und Teststufen in den SDLC einbezogen werden, und um zu verstehen, dass einige Testarten für bestimmte Teststufen relevanter sind als für andere. Dieses Modell bietet auch eine Möglichkeit, die Testarten zu differenzieren und allen Stakeholdern, einschließlich Entwicklern, Testern und Fachbereichsvertretern, zu beschreiben.

In diesem Modell können Tests sowohl geschäftsorientiert oder technologieorientiert sein. Tests können andererseits das Team unterstützen (d. h. die Entwicklung anleiten) oder das Produkt kritisch betrachten (d. h. sein Verhalten anhand der Erwartungen messen). Die Kombination dieser beiden Gesichtspunkte bestimmt die vier Quadranten:

- Quadrant Q1 (technologieorientiert, Unterstützung des Teams). Dieser Quadrant enthält Komponenten- und Komponentenintegrationstests. Diese Tests sollten automatisiert und in den CI-Prozess einbezogen werden.
- Quadrant Q2 (geschäftlich orientiert, Unterstützung des Teams). Dieser Quadrant enthält funktionale Tests, Beispiele, User Story-basierte Tests, Prototypen für Benutzererfahrung, API-Tests und Simulationen. Diese Tests prüfen die Abnahmekriterien und können manuell oder automatisiert sein.
- Quadrant Q3 (geschäftlich orientiert, kritische Betrachtung des Produkts). Dieser Quadrant enthält explorative Tests, Gebrauchstauglichkeitstests und Benutzerabnahmetests. Diese Tests sind benutzerorientiert und häufig manuell.
- Quadrant Q4 (technologieorientiert, kritische Betrachtung des Produkts). Dieser Quadrant enthält Smoke-Tests und nicht-funktionale Tests (außer Gebrauchstauglichkeitstests). Diese Tests sind häufig automatisiert.

5.2 Risikomanagement

Organisationen sind mit vielen internen und externen Faktoren konfrontiert, die es unsicher machen, ob und wann sie ihre Ziele erreichen werden (ISO 31000). Durch Risikomanagement können Organisationen die Wahrscheinlichkeiten der Zielerreichung erhöhen, die Qualität ihrer Produkte verbessern und das Vertrauen der Stakeholder stärken.

Die wichtigsten Aktivitäten des Risikomanagements sind:

- Risikoanalyse (bestehend aus Risikoidentifizierung und Risikobewertung; siehe Abschnitt 5.2.3)
- Risikosteuerung (bestehend aus Risikominderung und Risikoüberwachung; siehe Abschnitt 5.2.4)

Der Testansatz, bei dem Testaktivitäten auf der Grundlage von Risikoanalyse und Risikosteuerung ausgewählt, priorisiert und gesteuert werden, wird als risikobasierter Test bezeichnet.

5.2.1 Risikodefinition und Risikoattribute

Ein Risiko ist ein potenzielles Ereignis, eine Gefahr, eine Bedrohung oder eine Situation, deren Eintreten eine nachteilige Auswirkung verursacht. Ein Risiko kann durch zwei Faktoren charakterisiert werden:

- Eintrittswahrscheinlichkeit des Risikos – die Wahrscheinlichkeit des Eintretens des Risikos (größer als Null und kleiner als Eins)
- Schadensausmaß des Risikos (Schaden) – die Folgen des Eintretens

Diese beiden Faktoren drücken die Risikostufe aus, die ein Maß für das Risiko ist. Je höher die Risikostufe, desto wichtiger ist die Behandlung des Risikos.

5.2.2 Projektrisiken und Produktrisiken

Beim Testen von Software hat man es im Allgemeinen mit zwei Arten von Risiken zu tun: Projektrisiken und Produktrisiken.

Projektrisiken beziehen sich auf das Management und die Steuerung des Projekts. Zu den Projektrisiken gehören:

- Organisatorische Probleme (z. B. Verzögerungen bei der Lieferung von Arbeitsergebnissen, ungenaue Schätzungen, Kostenkürzungen)
- Personelle Aspekte (z. B. unzureichende Fähigkeiten, Konflikte, Kommunikationsprobleme, Personalmangel)
- Technische Probleme (z. B. schleichende Ausweitung des Projektumfangs (scope creep), unzureichende Werkzeugunterstützung)
- Lieferantenprobleme (z. B. Lieferausfall von Drittanbietern, Konkurs des unterstützenden Unternehmens)

Eingetretene Projektrisiken können Auswirkungen auf den Zeitplan, das Budget und/oder den Umfang des Projekts haben, was die Erreichung der Projektziele beeinträchtigt.

Produktrisiken stehen im Zusammenhang mit den Qualitätsmerkmalen des Produkts (z. B. beschrieben im Qualitätsmodell der ISO 25010). Beispiele für Produktrisiken sind: fehlende oder falsche Funktionalität, falsche Berechnungen, Laufzeitfehler, leistungsschwache Architektur, ineffiziente Algorithmen, unzureichende Reaktionszeit, schlechte Benutzererfahrung, Sicherheitsschwachstellen. Produktrisiken können, wenn sie eintreten, verschiedene negative Folgen nach sich ziehen, darunter:

- Unzufriedenheit der Benutzer
- Verlust von Einnahmen, Vertrauen und Ansehen
- Schaden für Dritte
- Hohe Wartungskosten, Überlastung des Helpdesks
- Strafrechtliche Sanktionen
- In extremen Fällen körperliche Schäden, Verletzungen oder sogar Tod

5.2.3 Produktrisikoanalyse

Aus der Sicht des Testens besteht das Ziel der Produktrisikoanalyse darin, ein Bewusstsein für das Produktrisiko zu schaffen, um den Testaufwand so zu fokussieren, dass die verbleibende Risikostufe des Produkts minimiert wird. Im Idealfall beginnt die Risikoanalyse des Produktrisikos bereits in einem frühen Stadium des SDLC.

Die Produktrisikoanalyse besteht aus Risikoidentifizierung und Risikobewertung. Bei der Risikoidentifizierung geht es um die Erstellung einer umfassenden Liste von Risiken. Die Stakeholder können Risiken mit Hilfe verschiedener Verfahren und Werkzeuge identifizieren, z. B. durch Brainstorming, Workshops, Interviews oder Ursache-Wirkungs-Diagramme. Die Risikobewertung umfasst die Kategorisierung der identifizierten Risiken, die Bestimmung ihrer Eintrittswahrscheinlichkeit, des Schadensausmaßes und der Risikostufe, Priorisierung und Vorschläge für den Umgang mit den Risiken. Die Kategorisierung hilft bei der Zuweisung von Maßnahmen zur Risikominderung, da Risiken, die in dieselbe Kategorie fallen, in der Regel mit einem ähnlichen Ansatz gemindert werden können.

Bei der Risikobewertung kann ein quantitativer oder qualitativer Ansatz oder eine Mischung aus beidem verwendet werden. Beim quantitativen Ansatz wird die Risikostufe als Multiplikation von Eintrittswahrscheinlichkeit des Risikos und Schadensausmaß des Risikos berechnet. Beim qualitativen Ansatz kann die Risikostufe anhand einer Risikomatrix bestimmt werden.

Die Risikoanalyse des Produktrisikos kann die Intensität und den Umfang der Tests beeinflussen. Ihre Ergebnisse werden verwendet, um

- den Umfang der durchzuführenden Tests zu bestimmen,
- die einzelnen Teststufen zu bestimmen und Testarten vorzuschlagen, die durchgeführt werden sollen,

- die einzusetzenden Testverfahren und die zu erreichende Überdeckung festzulegen,
- den Testaufwand für jede Aufgabe zu schätzen,
- Tests zu priorisieren, um die kritischen Fehlerzustände so früh wie möglich zu finden,
- festzustellen, ob neben dem Testen weitere Aktivitäten zur Verringerung des Risikos eingesetzt werden können.

5.2.4 Produktrisikosteuerung

Die Steuerung von Produktrisiken umfasst alle Maßnahmen, die als Reaktion auf identifizierte und bewertete Produktrisiken ergriffen werden. Die Produktrisikosteuerung besteht aus Risikominderung und Risikoüberwachung. Bei der Risikominderung geht es darum, die in der Risikobewertung vorgeschlagenen Maßnahmen zur Verringerung der Risikostufe umzusetzen. Ziel der Risikoüberwachung ist es, die Effektivität der Maßnahmen zur Risikominderung zu gewährleisten, weitere Informationen zur Verbesserung der Risikobewertung zu erhalten und neu auftretende Risiken zu erkennen.

Was die Steuerung von Produktrisiken betrifft, so sind nach der Analyse eines Risikos mehrere Reaktionsmöglichkeiten auf das Risiko möglich, z. B. Risikominderung durch Testen, Risikoakzeptanz, Risikotransfer oder Notfallplan (Veenendaal 2012). Folgende Maßnahmen können ergriffen werden, um die Produktrisiken durch Testen zu mindern:

- Auswahl von Testern mit dem richtigen Maß an Erfahrung und Fähigkeiten, die für einen bestimmten Risikotyp geeignet sind
- Anwendung einer geeigneten Stufe an unabhängigem Test
- Durchführung von Reviews und statischen Analysen
- Anwendung geeigneter Testverfahren und Überdeckungsgrade
- Anwendung geeigneter Testarten, für die betroffenen Qualitätsmerkmale
- Durchführung dynamischer Tests, einschließlich Regressionstests

5.3 Testüberwachung, Teststeuerung und Testabschluss

Bei der Testüberwachung geht es um das Sammeln von Informationen über das Testen. Diese Informationen werden verwendet, um den Testfortschritt zu bewerten und zu messen, ob die Testendekriterien oder die mit den Endekriterien verbundenen Testaufgaben erfüllt sind, wie z. B. die Erfüllung der Ziele für die Überdeckung von Produktrisiken, Anforderungen oder Abnahmekriterien.

Die Teststeuerung nutzt die Informationen aus der Testüberwachung, um in Form von Steuerungsmaßnahmen Anleitungen und notwendige Korrekturmaßnahmen zu geben, um das Testen so effektiv und effizient wie möglich zu gestalten. Beispiele für Steuerungsmaßnahmen sind:

- Neupriorisierung von Tests, wenn ein identifiziertes Risiko zu einem Problem wird
- Neubewertung, ob ein Testelement die Eingangskriterien oder die Endekriterien nach einer Überarbeitung erfüllt
- Anpassung des Testzeitplans, um eine Verzögerung bei der Bereitstellung der Testumgebung auszugleichen
- Hinzufügen neuer Ressourcen, bei Bedarf

Beim Testabschluss werden Daten aus abgeschlossenen Testaktivitäten gesammelt, um Erfahrungen, Testmittel und andere relevante Informationen zu konsolidieren. Testabschlussaktivitäten finden an Projektmeilensteinen statt, z. B. wenn eine Teststufe abgeschlossen ist, eine agile Iteration beendet ist, ein Testprojekt abgeschlossen (oder abgebrochen) ist, ein Softwaresystem freigegeben oder ein Wartungsrelease abgeschlossen ist.

5.3.1 Beim Testen verwendete Metriken

Testmetriken werden erfasst, um den Fortschritt gegenüber dem definierten Zeit- und Budgetplan, die aktuelle Qualität des Testobjekts und die Effektivität der Testaktivitäten in Bezug auf die Test- oder ein Iterationsziel aufzuzeigen. Die Testüberwachung erfasst eine Vielzahl von Metriken zur Unterstützung der Teststeuerung und des Testabschlusses.

Zu den gängigen Testmetriken gehören:

- Metriken zum Projektfortschritt (z. B. abgeschlossene Aufgaben, Ressourcenverbrauch, Testaufwand)
- Metriken zum Testfortschritt (z. B. Fortschritt bei der Testfallrealisierung, Fortschritt bei der Vorbereitung der Testumgebung, Anzahl der ausgeführten/nicht ausgeführten Testfälle, bestandene/nicht bestandene Testfälle, Zeit für die Testdurchführung)
- Metriken zur Produktqualität (z. B. Verfügbarkeit, Reaktionszeit, mittlere Betriebsdauer bis zum Ausfall (meantime to failure, MTTF))
- Metriken für Fehlerzustände (z. B. Anzahl und Prioritäten der gefundenen/behobenen Fehlerzustände, Fehlerdichte, Fehlerfindungsrate (defect detection percentage, DDP))
- Metriken zu Risiken (z. B. verbleibende Risikostufe)
- Metriken zur Überdeckung (z. B. Anforderungsüberdeckung, Codeüberdeckung)
- Metriken zu den Kosten (z. B. Kosten für das Testen, organisatorische Kosten für Qualität)

5.3.2 Zweck, Inhalt und Zielgruppen für Testberichte

Der Zweck eines Testberichts ist es, Informationen über die Testaktivitäten zusammenzufassen und zu kommunizieren, sowohl während als auch am Ende einer Testaktivität. Testfortschrittsberichte unterstützen die laufende Teststeuerung und müssen genügend Informationen liefern, um Änderungen am Testzeitplan, an den Ressourcen oder

am Testkonzept vorzunehmen, wenn solche Änderungen aufgrund von Abweichungen vom Plan oder veränderten Umständen erforderlich sind. Testabschlussberichte fassen eine bestimmte Phase des Testens zusammen (z. B. Teststufe, Testzyklus, Iteration) und können Informationen für nachfolgende Tests liefern.

Während der Testüberwachung und -steuerung erstellt das Testteam Testfortschrittsberichte für die Stakeholder, um sie auf dem Laufenden zu halten. Testfortschrittsberichte werden in der Regel in regelmäßigen Abständen (z. B. täglich, wöchentlich usw.) erstellt und enthalten typischerweise:

- Testzeitraum
- Testfortschritt (z. B. vor oder hinter dem Zeitplan), einschließlich aller festgestellten Abweichungen
- Hindernisse für das Testen und deren Umgehungsmöglichkeiten
- Testmetriken (siehe Abschnitt 5.3.1 für Beispiele)
- Neue und veränderte Risiken innerhalb des Testzeitraumes
- Geplante Tests für den nächsten Zeitraum

Ein Testabschlussbericht wird während des Testabschlusses erstellt, wenn ein Projekt, eine Teststufe oder eine Testart abgeschlossen ist und im Idealfall die Endekriterien erfüllt wurden. Er basiert auf Testfortschrittsberichten und anderen Daten. Typische Testabschlussberichte beinhalten:

- Zusammenfassung der durchgeführten Tests
- Bewertung der Tests und der Qualität des Produkts auf der Grundlage des ursprünglichen Testkonzepts (d. h. Testziele und Endekriterien)
- Abweichungen vom Testkonzept (z. B. Abweichungen vom geplanten Zeitplan, von der Dauer und vom Aufwand).
- Hindernisse beim Testen und Umgehungen
- Testmetriken auf der Grundlage von Testfortschrittsberichten
- Restrisiken, nicht behobene Fehlerzustände
- Lessons Learned, die für das Testen relevant sind

Unterschiedliche Zielgruppen erfordern unterschiedliche Informationen in den Berichten und beeinflussen den Grad der Formalität und die Häufigkeit der Berichterstattung. Die Berichterstattung über den Testfortschritt an andere Mitglieder desselben Teams erfolgt häufig und informell, während die Berichterstattung über das Testen für ein abgeschlossenes Projekt einer festgelegten Vorlage folgt und nur einmal erfolgt.

Die Norm ISO/IEC/IEEE 29119-3 enthält Vorlagen und Beispiele für Testfortschrittsberichte (genannt Teststatusberichte) und Testabschlussberichte.

5.3.3 Kommunikation des Teststatus

Die beste Art der Kommunikation des Teststatus hängt von den Bedürfnissen des Testmanagements, der organisationsweiten Teststrategie, den regulatorischen Vorgaben oder, im Falle von selbstorganisierten Teams (siehe Abschnitt 1.5.2), vom Team selbst ab. Zu den Optionen gehören:

- Mündliche Kommunikation mit Teammitgliedern und anderen Stakeholdern
- Dashboards (z. B. CI/CD-Dashboards, Taskboards und Burn-Down-Charts)
- Elektronische Kommunikationskanäle (z. B. E-Mail, Chat)
- Online-Dokumentation
- Formale Testberichte (siehe Abschnitt 5.3.2)

Eine oder mehrere dieser Optionen können verwendet werden. Eine formellere Kommunikation kann sich für verteilte Teams anbieten, in denen eine direkte Kommunikation von Angesicht zu Angesicht aufgrund von geografischen Entfernungen oder Zeitunterschieden nicht immer möglich ist. In der Regel sind verschiedene Stakeholder an unterschiedlichen Arten von Informationen interessiert, so dass die Kommunikation entsprechend angepasst werden sollte.

5.4 Konfigurationsmanagement

Beim Testen stellt das Konfigurationsmanagement (KM) eine Disziplin zur Identifizierung, Steuerung und Verfolgung von Arbeitsergebnissen wie Testkonzepten, Teststrategien, Testbedingungen, Testfällen, Testskripten, Testergebnissen, Testprotokollen und Testberichten als Konfigurationselemente dar.

Für ein komplexes Konfigurationselement (z. B. eine Testumgebung) zeichnet das KM die Elemente, aus denen es besteht, ihre Beziehungen und Versionen auf. Wenn das Konfigurationselement zum Testen freigegeben wird, wird es zur Baseline und kann nur durch einen formalen Änderungskontrollprozess geändert werden.

Das Konfigurationsmanagement sichert die Daten eines Konfigurationselements, wenn eine neue Baseline erstellt wird. Es ist möglich, zu einer früheren Baseline zurückzukehren, um frühere Testergebnisse zu reproduzieren.

Um das Testen richtig zu unterstützen, stellt das KM folgendes sicher:

- Alle Konfigurationselemente, einschließlich der Testelemente (einzelne Teile des Testobjekts), werden eindeutig identifiziert, versionskontrolliert, auf Änderungen hin verfolgt und mit anderen Konfigurationselementen in Beziehung gesetzt, damit die Verfolgbarkeit während des gesamten Testprozesses aufrechterhalten werden kann.
- Alle identifizierten Elemente der Dokumentation und Software werden in der Testdokumentation eindeutig referenziert.

Kontinuierliche Integration, kontinuierliche Auslieferung, kontinuierliche Bereitstellung und die damit verbundenen Tests werden in der Regel als Teil einer automatisierten DevOps-Auslieferungskette implementiert (siehe Abschnitt 2.1.4), zu der normalerweise auch automatisiertes KM gehört.

5.5 Fehlermanagement

Da eines der wichtigsten Testziele darin besteht, Fehlerzustände zu finden, ist ein etablierter Prozess für das Fehlermanagement unerlässlich. Obwohl wir hier von "Fehlern" sprechen, können sich die gemeldeten Anomalien als tatsächliche Fehlerzustände oder als etwas anderes herausstellen (z. B. falsch positives Ergebnis, Änderungsanforderung) – dies wird während der Bearbeitung der Fehlerberichte geklärt. Anomalien können in jeder Phase des SDLC gemeldet werden und die Form hängt vom SDLC ab. Der Prozess des Fehlermanagements umfasst mindestens einen Arbeitsablauf für die Behandlung einzelner Anomalien von ihrer Entdeckung bis zu ihrer Schließung sowie Regeln für ihre Klassifizierung. Der Workflow umfasst typischerweise Aktivitäten zur Protokollierung der gemeldeten Anomalien, zu deren Analyse und Klassifizierung, zur Entscheidung über eine geeignete Reaktion, z. B. Behebung oder Beibehaltung des Fehlerzustands, und letztendlich zur Schließung des Fehlerberichts. Der Prozess muss von allen Beteiligten befolgt werden. Es ist ratsam, Fehlerzustände aus statischen Tests (insbesondere der statischen Analyse) auf ähnliche Weise zu behandeln.

Typische Fehlerberichte haben die folgenden Ziele:

- Bereitstellung ausreichender Informationen für diejenigen, die für die Bearbeitung und Behebung gemeldeter Fehlerzustände verantwortlich sind, um das Problem zu lösen
- Verfolgung der Qualität des Arbeitsergebnisses
- Bereitstellung von Ideen zur Verbesserung des Entwicklungs- und Testprozesses

Ein Fehlerbericht, der während des dynamischen Testens aufgezeichnet wird, enthält in der Regel folgende Angaben:

- Eindeutige Kennung
- Titel mit einer kurzen Zusammenfassung der Anomalie
- Datum, an dem die Anomalie beobachtet wurde, ausstellende Organisation und Autor, einschließlich seiner Rolle
- Identifikation des Testobjekts und der Testumgebung
- Kontext des Fehlerzustands (z. B. laufender Testfall, durchgeführte Testaktivität, SDLC-Phase und andere relevante Informationen wie verwendete Testverfahren, Checklisten oder Testdaten)
- Beschreibung der Fehlerwirkung, um eine Reproduktion und Behebung zu ermöglichen, einschließlich der Schritte, die die Anomalie aufgedeckt haben, sowie relevante Testprotokolle, Datenbankauszüge, Screenshots oder Aufzeichnungen
- Erwartete Ergebnisse und tatsächliche Ergebnisse

- Schweregrad (Grad der Auswirkung) der Fehlerwirkung auf die Interessen der Stakeholder oder Anforderungen
- Priorität für die Behebung
- Fehlerstatus (z. B. offen, zurückgestellt, doppelt, auf Behebung wartend, auf Fehlernachtest wartend, wiedereröffnet, geschlossen, zurückgewiesen)
- Referenzen (z. B. auf den Testfall)

Einige dieser Daten können bei der Verwendung von Fehlermanagementwerkzeugen automatisch eingefügt werden (z. B. Kennung, Datum, Autor und Anfangsstatus). Dokumentvorlagen für einen Fehlerbericht und ein beispielhafter Fehlerbericht finden sich in der Norm ISO/IEC/IEEE 29119-3, die Fehlerberichte als Abweichungsberichte bezeichnet.

6. Testwerkzeuge – 20 Minuten

Schlüsselbegriffe

Testautomatisierung

Lernziele für Kapitel 6: **Der Lernende kann ...**

6.1 Werkzeugunterstützung für das Testen

FL-6.1.1 (K2) ... mögliche Unterstützung des Testens durch verschiedene Arten von Testwerkzeugen erklären

6.2 Nutzen und Risiken von Testautomatisierung

FL-6.2.1 (K1) ... die Nutzen und Risiken von Testautomatisierung wiedergeben

6.1 Werkzeugunterstützung für das Testen

Testwerkzeuge unterstützen und erleichtern viele Testaktivitäten. Beispiele hierfür sind unter anderem:

- Managementwerkzeuge – erhöhen die Effizienz des Testprozesses, indem sie das Management des SDLC, der Anforderungen, der Tests, der Fehlerzustände und der Konfiguration erleichtern
- Werkzeuge für statische Tests – unterstützen den Tester bei der Durchführung von Reviews und statischen Analysen
- Werkzeuge für Testentwurf und -realisierung – erleichtern die Erstellung von Testfällen, Testdaten und Testabläufen
- Werkzeuge zur Testdurchführung und -überdeckung – erleichtern die automatisierte Testdurchführung und die Messung der Überdeckung
- Werkzeuge für nicht-funktionale Tests – ermöglichen dem Tester die Durchführung nicht-funktionaler Tests, die manuell nur schwer oder gar nicht durchführbar sind
- DevOps-Werkzeuge – unterstützen die DevOps-Auslieferungskette, die Verfolgung von Arbeitsabläufen, den automatisierten Build-Prozess, CI/CD
- Werkzeuge für die Zusammenarbeit – erleichtern die Kommunikation
- Werkzeuge zur Unterstützung der Skalierbarkeit und Standardisierung der Bereitstellung (z. B. virtuelle Maschinen, Container-Tools)
- Jedes andere Werkzeug, das beim Testen hilft (z. B. ist ein Tabellenkalkulationsprogramm ein Testwerkzeug im Kontext des Testens)

6.2 Nutzen und Risiken von Testautomatisierung

Die bloße Anschaffung eines Werkzeugs ist keine Erfolgsgarantie. Jedes neue Werkzeug erfordert einen gewissen Aufwand, um einen echten und dauerhaften Nutzen zu erzielen (z. B. für die Einführung, Wartung und Schulung). Es gibt auch einige Risiken, die analysiert und gemindert werden müssen.

Zu den potenziellen Nutzen von Testautomatisierung gehören:

- Zeitersparnis durch Verringerung sich wiederholender manueller Arbeiten (z. B. Ausführung von Regressionstests, erneute Eingabe derselben Testdaten, Vergleich der erwarteten Ergebnisse mit den tatsächlichen Ergebnissen und Prüfung der Einhaltung von Programmierrichtlinien)
- Vermeidung einfacher menschlicher Fehlhandlungen durch größere Konsistenz und Wiederholbarkeit (z. B. werden Tests konsequent aus Anforderungen abgeleitet, Testdaten systematisch erstellt und Tests von einem Werkzeug in der gleichen Reihenfolge und mit der gleichen Häufigkeit ausgeführt)

- Objektivere Bewertung (z. B. Überdeckung) und Bereitstellung von Messungen, die für Menschen zu kompliziert in ihrer Herleitung sind
- Leichterer Zugang zu Informationen über das Testen zur Unterstützung des Testmanagements und der Berichterstattung (z. B. Statistiken, Diagramme und aggregierte Daten über den Testfortschritt, die Fehlerquoten und die Dauer der Testdurchführung)
- Verkürzte Testdurchführungszeiten für eine frühere Erkennung von Fehlerzuständen, schnellere Rückmeldungen und kürzere Produkteinführungszeiten
- Mehr Zeit für Tester, um neue, intensivere und effektivere Tests zu entwerfen

Zu den potenziellen Risiken von Testautomatisierung gehören:

- Unrealistische Erwartungen hinsichtlich der Vorteile eines Werkzeugs (einschließlich Funktionalität und leichte Handhabung).
- Ungenaue Schätzungen von Zeit, Kosten und Aufwand für die Einführung eines Testwerkzeugs, die Pflege von Testskripten und der Änderung des bestehenden manuellen Testprozesses.
- Verwendung eines Testwerkzeugs, wenn manuelles Testen besser geeignet ist.
- Zu starkes Vertrauen in ein Werkzeug, z. B. Vernachlässigung des Bedarfs des menschlichen kritischen Denkens.
- Die Abhängigkeit vom Werkzeuganbieter, der möglicherweise seine Geschäftstätigkeit einstellt, das Werkzeug vom Markt nimmt, das Werkzeug an einen anderen Anbieter verkauft oder schlechten Support bietet (z. B. bei Antworten auf Anfragen, bei Upgrades und der Behebung von Fehlerzuständen).
- Verwendung einer Open-Source-Software, die möglicherweise nicht mehr weiterentwickelt wird, d. h. es sind keine weiteren Updates verfügbar, oder ihre internen Komponenten müssen im Zuge der Weiterentwicklung recht häufig angepasst werden.
- Das Automatisierungswerkzeug ist nicht mit der Entwicklungsplattform kompatibel.
- Wahl eines ungeeigneten Werkzeugs, das nicht den regulatorischen Anforderungen und/oder den Sicherheitsstandards entspricht.

7. Literaturhinweise

7.1 Normen und Standards

ISO/IEC/IEEE 29119-1 (2022) Software- und Systemengineering - Software-Test - Teil 1: Konzepte und Definitionen

ISO/IEC/IEEE 29119-2 (2021) Software- und Systemengineering - Software-Test - Teil 2: Testprozesse

ISO/IEC/IEEE 29119-3 (2021) Software- und Systemengineering - Software-Test - Teil 3: Testdokumentation

ISO/IEC/IEEE 29119-4 (2021) Software- und Systemengineering - Software-Test - Teil 4: Testtechniken

ISO/IEC 25010 (2011) Software-Engineering - Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) - Qualitätsmodell und Leitlinien

ISO/IEC 20246 (2017) System und Software-Engineering - Bewertungen von Arbeitsergebnissen

ISO/IEC/IEEE 14764:2022 – Software-Engineering - Software-Lebenszyklus-Prozesse - Instandhaltung

ISO 31000 (2018) Risikomanagement - Leitlinien

7.2 Fachliteratur

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

Beizer, B. (1990) Software Testing Techniques (2e), Van Nostrand Reinhold: Boston MA

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) Systematic Software Testing, Artech House: Norwood MA

Crispin, L. and Gregory, J. (2008) Agile Testing: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL

Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley

Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley

Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

7.3 Artikel und Internetquellen

Brykczynski, B. (1999) “A survey of software inspection checklists,” *ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89*

Enders, A. (1975) “An Analysis of Errors and Their Causes in System Programs,” *IEEE Transactions on Software Engineering 1(2), pp. 140-149*

Manna, Z., Waldinger, R. (1978) “The logic of computer programming,” *IEEE Transactions on Software Engineering 4(3), pp. 199-229*

Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152–158*

Salman. I. (2016) “Cognitive biases in software quality and testing,” *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.*

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

IREB CPRE Glossar in Deutsch/Englisch: https://www.ireb.org/de/cpre/glossary/

7.4 Deutschsprachige Bücher und Artikel (in diesem Lehrplan nicht direkt referenziert)

Bath, G.; McKay, J.; Gronau, V. (Übersetzung) (2015): Praxiswissen Softwaretest – Test Analyst und Technical Test Analyst Aus- und Weiterbildung zum Certified Tester – Advanced Level nach ISTQB-Standard, 3., überarbeitete Auflage, dpunkt.verlag, Heidelberg

Baumgartner, M.; Gwihs, St.; Seidl, R.; Steirer, T.; Wendland, M (2021): Basiswissen Testautomatisierung, 3., aktualisierte und überarbeitete Auflage, dpunkt.verlag, Heidelberg

Daigl, M.; Glunz, R.(2016): ISO 29119 – Die Softwaretest-Normen verstehen und anwenden, dpunkt.verlag, Heidelberg

Hendrickson, E. (2014): Explore It! Wie Softwareentwickler und Tester mit explorativem Testen Risiken reduzieren und Fehler aufdecken (Aus dem Amerikanischen übersetzt von Meike Mertsch), dpunkt.verlag, Heidelberg

Liggesmeyer, P. (2009): Software-Qualität, Spektrum-Verlag, Heidelberg, Berlin

Linz, T. (2023): Testen in Scrum-Projekten Leitfaden für Softwarequalität in der agilen Welt. Aus- und Weiterbildung zum ISTQB® Certified Agile Tester – Foundation Extension, 3., aktualisierte und überarbeitete Auflage, dpunkt.verlag, Heidelberg

Ekssir-Monafred, M. (2022): Soft vs. Hard Skills in Software Testing, https://www.asqf.de/soft-vs-hard-skills-in-software-testing/

Rössler, Peter; Schlich, Maud; Kneuper, Ralf (2013): Reviews in der System- und Softwareentwicklung: Grundlagen, Praxis, kontinuierliche Verbesserung, 1. Auflage, dpunkt.verlag, Heidelberg

Sneed, Harry M.; Baumgartner, Manfred; Seidl, Richard (2011): Der Systemtest – Von den Anforderungen zum Qualitätsnachweis, 3., aktualisierte und erweiterte Auflage, Carl Hanser-Verlag, München

Spillner, A.; Linz, T.: Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB®-Standard, 7., überarbeitete u. aktualisierte Auflage, dpunkt.verlag, Heidelberg (erscheint voraussichtlich Ende 2023)

Spillner, A.; Breymann, U. (2016): Lean Testing für C++-Programmierer – angemessen statt aufwendig testen, dpunkt.verlag, Heidelberg

Winter, Mario; Ekssir-Monfared, Mohsen; Sneed, Harry M.; Seidl, Richard; Borner, Lars (2012): Der Integrationstest – Von Entwurf und Architektur zur Komponenten- und Systemintegration, Carl Hanser Verlag, München

Winter, M.; Roßner, Th.; Brandes, Ch.; Götz, H. (2016): Basiswissen modellbasierter Test, Aus- und Weiterbildung zum ISTQB® Foundation Level – Certified Model-Based Tester, 2., vollständig überarbeitete und aktualisierte Auflage, dpunkt.verlag, Heidelberg

8. Anhang A – Lernziele/kognitive Stufen des Wissens

Die folgenden Lernziele werden in diesem Lehrplan verwendet. Jedes Thema des Lehrplans wird anhand des jeweiligen Lernziels behandelt. Die Lernziele dieses Lehrplans enden mit einem Aktionswort, das dem jeweiligen kognitiven Wissensstand entspricht (siehe unten).

Wissensstufe 1: Sich erinnern (K1) – der Lernende kann einen Begriff oder ein Konzept erkennen, sich erinnern oder abrufen.

Aktionswörter: identifizieren, wiedergeben, erinnern, erkennen.

Beispiele: Der Lernende kann ...

- " ... typische Testziele identifizieren"
- "... die Konzepte der Testpyramide wiedergeben"
- "... den möglichen Mehrwert, den ein Tester für die Iterations- und Releaseplanung schafft, erkennen"

Wissensstufe 2: Verstehen (K2) – Der Lernende kann die Gründe für, oder Erklärungen zu Aussagen zu einem Thema auswählen. Typische beobachtbare Leistungen zusammenfassen, vergleichen, einordnen und Beispiele für Konzepte des Testens nennen.

Aktionswörter: einordnen, vergleichen, etwas gegenüberstellen, differenzieren, unterscheiden, veranschaulichen, erklären, Beispiele geben, interpretieren, zusammenfassen.

Beispiele: Der Lernende kann ...

- "... die verschiedenen Möglichkeiten zum Schreiben von Abnahmekriterien einordnen"
- "... die verschiedenen Rollen beim Testen vergleichen" (nach Gemeinsamkeiten, Unterschieden oder beidem suchen)
- "... zwischen Projektrisiken und Produktrisiken unterscheiden" (ermöglicht die Unterscheidung der Konzepte)
- "... Beispiele zu Zweck und Inhalt eines Testkonzepts geben"
- "... die Auswirkungen des Kontexts auf den Testprozess erklären"
- "... die Aktivitäten des Reviewprozesses zusammenfassen"

Wissensstufe 3: Anwenden (K3) – der Lernende kann ein Verfahren anwenden, wenn er mit einer vertrauten Aufgabe konfrontiert wird, oder das richtige Verfahren auswählen und es auf einen bestimmten Kontext anwenden.

Aktionsverben: anwenden, umsetzen, erstellen, nutzen.

Beispiele: Der Lernende kann ...

- "... Priorisierung von Testfällen anwenden" (sollte sich auf ein Verfahren, eine Technik, einen Prozess, einen Algorithmus usw. beziehen)
- "... einen Fehlerbericht erstellen"
- "... Grenzwertanalyse zur Ableitung von Testfällen anwenden"

Referenzen für die Taxonomiestufen der Lernziele:

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

9. Anhang B – Verfolgbarkeitsmatrix des geschäftlichen Nutzens (Business Outcomes) mit Lernzielen

Dieser Abschnitt listet die Anzahl der Lernziele des Foundation Levels auf, die mit dem geschäftlichen Nutzen in Verbindung stehen, sowie die Verfolgbarkeit zwischen dem geschäftlichen Nutzen des und den Lernzielen des Foundation Levels.

Geschäftlicher Nutzen: Foundation Level		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Verstehen, was Testen ist und warum es nützlich ist	6													
BO2	Die grundlegenden Konzepte des Testens von Software verstehen		22												
BO3	Den Testansatz und die zu anzuwendenden Aktivitäten in Abhängigkeit vom Kontext des Testens identifizieren			6											
BO4	Die Qualität der Dokumentation bewerten und verbessern				9										
BO5	Die Effektivität und Effizienz des Testens steigern					20									
BO6	Den Testprozess auf den Softwareentwicklungslebenszyklus anpassen						6								
BO7	Grundsätze des Testmanagements verstehen							6							
BO8	Klare und verständliche Fehlerberichte schreiben und kommunizieren								1						
BO9	Die Faktoren, die die Prioritäten und den Aufwand für das Testen beeinflussen, verstehen									7					
BO10	Als Teil eines funktionsübergreifenden Teams arbeiten										8				
BO11	Risiken und Vorteile der Testautomatisierung kennen											1			
BO12	Wesentliche Fähigkeiten, die für das Testen erforderlich sind, erkennen												5		
BO13	Die Auswirkungen von Risiken auf das Testen verstehen													4	
BO14	Über den Testfortschritt und die Qualität effektiv berichten														4

Kapitel/ Unter-kapitel	Lernziel – Der Lernende kann …	K-level	Geschäftlicher Nutzen														
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
Kapitel 1	**Grundlagen des Testens**															
1.1	**Was ist Testen?**															
1.1.1	… typische Testziele identifizieren	K1	X													
1.1.2	.. Testen von Debugging unterscheiden	K2		X												
1.2	**Warum ist Testen notwendig?**															
1.2.1	... Beispiele geben, warum Testen notwendig ist	K2	X													
1.2.2	... die Beziehung zwischen Testen und Qualitätssicherung wiedergeben	K1		X												
1.2.3	... zwischen Grundursache, Fehlhandlung, Fehlerzustand und Fehlerwirkung unterscheiden	K2		X												
1.3	**Grundsätze des Testens**															
1.3.1	... die sieben Grundsätze des Testens erklären	K2		X												
1.4	**Testaktivitäten, Testmittel und Rollen des Testens**															
1.4.1	... die verschiedenen Testaktivitäten und -aufgaben zusammenfassen	K2			X											
1.4.2	... die Auswirkungen des Kontexts auf den Testprozess erklären	K2			X			X								
1.4.3	... Testmittel, die die Testaktivitäten unterstützen, unterscheiden	K2			X											
1.4.4	... die Bedeutung der Pflege der Verfolgbarkeit erklären	K2				X	X									
1.4.5	... die verschiedenen Rollen beim Testen vergleichen	K2										X				
1.5	**Grundlegende Fähigkeiten und bewährte Praktiken beim Testen**															
1.5.1	... Beispiele, die für die allgemeinen Kompetenzen, die für das Testen erforderlich sind, geben	K2												X		
1.5.2	... die Vorteile des Whole-Team-Ansatzes wiedergeben	K1										X				
1.5.3	... die Vor- und Nachteile des unabhängigen Testens unterscheiden	K2			X											
Kapitel 2	**Testen während des Softwareentwicklungslebenszyklus**															
2.1	**Testen im Kontext eines Softwareentwicklungslebenszyklus**															

Kapitel/ Unter-kapitel	Lernziel – Der Lernende kann …	K-level	Geschäftlicher Nutzen														
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
2.1.1	... die Auswirkungen des gewählten Softwareentwicklungslebenszyklus auf das Testen erklären	K2						X								
2.1.2	... gute Praktiken für das Testen, die für alle Softwareentwicklungslebenszyklen gelten, wiedergeben	K1						X								
2.1.3	... die Beispiele für Test-First-Ansätze in der Entwicklung wiedergeben	K1					X									
2.1.4	... die möglichen Auswirkungen von DevOps auf das Testen zusammenfassen	K2					X	X			X	X				
2.1.5	... den Shift-Left-Ansatz erklären	K2					X	X								
2.1.6	... den Einsatz von Retrospektiven als Mechanismus zur Prozessverbesserung erklären	K2					X					X				
2.2	**Teststufen und Testarten**															
2.2.1	... die verschiedenen Teststufen unterscheiden	K2		X	X											
2.2.2	... die verschiedenen Testarten unterscheiden	K2		X												
2.2.3	... Fehlernachtests von Regressionstests unterscheiden	K2		X												
2.3	**Wartungstest**															
2.3.1	... Wartungstest und dessen Auslöser zusammenfassen	K2		X					X							
Kapitel 3	**Statischer Test**															
3.1	**Grundlagen des statischen Tests**															
3.1.1	... Produktarten, die mit den verschiedenen statischen Testverfahren geprüft werden können, erkennen	K1				X	X									
3.1.2	... den Wert statischer Tests erklären	K2	X			X	X									
3.1.3	... statischen und dynamischen Test vergleichen und gegenüberstellen	K2				X	X									
3.2	**Feedback- und Reviewprozess**															
3.2.1	... Vorteile eines frühzeitigen und häufigen Stakeholder-Feedbacks erkennen	K1	X			X						X				
3.2.2	... die Aktivitäten des Reviewprozesses zusammenfassen	K2			X	X										
3.2.3	... die bei der Durchführung von Reviews den Hauptrollen zugewiesenen Verantwortlichkeiten wiedergeben	K1				X								X		

Kapitel/ Unter-kapitel	Lernziel – Der Lernende kann …	K-level	Geschäftlicher Nutzen														
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
3.2.4	... verschiedene Arten von Reviews vergleichen und gegenüberstellen	K2		X												
3.2.5	... die Faktoren, die zu einem erfolgreichen Review beitragen, wiedergeben	K1					X							X		
Kapitel 4	**Testanalyse und -entwurf**															
4.1	**Testverfahren im Überblick**															
4.1.1	... Black-Box-Testverfahren, White-Box-Testverfahren und erfahrungsbasierte Testverfahren unterscheiden	K2		X												
4.2	**Black-Box-Testverfahren**															
4.2.1	... Äquivalenzklassenbildung zur Ableitung von Testfällen anwenden	K3					X									
4.2.2	... Grenzwertanalyse zur Ableitung von Testfällen anwenden	K3					X									
4.2.3	... Entscheidungstabellentest für die Ableitung von Testfällen anwenden	K3					X									
4.2.4	... Zustandsübergangstest zur Ableitung von Testfällen anwenden	K3					X									
4.3	**White-Box-Testverfahren**															
4.3.1	... Anweisungstest erklären	K2		X												
4.3.2	... Zweigtest erklären	K2		X												
4.3.3	... den Wert des White-Box-Tests erklären	K2	X	X												
4.4	**Erfahrungsbasierter Test**															
4.4.1	... intuitive Testfallermittlung erklären	K2		X												
4.4.2	... explorativen Test erklären	**K2**		**X**												
4.4.3	... checklistenbasierten Test erklären	K2		X												
4.5	**Auf Zusammenarbeit basierende Testansätze**															
4.5.1	... das Schreiben von User Storys in Zusammenarbeit mit Entwicklern und Fachvertretern erklären	K2				X						X				
4.5.2	... die verschiedenen Möglichkeiten zum Schreiben von Abnahmekriterien einordnen	K2										X				
4.5.3	... abnahmetestgetriebene Entwicklung (ATDD) zur Ableitung von Testfällen anwenden	K3					X									
Kapitel 5	**Management der Testaktivitäten**															

Kapitel/ Unter-kapitel	Lernziel – Der Lernende kann …	K-level	Geschäftlicher Nutzen														
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
5.1	**Testplanung**															
5.1.1	... Beispiele zu Zweck und Inhalt eines Testkonzepts geben	K2		X					X							
5.1.2	... den möglichen Mehrwert, den ein Tester für die Iterations- und Releaseplanung schafft, erkennen	K1	X									X		X		
5.1.3	... Eingangskriterien und Endekriterien vergleichen und gegenüberstellen	K2				X		X								X
5.1.4	... Schätzverfahren zur Berechnung des erforderlichen Testaufwands anwenden	K3							X		X					
5.1.5	... Priorisierung von Testfällen anwenden	K3							X		X					
5.1.6	... die Konzepte der Testpyramide wiedergeben	K1		X												
5.1.7	... die Testquadranten und ihre Beziehungen zu Teststufen und Testarten zusammenfassen	K2		X							X					
5.2	**Risikomanagement**															
5.2.1	... die Risikostufe anhand der Eintrittswahrscheinlichkeit des Risikos und des Schadensausmaßes des Risikos identifizieren	K1							X						X	
5.2.2	... zwischen Projektrisiken und Produktrisiken unterscheiden	K2		X											X	
5.2.3	... den möglichen Einfluss der Produktrisikoanalyse auf Intensität und Umfang des Testens erklären	K2					X				X				X	
5.2.4	... mögliche Maßnahmen, die als Reaktion auf analysierte Produktrisiken ergriffen werden können, erklären	K2		X			X								X	
5.3	**Testüberwachung, Teststeuerung und Testabschluss**															
5.3.1	... die beim Testen verwendeten Metriken wiedergeben	K1									X					X
5.3.2	... Zweck, Inhalt und Zielgruppen von Testberichten zusammenfassen	K2					X				X					X
5.3.3	... Beispiele geben, wie man den Teststatus kommunizieren kann	K2												X		X
5.4	**Konfigurationsmanagement**															
5.4.1	... mögliche Unterstützung des Testens durch das Konfigurationsmanagement zusammenfassen	K2					X		X							

Kapitel/ Unter-kapitel	Lernziel – Der Lernende kann …	K-level	Geschäftlicher Nutzen													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
5.5	**Fehlermanagement**															
5.5.1	… einen Fehlerbericht erstellen	K3		X						X						
Kapitel 6	**Testwerkzeuge**															
6.1	**Werkzeugunterstützung für das Testen**															
6.1.1	… mögliche Unterstützung des Testens durch verschiedene Arten von Testwerkzeugen erklären	K2					X									
6.2	**Nutzen und Risiken von Testautomatisierung**															
6.2.1	… die Nutzen und Risiken von Testautomatisierung wiedergeben	K1					X						X			

10. Anhang C – Release Notes

Der ISTQB® Foundation Lehrplan v4.0 ist eine umfassende Aktualisierung, die auf dem Foundation Level Lehrplan (v3.1.1) und dem Agile Tester Lehrplan 2014 basiert. Aus diesem Grund gibt es keine detaillierten Versionshinweise pro Kapitel und Abschnitt. Im Folgenden finden Sie jedoch eine Zusammenfassung der wichtigsten Änderungen. Darüber hinaus bietet ISTQB® in einem separaten Dokument mit Versionshinweisen die Verfolgbarkeit zwischen den Lernzielen in der Version 3.1.1 des Foundation Level Lehrplan, der Version 2014 des Agile Tester Lehrplan und den Lernzielen im neuen Foundation Level Lehrplan v4.0 an und zeigt auf, welche Lernziele hinzugefügt, aktualisiert oder entfernt wurden.

Zum Zeitpunkt der Erstellung des Lehrplans (2022-2023) haben mehr als eine Million Menschen in mehr als 100 Ländern die Foundation Level-Prüfung abgelegt, und mehr als 800.000 sind weltweit zertifizierte Tester. Da man davon ausgehen kann, dass alle von ihnen den Foundation Lehrplan gelesen haben, um die Prüfung bestehen zu können, ist der Foundation Lehrplan wahrscheinlich das meistgelesene Dokument zum Thema Softwaretest überhaupt! Mit dieser umfassenden Aktualisierung wird diesem Erbe Rechnung getragen. Außerdem soll die Meinung Hunderttausender weiterer Personen über die Qualität, die das ISTQB® der weltweiten Gemeinschaft der Tester bietet, verbessert werden.

In dieser Version wurden alle Lernziele überarbeitet, um sie atomar zu machen und um eine eins-zu-eins Verfolgbarkeit zwischen Lernziel und Lehrplanabschnitten zu schaffen, so dass es keine Inhalte ohne Lernziel gibt. Ziel war es, diese Version leichter lesbar, verständlich, erlernbar und übersetzbar zu machen, wobei der Schwerpunkt auf dem praktischen Nutzen und der Ausgewogenheit zwischen Wissen und Fähigkeiten liegt.

In dieser Hauptversion wurden die folgenden Änderungen vorgenommen:

- Kürzung des gesamten Lehrplans. Der Lehrplan ist kein Lehrbuch, sondern ein Dokument, das dazu dient, die grundlegenden Elemente eines Einführungskurses in das Testen von Software zu umreißen, einschließlich seiner Themen und auf welchem Niveau sie behandelt werden sollten. Daher gilt insbesondere:
  - In den meisten Fällen wurden Beispiele aus dem Text entfernt. Es ist die Aufgabe eines Schulungsanbieters, die Beispiele sowie die Übungen während der Schulung bereitzustellen.
  - Die "Checkliste zum Verfassen von Lehrplänen" wurde befolgt, die die maximale Textlänge für die Lernziele auf jeder Stufe vorgibt: (K1 = max. 10 Zeilen, K2 = max. 15 Zeilen, K3 = max. 25 Zeilen)
- Reduktion der Anzahl der Lernziele im Vergleich zu den Lehrplänen Foundation Level (FL) v3.1.1 und Agile Tester (AT) v2014 zusammen
  - 14 K1 Lernziele im Vergleich zu 21 Lernzielen in FL v3.1.1 (15) und AT 2014 (6)
  - 42 K2 Lernziele im Vergleich zu 53 Lernzielen in FL v3.1.1 (40) und AT 2014 (13)

- 8 K3 Lernziele im Vergleich zu 15 Lernzielen in FL v3.1.1 (7) und AT 2014 (8)

- Ausführlichere Verweise auf klassische und/oder anerkannte Bücher und Artikel über das Testen von Software und verwandte Themen werden bereitgestellt
- Wesentliche Änderungen in Kapitel 1 (Grundlagen des Testens)
  - Abschnitt über Fähigkeiten beim Testen erweitert und verbessert
  - Abschnitt über den Whole-Team-Ansatz (K1) hinzugefügt
  - Abschnitt über das unabhängige Testen wurde von Kapitel 5 in Kapitel 1 verschoben
- Wesentliche Änderungen in Kapitel 2 (Testen während des Softwareentwicklungslebenszyklus)
  - Die Abschnitte 2.1.1 und 2.1.2 wurden umgeschrieben und verbessert, die entsprechenden Lernziele wurden geändert
  - Mehr Fokus auf Praktiken wie: Test-First-Ansatz (K1), Shift-Left (K2), Retrospektiven (K2)
  - Neuer Abschnitt zum Testen im Kontext von DevOps (K2)
  - Aufteilung der Teststufe Integrationstest in zwei separate Teststufen: Komponentenintegrationstests und Systemintegrationstests
- Wesentliche Änderungen in Kapitel 3 (Statischer Test)
  - Abschnitt über Reviewverfahren, zusammen mit dem K3 Lernziel (Ein Reviewverfahren auf ein Arbeitsergebnis anwenden können, um Fehlerzustände zu finden) entfernt
- Wesentliche Änderungen in Kapitel 4 (Testanalyse und -entwurf)
  - Anwendungsfallbasiertes Testen wurde entfernt (ist aber im Lehrplan Advanced Test Analyst noch enthalten)
  - Stärkerer Fokus auf den auf Zusammenarbeit basierenden Testansatz: neues K3 Lernziel über die Verwendung von ATDD zur Ableitung von Testfällen und zwei neue K2 Lernziele über User Storys und Akzeptanzkriterien
  - Entscheidungstests und -überdeckung werden durch Zweigtests und -überdeckung ersetzt (erstens wird die Zweigüberdeckung in der Praxis häufiger verwendet; zweitens definieren verschiedene Standards die Entscheidung anders als den "Zweig"; drittens wird damit ein subtiler, aber schwerwiegender Fehler des alten FL2018 behoben, der behauptet, dass "100% Entscheidungsüberdeckung 100% Anweisungsüberdeckung impliziert" - dieser Satz ist im Falle von Programmen ohne Entscheidungen falsch)
  - Abschnitt über den Wert des White-Box-Tests verbessert
- Wesentliche Änderungen in Kapitel 5 (Management der Testaktivitäten)
  - Abschnitt über Teststrategien/-vorgehensweisen entfernt

- Neues K3 Lernziel über Schätzverfahren zum Abschätzen des Testaufwands
- Stärkerer Fokus auf die allseits gebräuchlichen Konzepte und Werkzeuge im Testmanagement von agilen Projekten: Iterations- und Releaseplanung (K1), Testpyramide (K1) und Testquadranten (K2)
- Der Abschnitt über Risikomanagement wurde besser strukturiert, indem vier Hauptaktivitäten beschrieben werden: Risikoidentifizierung, Risikobewertung, Risikominderung und Risikoüberwachung

- Wesentliche Änderungen in Kapitel 6 (Testwerkzeuge)
  - Der Inhalt zu einigen Sachverhalten der Testautomatisierung wurde reduziert, da er für den Foundation Level zu anspruchsvoll ist – der Abschnitt über die Auswahl von Werkzeugen, die Durchführung von Pilotprojekten und die Einführung von Werkzeugen im Unternehmen wurde entfernt

11. Index

G

H

I

K

L

M

N

P

Q

R

S

T

U

V

W

Z