Lehrplan

Certified Tester

Foundation Level

Version 2018 V3.1D

International Software Testing Qualifications Board

Deutschsprachige Ausgabe. Herausgegeben durch

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

Copyright Hinweis

Autoren

Autoren für Aktualisierung ISTQB® 2018 v3.1 waren Klaus Olsen (Leitung), Meile Posthuma und Stephanie Ulrich.

An Lokalisierung und Review der Übersetzung der Aktualisierungen haben mitgewirkt: Arne Becher, Ralf Bongard, Milena Donato, Dr. Matthias Hamburg, Andreas Hetz, Tobias Horn, Karl Kemminger, Martin Klonk, Nishan Portoyan, Horst Pohlmann, und Stephanie Ulrich (Leitung).

Autoren für die Aktualisierungen 2018: Klaus Olsen (Leitung), Tauhida Parveen (Stellvertretende Leitung), Rex Black (Projektmanager), Debra Friedenberg, Matthias Hamburg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh und Eshraka Zakaria.

An Lokalisierung und Review der Übersetzung der Aktualisierungen 2018 haben mitgewirkt: Arne Becher, Ralf Bongard, Alisha Bülow et al., Dr. Klaudia Dussa-Zieger, Dr. Matthias Hamburg, Andreas Hetz, Karl Kemminger, Martin Klonk, Elke Mai, Nishan Portoyan, Horst Pohlmann, Atilim Siegmund, Prof. Dr. Andreas Spillner (a.D.), Sabine Uhde und Stephanie Ulrich (Leitung).

Autoren für die Aktualisierungen 2011: Thomas Müller (Leitung), Debra Friedenberg und die ISTQB-Arbeitsgruppe Foundation Level.

Autoren für die Aktualisierungen 2010: Thomas Müller (Leitung), Armin Beer, Martin Klonk und Rahul Verma.

Änderungsübersicht Originalausgabe

Version	Datum	Bemerkung
ISTQB® 2018 V3.1	11-November-2019	Lehrplan Certified Tester Foundation Level Wartungsrelease – siehe Versionshinweise
ISTQB® 2018	4. Juni 2018	Lehrplan Certified Tester Foundation Level, Version 2018
ISTQB® 2011	1. April 2011	Lehrplan Certified Tester Foundation Level, Wartungsrelease – siehe Veröffentlichungshinweise
ISTQB® 2010	30. März 2010	Lehrplan Certified Tester Foundation Level, Wartungsrelease – siehe Veröffentlichungshinweise
ISTQB® 2007	1. Mai 2007	Lehrplan Certified Tester Foundation Level, Wartungsrelease
ISTQB® 2005	1. Juli 2005	Lehrplan Certified Tester Foundation Level
ASQF V2.2	Juli 2003	ASQF Syllabus Foundation Level, Version 2.2 “Lehrplan Grundlagen des Softwaretestens“
ISEB V2.0	25. Februar 1999	ISEB-Lehrplan Software Testing Foundation V2.0

Änderungsübersicht deutschsprachige Ausgabe

Version	Gültig ab:	Bemerkung
2018 V3.1D	20.1. 2020	Deutschsprachiges Wartungsrelease, für Details siehe Versionshinweise
2018 D	3. 9. 2018	Deutschsprachige Fassung des ISTQB-Release 2018
2011 1-0.2	1. 7. 2017	Deutschsprachiges Wartungsrelease, für Details siehe Anhang E
2011 1.0.1	19. 4. 2013	Deutschsprachiges Wartungsrelease, für Details siehe Anhang E
2011	1. 8. 2011	Wartungsrelease, für Details siehe Anhang E
2010 1.0.1	29. 11. 2010	Wartungsrelease: Dieses Update enthält kleinere Korrekturen in Anhang E, Formatierungskorrekturen und den Nachtrag der kognitiven Stufe zu Kapitel 5.2.3 Testeingangskriterien. Des Weiteren wurde ein Übersetzungsfehler in den Kapiteln 1.3 und 2.2.2 korrigiert, „Testleiter“ aus der Begriffsliste entfernt und die im Release 1.0 vorweggenommenen Korrekturen des engl. Lehrplans zu Beginn des Dokuments näher erklärt.
2010	1. 10. 2010	Überarbeitung (Details siehe Release Notes 2010 – Anhang E)
2007	1. 12. 2007	Überarbeitung (Details siehe Release Notes – Anhang E)
2005	1. 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“

Inhaltsverzeichnis

Copyright Hinweis 2
Autoren 2
Inhaltsverzeichnis 4
Dank 8
0. Einführung 10
0.1 Zweck dieses Lehrplans 10
0.2 Der Certified Tester Foundation Level im Softwaretesten 10
0.3 Prüfbare Lernziele und kognitive Stufen des Wissens 10
0.4 Die Foundation-Level-Zertifikatsprüfung 11
0.5 Akkreditierung 11
0.6 Detaillierungsgrad 11
0.7 Wie ist dieser Lehrplan aufgebaut 12
1. Grundlagen des Testens – 175 Minuten 13
1.1 Was ist Testen? 14
1.1.1 Typische Ziele des Testens 14
1.1.2 Testen und Debugging 15
1.2 Warum ist Testen notwendig? 15
1.2.1 Der Beitrag des Testens zum Erfolg 15
1.2.2 Qualitätssicherung und Testen 16
1.2.3 Fehlhandlungen, Fehlerzustände und Fehlerwirkungen 16
1.2.4 Fehlerzustände, Grundursachen und Wirkungen 17
1.3 Sieben Grundsätze des Testens 17
1.4 Testprozess 19
1.4.1 Testprozess im Kontext 19
1.4.2 Testaktivitäten und Aufgaben 20
1.4.3 Testarbeitsergebnisse 25
1.4.4 Verfolgbarkeit zwischen der Testbasis und Testarbeitsergebnissen 27
1.5 Die Psychologie des Testens 28
1.5.1 Humanpsychologie und Testen 28
1.5.2 Denkweisen von Testern und Entwicklern 29
2. Testen im Softwareentwicklungslebenszyklus – 100 Minuten 30
2.1 Softwareentwicklungslebenszyklus-Modelle 31
2.1.1 Softwareentwicklung und Softwaretesten 31
2.1.2 Softwareentwicklungslebenszyklus-Modelle im Kontext 32

2.2 Teststufen ... 33
2.2.1 Komponententest ... 34
2.2.2 Integrationstest ... 35
2.2.3 Systemtest ... 37
2.2.4 Abnahmetest ... 39
2.3 Testarten ... 42
2.3.1 Funktionale Tests ... 42
2.3.2 Nicht-funktionale Tests ... 43
2.3.3 White-Box-Tests ... 43
2.3.4 Änderungsbezogene Tests ... 44
2.3.5 Testarten und Teststufen ... 45
2.4 Wartungstest ... 46
2.4.1 Auslöser für Wartung ... 46
2.4.2 Auswirkungsanalyse für Wartung ... 47
3. Statischer Test – 135 Minuten ... 48
3.1 Grundlagen des statischen Tests ... 49
3.1.1 Arbeitsergebnisse, die durch statische Tests geprüft werden können ... 49
3.1.2 Vorteile des statischen Tests ... 49
3.1.3 Unterschiede zwischen statischen und dynamischen Tests ... 50
3.2 Reviewprozess ... 51
3.2.1 Reviewprozess für Arbeitsergebnisse ... 51
3.2.2 Rollen und Verantwortlichkeiten in einem formalen Review ... 52
3.2.3 Reviewarten ... 53
3.2.4 Die Anwendung von Reviewverfahren ... 55
3.2.5 Erfolgsfaktoren für Reviews ... 56
4. Testverfahren – 330 Minuten ... 58
4.1 Kategorien von Testverfahren ... 59
4.1.1 Kategorien von Testverfahren und ihre Eigenschaften ... 59
4.2 Black-Box-Testverfahren ... 60
4.2.1 Äquivalenzklassenbildung ... 60
4.2.2 Grenzwertanalyse ... 61
4.2.3 Entscheidungstabellentests ... 61
4.2.4 Zustandsübergangstest ... 62
4.2.5 Anwendungsfallbasierter Test ... 63
4.3 White-Box-Testverfahren ... 64
4.3.1 Anweisungstests und -überdeckung ... 64
4.3.2 Entscheidungstest und -überdeckung ... 64

4.3.3 Der Beitrag von Anweisungs- und Entscheidungstests ... 64
4.4 Erfahrungsbasierte Testverfahren ... 64
4.4.1 Intuitive Testfallermittlung ... 65
4.4.2 Exploratives Testen ... 65
4.4.3 Checklistenbasiertes Testen ... 65
5. Testmanagement – 225 Minuten ... 66
5.1 Testorganisation ... 68
5.1.1 Unabhängiges Testen ... 68
5.1.2 Aufgaben eines Testmanagers und eines Testers ... 69
5.2 Testplanung und -schätzung ... 71
5.2.1 Zweck und Inhalt eines Testkonzepts ... 71
5.2.2 Teststrategie und Testvorgehensweise ... 71
5.2.3 Eingangs- und Endekriterien (Definition-of-Ready und Definition-of-Done) ... 72
5.2.4 Testausführungsplan ... 73
5.2.5 Den Testaufwand beeinflussende Faktoren ... 73
5.2.6 Testschätzverfahren ... 74
5.3 Testüberwachung und -steuerung ... 75
5.3.1 Beim Testen verwendete Metriken ... 75
5.3.2 Zwecke, Inhalte und Zielgruppen für Testberichte ... 76
5.4 Konfigurationsmanagement ... 77
5.5 Risiken und Testen ... 77
5.5.1 Definition des Risikos ... 77
5.5.2 Produkt- und Projektrisiken ... 77
5.5.3 Risikobasiertes Testen und Produktqualität ... 79
5.6 Fehlermanagement ... 79
6. Werkzeugunterstützung für das Testen – 40 Minuten ... 82
6.1 Überlegungen zu Testwerkzeugen ... 83
6.1.1 Klassifizierung von Testwerkzeugen ... 83
6.1.2 Nutzen und Risiken der Testautomatisierung ... 85
6.1.3 Besondere Betrachtung der Testausführungs- und Testmanagementwerkzeuge ... 86
6.2 Effektive Nutzung von Werkzeugen ... 87
6.2.1 Hauptgrundsätze für die Auswahl von Werkzeugen ... 87
6.2.2 Pilotprojekte für die Einführung eines Werkzeugs in einem Unternehmen ... 87
6.2.3 Erfolgsfaktoren für Werkzeuge ... 88
7. Referenzen ... 89
7.1 Normen/Standards ... 89
7.2 ISTQB-Dokumente ... 89

7.3 Bücher und Artikel 89
7.4 Deutschsprachige Bücher und Artikel (in diesem Lehrplan nicht direkt referenziert) 90
7.5 Andere Quellen (in diesem Lehrplan nicht direkt referenziert) 91
8. Anhang A – Hintergrundinformation zum Lehrplan 92
9. Anhang B – Lernziele/Kognitive Stufen des Wissens 94
10. Anhang C – Veröffentlichungshinweise 96
11. Anhang D – Übersetzung von Fachbegriffen 97
12. Index 99

Dank

Das englischsprachige Dokument wurde formal von der Generalversammlung des ISTQB® am 11. November 2019 freigegeben.

Es wurde von einem Team des International Software Testing Qualifications Board erstellt: Klaus Olsen (Leitung), Meile Posthuma und Stephanie Ulrich.

Das Team dankt den Mitglieds-Boards für ihre Reviewkommentare zum Foundation Lehrplan 2018.

Das German Testing Board (GTB), das Swiss Testing Board (STB) und das Austrian Testing Board (ATB) danken ebenso dem Reviewteam der deutschsprachigen Fassung 2018 V3.1 D: Arne Becher, Ralf Bongard, Milena Donato, Dr. Matthias Hamburg, Andreas Hetz, Tobias Horn, Karl Kemminger, Martin Klonk, Nishan Portoyan, Horst Pohlmann, und Stephanie Ulrich (Leitung).

Der Foundation Lehrplan 2018 wurde von einem Team des International Software Testing Qualifications Board erstellt: Klaus Olsen (Leitung), Tauhida Parveen (Stellvertretende Leitung), Rex Black (Projektmanager), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh und Eshraka Zakaria.

Das Team dankt Rex Black und Dorothy Graham für ihre technischen Überarbeitungen sowie dem Reviewteam, dem Cross-Reviewteam und den Mitglieds-Boards für ihre Vorschläge und Beiträge.

Die folgenden Personen waren am Review, der Kommentierung und der Abstimmung zu diesem Lehrplan beteiligt: Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar und Karolina Zmitrowicz.

International Software Testing Qualifications Board, Arbeitsgruppe Foundation Level (Ausgabe 2018): Klaus Olsen (Leitung), Tauhida Parveen (Stellvertretende Leitung), Rex Black (Projektmanager), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria und Stevan Zivanovic. Das Kernteam dankt dem Reviewteam und allen Mitglieds-Boards für ihre Vorschläge.

Das German Testing Board (GTB), das Swiss Testing Board (STB) und das Austrian Testing Board (ATB) danken ebenso dem Reviewteam der deutschsprachigen Fassung 2018: Arne Becher, Ralf

Bongard, Alisha Bülow et al., Dr. Klaudia Dussa-Zieger, Dr. Matthias Hamburg, Andreas Hetz, Karl Kemminger, Martin Klonk, Elke Mai, Nishan Portoyan, Horst Pohlmann, Atilim Siegmund, Prof. Dr. Andreas Spillner (a.D.), Sabine Uhde und Stephanie Ulrich (Leitung).

International Software Testing Qualifications Board, Arbeitsgruppe Foundation Level (Ausgabe 2011): Thomas Müller (Leitung), Debra Friedenberg. Das Kernteam dankt dem Reviewteam (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 Mitglieds-Boards für ihre Vorschläge.

Das German Testing Board (GTB), das Swiss Testing Board (STB) und das Austrian Testing Board (ATB) danken ebenso dem Review-Team der deutschsprachigen Fassung 2011: Arne Becher, Francisca Blunschi (STB), Thomas Müller (STB), Horst Pohlmann (GTB), Sabine Uhde (GTB) und Stephanie Ulrich (Leitung, GTB).

International Software Testing Qualifications Board, Arbeitsgruppe Foundation Level (Ausgabe 2010): Thomas Müller (Leitung), Rahul Verma, Martin Klonk und Armin Beer. Das Kernteam dankt dem Reviewteam (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 Mitglieds-Boards für ihre Vorschläge.

Das German Testing Board (GTB), das Swiss Testing Board (STB) und das Austrian Testing Board (ATB) danken ebenso dem Reviewteam der deutschsprachigen Fassung 2010: Jörn Münzel (GTB), Timea Illes-Seifert (GTB), Horst Pohlmann (GTB), Stephanie Ulrich (Leitung, GTB), Josef Bruder (STB), Matthias Daigl (Imbus), Falk Fraikin, Karl Kemminger (ATB), Reto Müller (STB), Doris Preindl (ATB), Thomas Puchter (ATB) und Wolfgang Zuser (ATB).

International Software Testing Qualifications Board, Arbeitsgruppe Foundation Level (Ausgabe 2007): Thomas Müller (Leitung), Dorothy Graham, Debra Friedenberg und Erik van Veenendaal. Das Kernteam dankt dem Reviewteam (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson und Wonil Kwon) und allen Mitglieds-Boards für ihre Vorschläge.

Die Arbeitsgruppenmitglieder der deutschsprachigen Übersetzung (Ausgabe 2007) bedanken sich beim Reviewteam, bestehend aus Mitgliedern des German Testing Board e.V. (GTB) und des Swiss Testing Board (STB), für die vielen konstruktiven Verbesserungsvorschläge: Timea Illes (GTB), Arne Becher, Thomas Müller (STB), Horst Pohlmann (GTB) und Stephanie Ulrich (GTB).

International Software Testing Qualifications Board, 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 Reviewteam und allen Mitglieds-Boards für ihre Vorschläge.

0. Einführung

0.1 Zweck dieses Lehrplans

Dieser Lehrplan definiert die Basisstufe (Foundation Level) des Softwaretestausbildungsprogramms des International Software Testing Qualifications Board (im Folgenden kurz ISTQB® genannt). Das ISTQB® stellt diesen Lehrplan wie folgt zur Verfügung:

1. Nationalen Mitglieds-Boards, die den Lehrplan in ihre Sprache(n) übersetzen und Seminaranbieter akkreditieren dürfen. Nationale Mitglieds-Boards dürfen den Lehrplan an die Anforderungen ihrer nationalen Sprache anpassen und Referenzen auf lokale Veröffentlichungen hinzufügen.
2. Zertifizierungsstellen zur Ableitung von Prüfungsfragen in ihrer nationalen Sprache, die an die Lernziele dieses Lehrplans angepasst sind.
3. Seminaranbietern zur Erstellung von Kursmaterialien und zur Bestimmung angemessener Lehrmethoden.
4. Zertifizierungskandidaten zur Vorbereitung auf die Zertifizierungsprüfung (entweder als Teil eines Seminars 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.

Das ISTQB® kann anderen Stellen die Nutzung dieses Lehrplans zu anderen Zwecken erlauben, sofern sie zuvor die schriftliche Erlaubnis des ISTQB® erfragen und erhalten.

0.2 Der Certified Tester Foundation Level im Softwaretesten

Die Basisstufe des „Certified Tester“-Ausbildungsprogramms soll alle in das Thema Softwaretesten involvierten Personen ansprechen. Das schließt Personen in Rollen wie Tester, Testanalysten, Testentwickler, Testberater, Testmanager, Benutzerabnahmetester und Softwareentwickler mit ein. Die Basisstufe richtet sich ebenso an Personen in der Rolle Product Owner, Projektmanager, Qualitätsmanager, Softwareentwicklungsmanager, Systemanalytiker (Businessanalysten), IT-Leiter oder Managementberater, welche sich ein Basiswissen und Grundlagenverständnis über das Thema Softwaretesten erwerben möchten. Besitzer des Foundation-Zertifikats können an Zertifizierungen höherer Stufen im Softwaretesten teilnehmen.

Der ISTQB®-Foundation-Level-Überblick 2018 ist ein separates Dokument, welches auch für den Foundation Lehrplan 2019 3.1D gilt und dass die folgenden Informationen enthält:

- Geschäftlicher Nutzen des Lehrplans
- Eine Matrix, die die Nachverfolgbarkeit zwischen geschäftlichem Nutzen und Lernzielen darstellt
- Zusammenfassung dieses Lehrplans

0.3 Prüfbare Lernziele und kognitive Stufen des Wissens

Lernziele unterstützen den geschäftlichen Nutzen und werden verwendet, um die Prüfungen für Certified Tester Foundation Level zu erstellen.

Im Allgemeinen sind alle Inhalte dieses Lehrplans auf K1-Stufe prüfbar, außer der Einführung und der Anhänge. Das bedeutet, vom Prüfling kann gefordert werden, einen Schlüsselbegriff oder ein Konzept aus jedem der sechs Kapitel erkennen, sich daran erinnern oder wiedergeben zu können. Die Wissensstufen der spezifischen Lernziele werden am Beginn jedes Kapitels genannt und sind wie folgt klassifiziert:

- K1: erinnern
- K2: verstehen
- K3: anwenden

Weitere Einzelheiten und Beispiele von Lernzielen werden in Anhang B gegeben.

Die Definitionen aller Begriffe, die als Schlüsselbegriffe unterhalb der Kapitelüberschrift aufgelistet sind, sollen im Gedächtnis behalten werden (K1), auch wenn sie nicht ausdrücklich in den Lernzielen erwähnt werden.

0.4 Die Foundation-Level-Zertifikatsprüfung

Die Foundation-Level-Zertifikatsprüfung basiert auf diesem Lehrplan. Antworten auf Prüfungsfragen können die Nutzung von Materialien auf Grundlage von mehr als einem Abschnitt dieses Lehrplans erfordern. Alle Abschnitte dieses Lehrplans sind prüfungsrelevant, außer der Einführung und der Anhänge. Standards, Bücher und andere ISTQB®-Lehrpläne sind als Referenzen genannt, aber deren Inhalt ist nicht über das hinaus prüfungsrelevant, was in diesem Lehrplan selbst aus diesen Standards, Büchern oder anderen ISTQB®-Lehrplänen zusammengefasst ist.

Das Format der Prüfung ist Multiple Choice. Es gibt 40 Fragen. Zum Bestehen der Prüfung müssen mindestens 65% der Fragen (also 26 Fragen) korrekt beantwortet werden.

Prüfungen können als Teil eines akkreditierten Seminars oder unabhängig davon (z.B. in einem Prüfungszentrum oder in einer öffentlichen Prüfung) absolviert werden. Der Abschluss eines akkreditierten Seminars ist keine Voraussetzung für die Teilnahme an der Prüfung.

0.5 Akkreditierung

Nationale ISTQB®-Mitglieds-Boards dürfen Seminaranbieter akkreditieren, deren Kursmaterialien diesem Lehrplan entsprechen. 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 der/den Organisation/en bezogen werden, welche die Akkreditierung im Auftrag des nationalen Boards durchführt/durchführen. Ein akkreditierter Kurs ist als zu diesem Lehrplan konform anerkannt und darf eine ISTQB®-Prüfung beinhalten.

0.6 Detaillierungsgrad

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

- Allgemeine Lehrziele, die die Intention des Foundation Level beschreiben
- Eine Liste von Begriffen, die die Studierenden kennen müssen
- Lernziele für jeden Wissensbereich, die das zu erzielende kognitive Lernergebnis beschreiben
- Eine Beschreibung der wichtigen Konzepte sowie der zugehörigen Referenzen auf Quellen wie allgemein anerkannte Fachliteratur und Standards/Normen

Der Inhalt des Lehrplans ist keine Beschreibung des gesamten Wissensgebiets „Softwaretesten“. Er spiegelt den Detaillierungsgrad wider, der im Foundation-Level-Kurs abgedeckt wird. Er konzentriert sich auf Testkonzepte und Verfahren, die auf alle Softwareprojekte anwendbar sind, auch auf agile Projekte. Dieser Lehrplan beinhaltet keine spezifischen Lernziele, die sich auf spezielle Softwareentwicklungslebenszyklen oder Methoden beziehen, aber er beschreibt, wie diese Konzepte in agilen Projekten, in anderen Arten von iterativen und inkrementellen Lebenszyklen und in sequenziellen Lebenszyklen angewendet werden.

0.7 Wie ist dieser Lehrplan aufgebaut

Es gibt sechs Kapitel mit prüfungsrelevantem Inhalt. Die Hauptüberschrift für jedes Kapitel legt die Zeit fest, die für das Kapitel vorgesehen ist. Für die Unterkapitel ist keine Zeitangabe vorhanden. Für akkreditierte Seminare erfordert dieser Lehrplan mindestens 16,75 Std. Unterricht, der sich wie folgt auf die sechs Kapitel aufteilt:

- Kapitel 1: 175 Minuten Grundlagen des Testens
- Kapitel 2: 100 Minuten Testen im Softwareentwicklungslebenszyklus
- Kapitel 3: 135 Minuten Statischer Test
- Kapitel 4: 330 Minuten Testverfahren
- Kapitel 5: 225 Minuten Testmanagement
- Kapitel 6: 40 Minuten Werkzeugunterstützung für das Testen

1. Grundlagen des Testens – 175 Minuten

Schlüsselbegriffe

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

Lernziele für die Grundlagen des Testens

1.1 Was ist Testen?

FL-1.1.1 (K1) Typische Ziele des Testens identifizieren können

FL-1.1.2 (K2) Testen von Debugging unterscheiden können

1.2 Warum ist Testen notwendig?

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

FL-1.2.2 (K2) Die Beziehung zwischen Testen und Qualitätssicherung beschreiben können und Beispiele dafür geben können, wie Testen zu höherer Qualität beiträgt

FL-1.2.3 (K2) Zwischen Fehlhandlung, Fehlerzustand und Fehlerwirkung unterscheiden können

FL-1.2.4 (K2) Zwischen der Grundursache eines Fehlerzustands und seinen Auswirkungen unterscheiden können

1.3 Sieben Grundsätze des Testens

FL-1.3.1 (K2) Die sieben Grundsätze des Softwaretestens erklären können

1.4 Testprozess

FL-1.4.1 (K2) Die Auswirkungen des Kontexts auf den Testprozess erklären können

FL-1.4.2 (K2) Die Testaktivitäten und zugehörigen Aufgaben innerhalb des Testprozesses beschreiben können

FL-1.4.3 (K2) Arbeitsergebnisse unterscheiden können, die den Testprozess unterstützen

FL-1.4.4 (K2) Die Bedeutung der Pflege der Verfolgbarkeit zwischen Testbasis und Testarbeitsergebnissen erklären können

1.5 Die Psychologie des Testens

FL-1.5.1 (K1) Die psychologischen Faktoren identifizieren können, die den Erfolg des Testens beeinflussen

FL-1.5.2 (K2) Den Unterschied zwischen der für Testaktivitäten erforderlichen Denkweise und der für Entwicklungsaktivitäten erforderlichen Denkweise erklären können

1.1 Was ist Testen?

Softwaresysteme sind ein wesentlicher Bestandteil des Lebens: von Fachanwendungen (z.B. im Bankwesen) bis hin zu Verbraucherprodukten (z.B. Autos). Die meisten Menschen haben bereits Erfahrungen mit Software gemacht, die nicht wie erwartet funktioniert hat. Software, die nicht korrekt arbeitet, kann zu vielfältigen Problemen führen, u.a. zu Geld-, Zeit- oder Imageverlust, sogar bis hin zu Verletzungen oder Tod. Softwaretesten ist ein Mittel, die Qualität von Software zu beurteilen und das Risiko einer Fehlerwirkung im Betrieb zu reduzieren.

Es ist eine gängige Fehleinschätzung, dass Testen ausschließlich darin besteht, Tests auszuführen, d.h. im Sinne von: die Software auszuführen und die Ergebnisse zu prüfen. Wie in Abschnitt *1.4 Testprozess* beschrieben, ist Softwaretesten ein Prozess, der viele unterschiedliche Aktivitäten umfasst. Die Testdurchführung einschließlich der Prüfung der Ergebnisse ist nur eine dieser Aktivitäten. Der Testprozess beinhaltet darüber hinaus Aktivitäten wie die Planung, die Analyse, den Entwurf und die Realisierung von Tests, das Berichten über Testfortschritt und -ergebnisse und die Beurteilung der Qualität eines Testobjekts.

Einige Tests beinhalten die Ausführung von Komponenten oder Systemen; derartige Tests werden dynamische Tests genannt. Andere Tests umfassen kein Ausführen von zu testenden Komponenten oder Systemen; derartige Tests werden statische Tests genannt. Testen beinhaltet also auch die Prüfung von Arbeitsergebnissen wie Anforderungen, User-Stories und Quellcode im Rahmen von Reviews.

Eine weitere gängige Fehleinschätzung ist, dass sich Testen ausschließlich auf die Verifizierung von Anforderungen, User-Stories oder anderen Spezifikationen konzentriert. Auch wenn das Testen es erfordert, zu prüfen, ob das System spezifische Anforderungen erfüllt, so umfasst es auch die Validierung, also die Prüfung, ob das System in seiner Einsatzumgebung die Bedürfnisse von Benutzern und anderen Stakeholdern erfüllen wird.

Testaktivitäten werden in unterschiedlichen Lebenszyklen unterschiedlich organisiert und durchgeführt (siehe Abschnitt *2.1 Softwareentwicklungslebenszyklus-Modelle*).

1.1.1 Typische Ziele des Testens

Für jedes mögliche Projekt können die Ziele des Testens Folgendes beinhalten:

- Arbeitsergebnisse wie Anforderungen, User-Stories, Architekturdesign und Code bewerten, um Fehler zu identifizieren und in Folgearbeitsergebnissen zu vermeiden
- Verifizieren, ob alle spezifischen Anforderungen erfüllt sind
- Prüfen, ob das Testobjekt vollständig ist und validieren, ob das Testobjekt so funktioniert, wie es die Benutzer und andere Stakeholder erwarten
- Vertrauen in das Qualitätsniveau des Testobjekts schaffen
- Fehlerwirkungen und Fehlerzustände aufdecken, wodurch man Risiken aufgrund einer unzureichenden Softwarequalität reduziert
- Stakeholdern ausreichende Informationen zur Verfügung stellen, damit diese fundierte Entscheidungen treffen können, insbesondere bezüglich des Qualitätsniveaus des Testobjekts
- Konform mit vertraglichen, rechtlichen oder regulatorischen Anforderungen oder Standards zu sein und/oder um die Konformität (compliance) des Testobjekts mit diesen Anforderungen oder Standards zu verifizieren

Die Ziele des Testens können abhängig vom Kontext der zu testenden Komponente oder des Systems, das getestet wird, von der Teststufe und dem Softwareentwicklungslebenszyklus-Modell variieren. Diese Unterschiede können zum Beispiel Folgendes beinhalten:

- Während des Komponententests kann es ein Ziel sein, so viele Fehlerwirkungen wie möglich zu finden, damit die zugrunde liegenden Fehlerzustände frühzeitig identifiziert und behoben werden. Ein weiteres Ziel kann es sein, die Codeüberdeckung durch Komponententests zu erhöhen.
- Während des Abnahmetests kann es ein Ziel sein, zu bestätigen, dass das System so funktioniert wie erwartet und die Anforderungen erfüllt wurden. Ein weiteres Ziel dieses Tests kann darin bestehen, Stakeholdern Informationen über das Risiko einer Systemfreigabe zu einem festgelegten Zeitpunkt zu geben.

1.1.2 Testen und Debugging

Testen und Debugging sind unterschiedliche Dinge. Die Durchführung von Tests kann Fehlerwirkungen aufzeigen, die durch Fehlerzustände in der Software hervorgerufen werden. Debugging ist im Gegensatz dazu die Entwicklungsaktivität, die solche Fehlerzustände findet, analysiert und behebt. Die nachfolgenden Fehlernachtests prüfen, ob die Debugging-Maßnahmen die ursprünglichen Fehlerzustände behoben haben. In manchen Fällen sind Tester für die ursprünglichen Tests und die abschließenden Fehlernachtests verantwortlich, während Entwickler das Debugging, die zugehörigen Komponenten- und Komponentenintegrationstests (kontinuierliche Integration) durchführen. Dennoch können in der agilen Softwareentwicklung und in einigen anderen Softwareentwicklungslebenszyklusmodellen Tester auch am Debugging und im Komponententest involviert sein.

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

1.2 Warum ist Testen notwendig?

Gründliches Testen von Komponenten oder Systemen und ihrer zugehörigen Dokumentation kann dabei helfen, das Risiko von Fehlerwirkungen zu reduzieren, die während des Betriebs auftreten können. Wenn Fehler entdeckt und in der Folge behoben werden, trägt dies zur Qualität der Komponenten oder Systeme bei. Darüber hinaus kann Softwaretesten auch notwendig sein, um vertragliche oder rechtliche Anforderungen oder branchenspezifische Standards zu erfüllen.

1.2.1 Der Beitrag des Testens zum Erfolg

Historisch betrachtet ist es schon immer üblich gewesen, dass Software und Systeme in Betrieb genommen werden, obwohl Fehlerwirkungen infolge von Fehlerzuständen auftreten oder in anderer Weise die Bedürfnisse der Stakeholder nicht erfüllt werden. Allerdings kann der Einsatz geeigneter Testverfahren die Häufigkeit derartiger problematischer Inbetriebnahmen reduzieren, wenn diese Verfahren mit dem entsprechenden Grad an Testkompetenz, in den geeigneten Teststufen und zum richtigen Zeitpunkt im Softwareentwicklungslebenszyklus eingesetzt werden. Beispiele dafür sind u.a.:

- Der Einbezug von Testern in Anforderungsreviews oder bei User-Stories-Verfeinerungen (Refinements) kann Fehlerzustände in diesen Arbeitsergebnissen aufdecken. Die Identifizierung und Entfernung von Fehlerzuständen in Anforderungen reduzieren das Risiko der Entwicklung von fehlerhaften oder nicht testbaren Features.
- Die enge Zusammenarbeit von Testern mit Systementwicklern während des Systementwurfs kann das gemeinsame Verständnis für den Entwurf und die Möglichkeiten, es zu testen, verbessern. Dieses bessere Verständnis kann das Risiko grundlegender Entwurfsfehler reduzieren und die Identifikation potenzieller Tests zu einem frühen Zeitpunkt ermöglichen.

- Die enge Zusammenarbeit von Testern und Entwicklern während der Entwicklung des Codes kann das gemeinsame Verständnis der Beteiligten für den Code und wie dieser zu testen ist, verbessern. Dieses bessere Verständnis kann das Risiko von Fehlerzuständen innerhalb des Codes und in Tests reduzieren.
- Die Verifizierung und Validierung der Software durch Tester vor der Freigabe kann Fehlerwirkungen aufdecken, die andernfalls übersehen worden wären, und kann den Prozess des Entfernens (d.h. Debugging) von Fehlern, die die Fehlerwirkungen hervorgerufen haben, unterstützen. Dies erhöht die Wahrscheinlichkeit, dass die Software den Bedürfnissen der Stakeholder entspricht und die Anforderungen erfüllt.

Zusätzlich zu diesen Beispielen trägt das Erreichen definierter Testziele (siehe Abschnitt 1.1.1 *Typische Ziele des Testens*) zum allgemeinen Erfolg der Softwareentwicklung und der Wartung bei.

1.2.2 Qualitätssicherung und Testen

Obwohl der Begriff *Qualitätssicherung* (oder kurz QS) oft in Bezug auf das Testen genutzt wird, sind Qualitätssicherung und Testen nicht das Gleiche, allerdings sind sie eng verwandt. Ein allgemeineres Konzept, das Qualitätsmanagement, vereint die beiden Begriffe. Qualitätsmanagement beinhaltet alle Aktivitäten, die eine Organisation im Hinblick auf Qualität leiten und steuern. Neben anderen Aktivitäten beinhaltet Qualitätsmanagement sowohl Qualitätssicherung als auch Qualitätssteuerung. Qualitätssicherung konzentriert sich in der Regel auf die Einhaltung gültiger Prozesse, um Vertrauen dafür zu schaffen, dass die angemessenen Qualitätsgrade erreicht werden. Wenn Prozesse korrekt befolgt werden, weisen die Arbeitsergebnisse, die aus solchen Prozessen hervorgehen, in der Regel eine höhere Qualität auf, was zur Vermeidung von Fehlerzuständen beiträgt. Darüber hinaus ist die Nutzung der Grundursachenanalyse zur Feststellung und Behebung von Fehlerzuständen in Verbindung mit der korrekten Verwendung der Befunde von Retrospektiven (Bewertungssitzungen) zur Verbesserung von Prozessen von Bedeutung für eine effektive Qualitätssicherung.

Qualitätssteuerung beinhaltet verschiedene Aktivitäten, darunter Testaktivitäten, die das Erreichen von angemessenen Qualitätsgraden unterstützen. Testaktivitäten sind Teil des gesamten Softwareentwicklungs- und -wartungsprozesses. Da die Qualitätssicherung sich mit der korrekten Ausführung des gesamten Prozesses beschäftigt, unterstützt Qualitätssicherung korrektes Testen. Wie in den Abschnitten 1.1.1 *Typische Ziele des Testens* und 1.2.1 *Der Beitrag des Testens zum Erfolg* beschrieben, trägt Testen zum Erreichen von Qualität auf verschiedene Arten bei.

1.2.3 Fehlhandlungen, Fehlerzustände und Fehlerwirkungen

Eine Person kann eine Fehlhandlung vornehmen (einen Fehler machen), was zur Entstehung eines Fehlerzustands (auch nur Fehler oder Bug genannt) im Softwarecode oder in einem zugehörigen Arbeitsergebnis führt. Eine Fehlhandlung, die zur Entstehung eines Fehlerzustands in einem Arbeitsergebnis führt, kann eine weitere Fehlhandlung hervorrufen, die wiederum zu der Entstehung eines Fehlerzustands in einem damit zusammenhängenden Arbeitsergebnis führt. Zum Beispiel kann eine Ungenauigkeit bei der Erhebung einer Anforderung zu einer fehlerhaften Beschreibung der Anforderung führen, die dann zu einem Programmierfehler und weiter zu einem Fehlerzustand im Code führt.

Wenn ein Fehlerzustand im Code ausgeführt wird, kann dies eine Fehlerwirkung hervorrufen, aber nicht notwendigerweise unter allen Umständen. Zum Beispiel erfordern einige Fehlerzustände sehr spezifische Eingaben oder Vorgaben, um eine Fehlerwirkung hervorzurufen, was nur selten oder niemals eintritt.

Fehlhandlungen können aus vielfältigen Gründen entstehen, wie z.B.:

- Zeitdruck
- Menschliche Fehlbarkeit

- Unerfahrene oder nicht ausreichend ausgebildete Projektbeteiligte
- Fehlkommunikation zwischen Projektbeteiligten, darunter Fehlkommunikation über Anforderungen und Entwurf
- Komplexität des Codes, des Entwurfs, der Architektur des zugrunde liegenden Problems, das gelöst werden soll, und/oder der genutzten Technologien
- Missverständnisse über systeminterne und systemübergreifende Schnittstellen, insbesondere wenn es eine Vielzahl solcher Schnittstellen gibt
- Neue, unbekannte Technologien

Zusätzlich zu Fehlerwirkungen, die durch Fehlerzustände im Code hervorgerufen werden, können Fehlerwirkungen auch durch Umweltbedingungen ausgelöst werden. Zum Beispiel können Strahlungen, elektromagnetische Felder und Beschmutzung zu Fehlerzuständen in Firmware führen oder die Ausführung von Software durch die Änderung von Hardwarebedingungen beeinflussen.

Nicht alle unerwarteten Testergebnisse sind Fehlerwirkungen. Falsch positive Ergebnisse können aufgrund von Fehlhandlungen in der Durchführung der Tests oder aufgrund von Fehlerzuständen in den Testdaten, der Testumgebung oder anderen Testmitteln oder aus anderen Gründen entstehen. Die umgekehrte Situation kann ebenfalls entstehen, wenn ähnliche Fehlhandlungen oder Fehlerzustände zu „falsch negativen“ Ergebnissen führen. „Falsch negative“ Ergebnisse sind Tests, die Fehlerzustände nicht entdecken, die sie hätten entdecken sollen; „falsch positive“ Ergebnisse sind berichtete Fehlerzustände, die tatsächlich keine Fehlerzustände sind.

1.2.4 Fehlerzustände, Grundursachen und Wirkungen

Die Grundursachen von Fehlerzuständen sind die frühesten Aktionen oder Bedingungen, die zur Entstehung der Fehlerzustände beigetragen haben. Fehlerzustände können analysiert werden, um ihre Grundursachen zu identifizieren und so das Auftreten von ähnlichen Fehlerzuständen in der Zukunft zu verhindern. Durch die Konzentration auf die bedeutendsten Grundursachen kann die Grundursachenanalyse zu Prozessverbesserungen führen, die das Auftreten einer Vielzahl von zukünftigen Fehlerzuständen verhindert.

Angenommen, falsche Zinszahlungen aufgrund einer einzigen Zeile falschen Codes führen zu Kundenbeschwerden. Der fehlerhafte Code wurde aufgrund einer User-Story geschrieben, die wegen des Missverständnisses des Product Owners darüber, wie der Zins zu berechnen sei, unklar formuliert war. Wenn ein Großteil der Fehlerzustände im Bereich der Zinsberechnung auftreten und diese Fehlerzustände ihre Grundursache in ähnlichen Missverständnissen haben, könnte der Product Owner im Bereich der Zinsberechnung geschult werden, um diese Art von Fehlern in der Zukunft zu vermeiden.

In diesem Beispiel zeigen die Kundenreklamationen Auswirkungen. Die falschen Zinszahlungen sind die direkten Fehlerwirkungen. Die falsche Berechnung im Code ist ein Fehlerzustand und dieser resultierte aus dem ursprünglichen Fehlerzustand, der Ungenauigkeit in der User-Story. Die Grundursache des ursprünglichen Fehlerzustands war ein Fehlen an Wissen auf Seiten des Product Owners, das dazu führte, dass er beim Schreiben der User-Story eine Fehlerhandlung beging. Der Prozess der Grundursachenanalyse wird im ISTQB-CTEL-TM und ISTQB-CTEL-ITP beschrieben.

1.3 Sieben Grundsätze des Testens

In den letzten 50 Jahren wurde eine Reihe von Grundsätzen für das Testen entwickelt, die generelle Leitlinien für alle Tests liefern.

1. Testen zeigt die Anwesenheit von Fehlerzuständen, nicht deren Abwesenheit

Testen kann zeigen, dass Fehlerzustände vorliegen, aber es kann nicht beweisen, dass es keine Fehlerzustände gibt. Testen reduziert die Wahrscheinlichkeit, dass noch unentdeckte Fehlerzustände in der

Software vorhanden sind, aber auch wenn keine Fehlerzustände gefunden werden, ist Testen kein Beweis für Korrektheit.

2. Vollständiges Testen ist nicht möglich

Ein vollständiger Test, bei dem alle möglichen Eingabewerte und deren Kombinationen unter Berücksichtigung aller unterschiedlichen Vorbedingungen ausgeführt werden, ist nicht durchführbar, mit Ausnahme von sehr trivialen Testobjekten. Anstatt zu versuchen, vollständig zu testen, sollten Risikoanalyse, Testverfahren und Prioritäten genutzt werden, um den Testaufwand zu konzentrieren.

3. Frühes Testen spart Zeit und Geld

Um Fehlerzustände früh zu finden, sollten sowohl statische als auch dynamische Testaktivitäten so früh wie möglich im Softwareentwicklungslebenszyklus gestartet werden. Frühes Testen wird oft als *Shift left* bezeichnet. Frühes Testen im Softwareentwicklungslebenszyklus hilft dabei, kostenintensive Änderungen zu reduzieren oder vollständig zu vermeiden (siehe Abschnitt *3.1 Grundlagen des statischen* Tests).

4. Häufung von Fehlerzuständen

Eine kleine Anzahl von Modulen enthält in der Regel die meisten Fehlerzustände, die während des Testens in der Phase vor Inbetriebnahme entdeckt werden, oder ist verantwortlich für die meisten der betrieblichen Fehlerwirkungen. Vorausgesagte Anhäufungen von Fehlerzuständen und die tatsächlich beobachteten Anhäufungen von Fehlerzuständen im Test oder im Betrieb sind ein wichtiger Beitrag zur Risikoanalyse, die genutzt wird, um den Testaufwand zu konzentrieren (wie in Grundsatz 2 erwähnt).

5. Vorsicht vor dem Pestizid-Paradoxon

Wenn die gleichen Tests immer wieder wiederholt werden, finden diese Tests irgendwann keine neuen Fehlerzustände mehr. Um neue Fehlerzustände zu finden, müssen bestehende Tests und Testdaten möglicherweise verändert werden und neue Tests geschrieben werden (Tests sind nicht länger effektiv im Erkennen von Fehlerzuständen, so wie Pestizide nach einer Weile nicht mehr effektiv in der Vernichtung von Insekten sind). In manchen Fällen, wie dem automatisierten Regressionstest, hat das Pestizid-Paradoxon einen vermeintlich positiven Effekt, der in der relativ geringen Anzahl von Regressionsfehlern liegt.

6. Testen ist kontextabhängig

Je nach Einsatzgebiet und Kontext ist das Testen anzupassen. Zum Beispiel wird sicherheitskritische industrielle Steuerungssoftware anders getestet als eine mobile E-Commerce-Applikation. Ein weiteres Beispiel ist das Testen in agilen Projekten, das anders durchgeführt wird als das Testen in einem Projekt mit sequenziellem Softwareentwicklungslebenszyklus (siehe Abschnitt 2.1 *Softwareentwicklungslebenszyklus-Modelle*).

7. Trugschluss: „Keine Fehler“ bedeutet ein brauchbares System

Einige Unternehmen erwarten, dass Tester alle denkbaren Tests durchführen und alle denkbaren Fehlerzustände finden können, aber die Grundsätze 2 und 1 lehren uns, dass dies unmöglich ist. Des Weiteren ist es ein Trugschluss (d.h. ein Irrglaube), zu erwarten, dass *allein* das Finden und Beheben einer großen Anzahl von Fehlerzuständen den Erfolg eines Systems sicherstellen werde. Beispielsweise kann trotz gründlicher Tests aller spezifizierten Anforderungen und Beheben aller gefundenen Fehlerzustände ein System erstellt werden, das schwer zu nutzen ist, dass die Bedürfnisse und Erwartungen der Benutzer nicht erfüllt oder das geringwertigere Qualität hat als vergleichbare Systeme.

Siehe Myers 2011, Kaner 2002, Weinberg 2008 und Beizer 1990 für Beispiele dieser und anderer Grundsätze des Testens.

1.4Testprozess

Es gibt nicht den einen universellen Softwaretestprozess, aber es gibt eine Reihe von gebräuchlichen Test-aktivitäten. Ohne diese Aktivitäten erreicht das Testen die festgelegten Ziele mit einer weit geringeren Wahrscheinlichkeit. Diese Menge von Testaktivitäten bildet den Testprozess. Der geeignete, spezifische Soft-waretestprozess in einer vorgegebenen Situation hängt von vielen Faktoren ab. Welche Testaktivitäten in diesem Testprozess beinhaltet sind, wie diese Aktivitäten eingesetzt werden und wann diese Aktivitäten stattfinden, kann in der Teststrategie eines Unternehmens behandelt werden.

1.4.1 Testprozess im Kontext

Kontextabhängige Faktoren, die den Testprozess in einer Organisation beeinflussen, sind u.a.:

- Verwendetes Softwareentwicklungslebenszyklus-Modell und Projektmethoden
- Mögliche Teststufen und Testarten
- Produkt- und Projektrisiken
- Geschäftsbereich
- Betriebliche Beschränkungen, u.a.:
  - Budget und Ressourcen
  - Fristen
  - Komplexität
  - Vertragliche und regulatorische Anforderungen
- Richtlinien und Praktiken des Unternehmens
- Geforderte interne und externe Standards

Die folgenden Abschnitte beschreiben allgemeine Aspekte von Testprozessen in Unternehmen in Bezug auf Folgendes:

- Testaktivitäten und Aufgaben
- Testarbeitsergebnisse
- Verfolgbarkeit zwischen Testbasis und Testarbeitsergebnissen

Es ist sehr nützlich, wenn die Testbasis (für jede in Betracht gezogene Stufe oder Art des Testens) messbar definierte Überdeckungskriterien hat. Die Überdeckungskriterien können effektiv als Key-Performance-Indicator (KPI) genutzt werden, um die Aktivitäten zu lenken, die das Erreichen der Ziele des Softwaretests aufzeigen (siehe Abschnitt *1.1.1 Typische Ziele des Testens*).

Für eine mobile Applikation kann die Testbasis zum Beispiel eine Liste von Anforderungen und eine Liste von zu unterstützenden mobilen Endgeräten enthalten. Jede Anforderung ist ein Element der Testbasis. Jedes zu unterstützende Endgerät ist ebenfalls ein Element der Testbasis. Die Überdeckungskriterien können erfordern, dass wenigstens ein Testfall für jedes Element der Testbasis entwickelt wird. Einmal ausgeführt, geben die Ergebnisse dieser Tests den Stakeholdern an, ob spezifizierte Anforderungen erfüllt sind und ob Fehlerwirkungen in unterstützten Endgeräten beobachtet wurden.

Die ISO-Norm (ISO/IEC/IEEE 29119-2) bietet weitere Informationen zu Testprozessen.

1.4.2 Testaktivitäten und Aufgaben

Ein Testprozess besteht aus den folgenden Hauptgruppen von Aktivitäten:

- Testplanung
- Testüberwachung und -steuerung
- Testanalyse
- Testentwurf
- Testrealisierung
- Testdurchführung
- Testabschluss

Jede Hauptgruppe von Aktivitäten besteht aus einzelnen Aktivitäten, die im Folgenden in den Unterabschnitten beschrieben werden. Jede einzelne Aktivität besteht ihrerseits aus mehreren Einzelaufgaben, die je nach Projekt oder Release variieren können.

Darüber hinaus werden diese Hauptgruppen von Aktivitäten oft iterativ realisiert, selbst wenn viele davon logisch aufeinander folgend erscheinen. Zum Beispiel beinhaltet agile Entwicklung kleine Iterationen im Softwareentwurf, Build und Test, die kontinuierlich durchgeführt werden und durch stetige Planung unterstützt werden. Also finden auch Testaktivitäten innerhalb dieses Softwareentwicklungsansatzes auf iterativer, kontinuierlicher Basis statt. Selbst bei sequenzieller Softwareentwicklung wird die angestrebte schrittweise logische Abfolge von Aktivitäten der Hauptgruppen sowohl Überlappung, Kombination, Parallelität oder Fortfall von einzelnen Aktivitäten der Hauptgruppen beinhalten, so dass eine Anpassung dieser Hauptaktivitäten innerhalb des System- und Projektkontexts grundsätzlich notwendig ist.

Testplanung

Testplanung beinhaltet Aktivitäten, die die Testziele definieren und den Ansatz für das Erreichen der Testziele innerhalb der Beschränkungen, die sich aus dem Umfeld ergeben, festlegen (z.B. das Spezifizieren geeigneter Testverfahren und Aufgaben sowie die Formulierung eines Testplans zur Einhaltung eines Termins). Testkonzepte können aufgrund von Feedback von Überwachungs- und Steuerungsaktivitäten überarbeitet werden. Testplanung wird in Abschnitt *5.2 Testplanung und -schätzung* näher erläutert.

Testüberwachung und -steuerung

Testüberwachung beinhaltet den andauernden Vergleich des tatsächlichen Fortschritts mit dem geplanten Fortschritt unter Nutzung jeglicher Überwachungsmetriken, die im Testkonzept definiert wurden. Teststeuerung beinhaltet das Ergreifen notwendiger Maßnahmen zur Erreichung der Ziele des Testkonzepts (das über den Zeitverlauf hinweg aktualisiert werden kann). Testüberwachung und -steuerung werden durch die Bewertung von Endekriterien unterstützt, die in einigen Softwareentwicklungslebenszyklusmodellen als Definition-of-Done bezeichnet werden (siehe ISTQB-AT). Zum Beispiel kann die Bewertung von Endekriterien für die Testdurchführung als Teil einer vorgegebenen Teststufe Folgendes beinhalten:

- Die Prüfung von Testergebnissen und -protokollen gegen spezifizierte Überdeckungskriterien
- Die Beurteilung des Grades der Komponenten- oder Systemqualität auf Grundlage der Testergebnisse und -protokolle
- Die Entscheidung, ob mehr Tests notwendig sind (z.B. ob Tests, die ursprünglich einen bestimmten Grad an Produktrisikoüberdeckung erreichen sollten und dies nicht erreicht haben, das Erstellen und Durchführen weiterer Tests erforderlich macht)

Der Testfortschritt anhand des Plans wird den Stakeholdern in Testfortschrittsberichten mitgeteilt. Diese enthalten auch Abweichungen vom Plan sowie Informationen, die jegliche Entscheidung unterstützt, die Tests zu beenden.

Testüberwachung und -steuerung werden in Abschnitt *5.3 Testüberwachung und -steuerung* näher erläutert.

Testanalyse

Während der Testanalyse wird die Testbasis analysiert, um testbare Features zu identifizieren und die entsprechenden Testbedingungen zu definieren. Das heißt, die Testanalyse bestimmt mit messbaren Überdeckungskriterien, „was zu testen ist".

Die Testanalyse beinhaltet die folgenden Hauptaktivitäten:

- Analyse der Testbasis, die für die in Betracht gezogene Teststufe geeignet ist, z.B.:
  - Anforderungsspezifikationen, wie Fachanforderungen, funktionale Anforderungen, Systemanforderungen, User-Stories[1], Epics[2], Anwendungsfall oder ähnliche Arbeitsergebnisse, die das gewünschte funktionale und nicht-funktionale Komponenten- oder Systemverhalten beschreiben
  - Entwurfs- und Realisierungsinformationen wie System- oder Softwarearchitekturdiagramme oder -dokumente, Entwurfsspezifikationen, Aufrufdiagramme, Modelldiagramme (z.B. UML- oder Entity-Relationship-Diagramme), Schnittstellenspezifikationen oder ähnliche Arbeitsergebnisse, die die Komponenten- oder Systemstruktur spezifizieren
  - Die Realisierung der Komponente oder des Systems selbst, darunter der Code, Datenbank-Metadaten und -abfragen sowie Schnittstellen
  - Risikoanalyseberichte, die funktionale, nicht-funktionale und strukturelle Aspekte der Komponente oder des Systems in Betracht ziehen
- Bewertung der Testbasis und der Testelemente zur Identifikation von unterschiedlichen Fehlerarten, wie z.B.:
  - Mehrdeutigkeiten
  - Auslassungen
  - Inkonsistenzen
  - Ungenauigkeiten
  - Widersprüche
  - Überflüssige Anweisungen
- Identifikation von Features und Feature-Sets, die getestet werden müssen.
- Definition und Priorisierung der Testbedingungen für jedes Feature auf Grundlage der Analyse der Testbasis und unter Berücksichtigung funktionaler, nicht-funktionaler und struktureller Merkmale, anderer fachlicher oder technischer Faktoren und des Risikograds

[1] Beschreiben die Anforderungen aus Sicht der Fachbereichsvertreter, aber auch der Entwickler und Tester und werden von diesen gemeinsam verfasst [informativer Bestandteil dieses Lehrplans, Details siehe ISTQB-CTFL-AT].

[2] Größere Ansammlungen von untereinander verbundenen Features oder eine Sammlung von Unter-Features, die zu einem einzigen, komplexen Feature zusammengeführt werden [informativer Bestandteil dieses Lehrplans, ISTQB-CTFL-AT].

- Erfassung von bidirektionaler Verfolgbarkeit zwischen jedem Element der Testbasis und den zugehörigen Testbedingungen (siehe Abschnitte *1.4.3 Testarbeitsergebnisse* und *1.4.4 Verfolgbarkeit zwischen der Testbasis und Testarbeitsergebnissen*)

Die Anwendung von Black-Box-, White-Box- und erfahrungsbasierten Testverfahren kann im Prozess der Testanalyse nützlich sein (siehe Kapitel 4), um die Wahrscheinlichkeit zu reduzieren, dass wichtige Testbedingungen nicht berücksichtigt werden, und um präzisere und genauere Testbedingungen zu definieren.

In manchen Fällen liefert die Testanalyse Testbedingungen, die als Testziele in Test-Chartas genutzt werden können. Test-Chartas sind typische Arbeitsergebnisse in einigen Arten des erfahrungsbasierten Testens (siehe Abschnitt *4.4.2 Exploratives Testen*). Wenn diese Testziele zur Testbasis verfolgbar sind, kann die Überdeckung, die während dieser erfahrungsbasierten Tests erzielt wird, gemessen werden.

Die Identifikation von Fehlerzuständen während der Testanalyse ist ein wichtiger potenzieller Nutzen, insbesondere dann, wenn kein anderer Reviewprozess genutzt wird und/oder der Testprozess eng mit dem Reviewprozess verbunden ist. Derartige Testanalyseaktivitäten verifizieren nicht nur, ob die Anforderungen konsistent, korrekt ausgedrückt und vollständig sind, sondern bewerten auch, ob die Anforderungen in angemessener Weise die Bedürfnisse der Kunden, Benutzer und anderer Stakeholder erfassen. Verfahren wie die verhaltensgetriebene Entwicklung („behavior driven development“, BDD) und abnahmetestgetriebene Entwicklung („acceptance test driven development”, ATDD), die die Generierung von Testbedingungen und Testfällen aus User-Stories und Abnahmekriterien vor dem Coding beinhalten. Diese Verfahren verifizieren, validieren und erkennen beispielsweise auch Fehlerzustände in den User-Stories und in den Abnahmekriterien (siehe ISTQB-CTFL-AT).

Testentwurf

Während des Testentwurfs werden die Testbedingungen in abstrakte Testfälle, Sets aus abstrakten Testfällen und andere Testmittel überführt. Die Testanalyse beantwortet also die Frage: „Was wird getestet?“, während der Testentwurf die Frage beantwortet: „Wie wird getestet?“.

Der Testentwurf beinhaltet die folgenden Hauptaktivitäten:

- Entwurf und Priorisierung von Testfällen und Sets an Testfällen
- Identifizierung von notwendigen Testdaten zur Unterstützung der Testbedingungen und Testfälle
- Entwurf der Testumgebung und Identifizierung benötigter Infrastruktur und Werkzeuge
- Erfassung der bidirektionalen Verfolgbarkeit zwischen der Testbasis, den Testbedingungen und den Testfällen (siehe Abschnitt *1.4.4 Verfolgbarkeit zwischen der Testbasis und Testarbeitsergebnissen*)

Die Überführung von Testbedingungen in Testfälle und Sets an Testfällen während des Testentwurfs beinhaltet oft den Einsatz von Testverfahren (siehe Kapitel 4).

Wie bei der Testanalyse kann auch der Testentwurf zur Identifizierung ähnlicher Fehlerarten in der Testbasis führen. Wie bei der Testanalyse ist die Identifizierung von Fehlerzuständen während des Testentwurfs ein wichtiger potenzieller Nutzen.

Testrealisierung

Während der Testrealisierung werden die Testmittel, die für die Testdurchführung notwendig sind, erstellt und/oder vervollständigt, unter anderem werden die Testfälle in Testabläufen in eine Reihenfolge gebracht. Der Testentwurf beantwortet also die Frage: „Wie wird getestet?”, während die Testrealisierung die Frage beantwortet: „Ist jetzt alles für die Durchführung der Tests bereit?“.

Die Testrealisierung beinhaltet die folgenden Hauptaktivitäten:

- Entwicklung und Priorisierung von Testabläufen und möglicherweise die Erstellung automatisierter Testskripte

- Erstellen von Testsuiten aus den Testabläufen und automatisierten Testskripten (falls es solche gibt)
- Anordnen der Testsuiten innerhalb eines Testausführungsplans in einer Art und Weise, die zu einer effizienten Testdurchführung führt (siehe Abschnitt *5.2.4 Testausführungsplan*)
- Aufbau der Testumgebung (unter anderem möglicherweise Testrahmen, Service-Virtualisierung, Simulatoren und andere Infrastrukturelemente) und Verifizierung, dass alles, was benötigt wird, korrekt aufgesetzt ist
- Vorbereitung von Testdaten und Sicherstellung, dass sie ordnungsgemäß in die Testumgebung geladen sind
- Verifizierung und Aktualisierung der bidirektionalen Verfolgbarkeit zwischen der Testbasis, den Testbedingungen, den Testfällen, den Testabläufen und den Testsuiten (siehe Abschnitt 1.4.4 Verfolgbarkeit zwischen der Testbasis und Testarbeitsergebnissen)

Testentwurf und Testrealisierungsaufgaben werden oft kombiniert.

In explorativen Tests und anderen Arten von erfahrungsbasierten Tests können Testentwurf und Realisierung als Teil der Testdurchführung auftreten und dokumentiert werden. Exploratives Testen kann auf Test-Chartas basieren (erstellt als Teil der Testanalyse) und explorative Tests werden unverzüglich durchgeführt, sobald sie entworfen und realisiert sind (siehe Abschnitt *4.4.2 Exploratives Testen*).

Testdurchführung

Während der Testdurchführung laufen Testsuiten in Übereinstimmung mit dem Testausführungsplan ab.

Die Testdurchführung beinhaltet die folgenden Hauptaktivitäten:

- Aufzeichnung der IDs und Versionen des Testelements/der Testelemente oder des Testobjekts, des Testwerkzeugs/der Testwerkzeuge und Testmittel
- Durchführung der Tests entweder manuell oder durch Nutzung von Testausführungswerkzeugen
- Vergleich der Istergebnisse mit den erwarteten Ergebnissen
- Analyse der Anomalien zur Feststellung ihrer wahrscheinlichen Ursachen (z.B. können Fehlerwirkungen aufgrund von Fehlerzuständen im Code entstehen, aber falsch positive Ergebnisse können ebenfalls auftreten; siehe Abschnitt *1.2.3 Fehlhandlungen, Fehlerzustände und Fehlerwirkungen*)
- Bericht über Fehlerzustände auf Grundlage der beobachteten Fehlerwirkungen (siehe Abschnitt *5.6 Fehlermanagement*)
- Aufzeichnung der Ergebnisse der Testdurchführung (z.B. bestanden, fehlgeschlagen, blockiert)
- Wiederholung von Testaktivitäten entweder als Ergebnis einer Aktion aufgrund einer Anomalie oder als Teil der geplanten Tests (z.B. Durchführung eines korrigierten Tests, Fehlernachtests und/oder Regressionstests)
- Verifizierung und Aktualisierung der bidirektionalen Verfolgbarkeit zwischen der Testbasis, den Testbedingungen, den Testfällen, den Testabläufen und den Testergebnissen

Testabschluss

Die Aktivitäten am Abschluss des Tests sammeln Daten aus beendeten Testaktivitäten, um Erfahrungen, Testmittel und andere relevante Informationen zu konsolidieren. Testabschlussaktivitäten finden an Projektmeilensteinen statt, z.B. wenn ein Softwaresystem freigegeben wird, ein Testprojekt abgeschlossen (oder abgebrochen) wird, eine agile Projektiteration beendet ist, eine Teststufe abgeschlossen ist oder ein Wartungsrelease abgeschlossen worden ist.

Der Testabschluss beinhaltet die folgenden Hauptaktivitäten:

- Prüfen, ob alle Fehlerberichte geschlossen sind, Eintragen von Change Requests oder Product-Backlog-Elementen für Fehlerzustände, die am Ende der Testdurchführung weiterhin nicht gelöst sind
- Erstellen eines Testabschlussberichts, der den Stakeholdern kommuniziert wird
- Finalisieren und Archivieren der Testumgebung, der Testdaten, der Testinfrastruktur und anderer Testmittel für spätere Wiederverwendung
- Übergabe der Testmittel an die Wartungsteams, andere Projektteams und/oder Stakeholder, die von deren Nutzung profitieren könnten
- Analyse der gewonnenen Erkenntnisse aus den abgeschlossenen Testaktivitäten, um notwendige Änderungen für zukünftige Iterationen, Produktreleases und Projekte zu bestimmen
- Nutzung der gesammelten Informationen zur Verbesserung der Testprozessreife

1.4.3 Testarbeitsergebnisse

Im Verlauf des Testprozesses werden Testarbeitsergebnisse erstellt. So wie es wesentliche Unterschiede in der Art und Weise gibt, wie Unternehmen die Testprozesse implementieren, gibt es auch wesentliche Unterschiede in den Arten der Arbeitsergebnisse, die während dieses Prozesses erstellt werden, in der Art und Weise, wie diese Arbeitsergebnisse aufgebaut und verwaltet werden, und welche Namen für diese Arbeitsergebnisse verwendet werden. Dieser Lehrplan richtet sich nach dem zuvor beschriebenen Testprozess und den Arbeitsergebnissen, die in diesem Lehrplan und im ISTQB-Glossar genannt sind. Die ISO-Norm (ISO/IEC/IEEE 29119-3) kann auch als Richtlinie für Testarbeitsergebnisse genutzt werden.

Viele der in diesem Abschnitt beschriebenen Testarbeitsergebnisse können durch die Nutzung von Testmanagementwerkzeugen und Fehlermanagementwerkzeugen erfasst und verwaltet werden (siehe Kapitel 6).

Arbeitsergebnisse der Testplanung

Arbeitsergebnisse der Testplanung beinhalten üblicherweise einen oder mehrere Testkonzepte. Das Testkonzept enthält Informationen über die Testbasis, auf die sich die anderen Testarbeitsergebnisse via Verfolgbarkeitsinformationen beziehen werden (siehe unten und Abschnitt *1.4.4 Verfolgbarkeit zwischen der Testbasis und Testarbeitsergebnissen*), sowie Endekriterien (oder Definition-of-Done), die während der Testüberwachung und -steuerung genutzt werden. Testkonzepte werden in Abschnitt *5.2 Testplanung und -schätzung* beschrieben.

Arbeitsergebnisse der Testüberwachung und -steuerung

Arbeitsergebnisse der Testüberwachung und -steuerung beinhalten üblicherweise verschiedene Arten von Testberichten, unter anderem kontinuierliche und/oder regelmäßig erstellte Testfortschrittsberichte und Testabschlussberichte, die an verschiedenen Abschlussmeilensteinen erstellt werden. Alle Testberichte sollten zielgruppenrelevante Details über den Testfortschritt zum Zeitpunkt des Berichts enthalten sowie die Testdurchführungsergebnisse zusammenfassen, sobald diese verfügbar sind.

Arbeitsergebnisse der Testüberwachung und -steuerung sollten auch Projektmanagementfragen ansprechen wie den abgeschlossenen Aufgaben, die Ressourcenzuordnung und -nutzung und den Aufwand.

Testüberwachung und -steuerung und die Arbeitsergebnisse, die während dieser Aktivitäten erstellt werden, werden in Abschnitt *5.3 Testüberwachung und -steuerung* dieses Lehrplans näher erläutert.

Arbeitsergebnisse der Testanalyse

Arbeitsergebnisse der Testanalyse beinhalten definierte und priorisierte Testbedingungen, von denen jede idealerweise bidirektional verfolgbar zu den spezifischen Elementen der Testbasis ist, die sie abdeckt. Für exploratives Testen kann die Testanalyse das Erstellen von Test-Chartas beinhalten. Die Testanalyse kann auch die Entdeckung und den Bericht von Fehlerzuständen in der Testbasis zur Folge haben.

Arbeitsergebnisse des Testentwurfs

Ergebnisse des Testentwurfs sind Testfälle und Sets von Testfällen, um die Testbedingungen abzudecken, die in der Testanalyse definiert wurden. Es ist oft hilfreich, abstrakte Testfälle ohne konkrete Werte für Eingabedaten und erwartete Ergebnisse zu entwerfen. Solche abstrakten Testfälle sind über mehrere Testzyklen mit unterschiedlichen konkreten Daten wiederverwendbar und dokumentieren dennoch in adäquater Weise den Umfang des Testfalls. Idealerweise ist jeder Testfall bidirektional verfolgbar zu den Testbedingungen, die er abdeckt.

Der Testentwurf resultiert auch

- im Entwurf und/oder der Identifizierung der notwendigen Testdaten
- dem Entwurf der Testumgebung
- der Identifizierung der Infrastruktur und Werkzeuge.

Allerdings schwankt der Umfang, in dem diese Ergebnisse dokumentiert werden, stark.

Arbeitsergebnisse der Testrealisierung

Arbeitsergebnisse der Testrealisierung beinhalten:

- Testabläufe und die Aneinanderreihung dieser Testabläufe
- Testsuiten
- Einen Testausführungsplan

Sobald die Testrealisierung abgeschlossen ist, kann idealerweise das Erreichen von Überdeckungskriterien, die im Testkonzept bestimmt wurden, über die bidirektionale Verfolgbarkeit zwischen Testabläufen und spezifischen Elementen der Testbasis durch die Testfälle und Testbedingungen gezeigt werden.

In manchen Fällen beinhaltet die Testrealisierung das Erstellen von Arbeitsergebnissen, die Werkzeuge nutzen oder von ihnen genutzt werden, wie Service-Virtualisierung und automatisierte Testskripte.

Die Testrealisierung kann auch in der Erstellung und Verifizierung von Testdaten und der Testumgebung resultieren. Die Vollständigkeit der Dokumentation der Verifizierungsergebnisse der Daten und/oder der Umgebung kann stark schwanken.

Die Testdaten dienen dazu, den Eingabewerten und erwarteten Ergebnissen von Testfällen konkrete Werte zuzuweisen. Solche konkreten Werte und explizite Anweisungen, wie diese konkreten Werte genutzt werden sollen, verwandeln abstrakte Testfälle in durchführbare konkrete Testfälle. Die gleichen abstrakten Testfälle können bei Durchführung in anderen Releases des Testobjekts andere Testdaten nutzen. Die konkreten erwarteten Ergebnisse, die mit den konkreten Testdaten verbunden sind, werden durch die Nutzung eines Testorakels identifiziert.

Im explorativen Testen können einige Arbeitsergebnisse des Testentwurfs und der Testrealisierung während der Testdurchführung erstellt werden. Der Umfang, mit dem explorative Tests (und ihre Verfolgbarkeit zu spezifischen Elementen der Testbasis) dokumentiert werden, kann allerdings stark schwanken.

Testbedingungen, die in der Testanalyse definiert sind, können in der Testrealisierung weiter verfeinert werden.

Arbeitsergebnisse der Testdurchführung

Arbeitsergebnisse der Testdurchführung beinhalten:

- Dokumentation des Status individueller Testfälle oder Testabläufe (z.B. ausführbar, bestanden, fehlgeschlagen, blockiert, geplant ausgelassen usw.)
- Fehlerbericht (siehe Abschnitt 5.6 Fehlermanagement)
- Dokumentation darüber, welches Testelement(e), Testwerkzeug(e) und Testmittel für die Tests benutzt wurden

Sobald die Testdurchführung vollständig ist, kann idealerweise der Status jedes Elements der Testbasis bestimmt und via bidirektionaler Verfolgbarkeit zu den zugehörigen Testabläufen berichtet werden. Zum Beispiel können wir sagen, welche Anforderungen alle geplanten Tests bestanden haben, welche Anforderungen aufgrund fehlgeschlagener Tests nicht erfolgreich verifiziert sind und/oder Fehlerzustände aufweisen und welche Anforderungen noch auf die Durchführung geplanter Tests warten. Dies erlaubt die Verifizierung, dass Überdeckungskriterien eingehalten wurden, und ermöglicht den Bericht von Testergebnissen in einer Art und Weise, dass sie für die Stakeholder verständlich sind.

Arbeitsergebnisse des Testabschlusses

Testabschlussarbeitsergebnisse beinhalten Testabschlussberichte, offene Punkte zur Verbesserung in nachfolgenden Projekten oder Iterationen, Change Requests oder Product-Backlog-Elemente und die finalisierten Testmittel.

1.4.4 Verfolgbarkeit zwischen der Testbasis und Testarbeitsergebnissen

Wie im letzten Abschnitt erwähnt, können Testarbeitsergebnisse und die Namen dieser Arbeitsergebnisse stark variieren. Unabhängig von diesen Variationen ist es wichtig für die Implementierung einer effektiven Testüberwachung und -steuerung, die Verfolgbarkeit durch den gesamten Testprozess zwischen allen Elementen der Testbasis und den verschiedenen Testarbeitsergebnissen, die diesem Element wie zuvor beschrieben zugeordnet sind, zu schaffen und zu erhalten. Über die Bewertung der Testüberdeckung hinaus unterstützt gute Verfolgbarkeit folgende Punkte:

- Die Auswirkungsanalyse von Änderungen
- Testen nachvollziehbarer zu machen
- IT-Governance-Kriterien zu erfüllen
- Die Verständlichkeit von Testfortschrittsberichten und Testabschlussberichten zu verbessern, um den Status der Elemente der Testbasis einzubeziehen (z.B. Anforderungen, die die Tests bestanden haben, Anforderungen, die in Tests fehlgeschlagen sind, und Anforderungen, für die Tests noch ausstehen)
- Bericht über die technischen Aspekte des Testens an die Stakeholder in einer Art und Weise, die sie verstehen können
- Bereitstellung von Informationen zur Beurteilung von Produktqualität, Prozessfähigkeit und Projektfortschritt gegenüber Geschäftszielen

Einige Testmanagementwerkzeuge liefern Modelle für Testarbeitsergebnisse, die einen Teil oder alle Testarbeitsergebnisse, die in diesem Abschnitt erwähnt sind, abdecken. Einige Unternehmen bauen ihre eigenen Managementsysteme, um die Arbeitsergebnisse zu verwalten und die Informationsverfolgbarkeit zu liefern, die sie benötigen.

1.5 Die Psychologie des Testens

Softwareentwicklung und auch Softwaretesten erfordert Menschen. Daher hat auch die Humanpsychologie wichtige Auswirkungen auf das Softwaretesten.

1.5.1 Humanpsychologie und Testen

Das Identifizieren von Fehlerzuständen während eines statischen Tests, wie eines Anforderungsreviews oder einer User-Story-Verfeinerungssession (auch Backlog-Refinement), oder das Identifizieren von Fehlerwirkungen während der Durchführung dynamischer Tests kann als Kritik am Produkt und an seinem Autor verstanden werden. Ein Element der Humanpsychologie namens Bestätigungsfehler kann dabei hinderlich sein, Informationen, die aktuellen Überzeugungen widersprechen, zu akzeptieren. Beispielsweise erwarten Entwickler, dass ihr Code korrekt ist, und die Bestätigungsfehler erschweren es ihnen, zu akzeptieren, dass der Code nicht korrekt ist. Zusätzlich zu diesem Bestätigungsfehler können auch andere kognitive Voreingenommenheiten Menschen daran hindern, die Informationen, die durch Tests bereitgestellt werden, zu verstehen oder zu akzeptieren. Außerdem ist es eine normale menschliche Eigenschaft, den Überbringer der schlechten Nachrichten für diese verantwortlich zu machen, und die Informationen, die das Testen hervorbringt, beinhalten häufig schlechte Nachrichten.

Aufgrund dieser psychologischen Faktoren könnten einige Menschen Testen als destruktive Aktivität ansehen, obwohl es in hohem Maße zum Projekterfolg und zur Produktqualität beiträgt (siehe Abschnitte *1.1 Was ist Testen?* und *1.2 Warum ist Testen notwendig?*). Um diese Vorurteile zu reduzieren, sollte die Information über Fehlerzustände und Fehlerwirkungen konstruktiv kommuniziert werden. Auf diese Weise können Spannungen zwischen den Testern und den Analysten, Product Ownern, Designern und Entwicklern reduziert werden. Dies trifft sowohl auf den statischen als auch auf den dynamischen Test zu.

Tester und Testmanager müssen gute soziale Kompetenzen haben, um in der Lage zu sein, effektive Kommunikation über Fehlerzustände, Fehlerwirkungen, Testergebnisse, Testfortschritte und Risiken zu gewährleisten und um positive Beziehungen zu den Kollegen aufzubauen. Gute Kommunikation beinhaltet die folgenden Beispiele:

- Beginne mit Zusammenarbeit statt mit Streit. Erinnere jeden Beteiligten an das gemeinsame Ziel eines Systems von höherer Qualität.
- Betone den Nutzen des Testens. Zum Beispiel können den Autoren Informationen über Fehlerzustände helfen, ihre Arbeitsergebnisse und ihre Fähigkeiten zu verbessern. Während des Testens gefundene und behobene Fehlerzustände ersparen dem Unternehmen Zeit und Geld und reduzieren das allgemeine Risiko für die Produktqualität.
- Kommuniziere Testergebnisse und andere Erkenntnisse in einer neutralen, faktenorientierten Weise, ohne denjenigen zu kritisieren, der das defekte Element erstellt hat. Schreibe objektive und tatsachenbasierte Fehlerberichte und Reviewbefunde.
- Versuche zu verstehen, wie die andere Person sich fühlt, und die Gründe dafür nachzuvollziehen, warum sie negativ auf die Information reagieren könnte.
- Lasse dir bestätigen, was die andere Person verstanden hat, was gesagt wurde und umgekehrt.

Typische Ziele des Testens wurden bereits zuvor behandelt (siehe *Abschnitt 1.1 Was ist Testen?*). Eine klare Definition des richtigen Sets an Testzielen hat wichtige psychologische Auswirkungen. Die meisten Menschen neigen dazu, ihre Pläne und ihr Verhalten an die Ziele anzupassen, die vom Team, vom Management und anderen Stakeholdern festgelegt wurden. Es ist auch wichtig, dass Tester mit einem Minimum an persönlicher Voreingenommenheit an diesen Zielen festhalten.

1.5.2 Denkweisen von Testern und Entwicklern

Entwickler und Tester denken oft unterschiedlich. Das Hauptziel der Entwicklung ist es, ein Produkt zu entwerfen und zu erstellen. Wie schon zuvor erwähnt, beinhalten die Ziele des Testens die Verifizierung und Validierung des Produkts, das Finden von Fehlerzuständen vor Inbetriebnahme usw. Dies sind unterschiedliche Sets an Zielen, die unterschiedliche Denkweisen erfordern. Diese Denkweisen zusammenzuführen hilft dabei, ein höheres Maß an Produktqualität zu erreichen.

Eine Denkweise spiegelt die Annahmen und bevorzugten Methoden eines Menschen für die Entscheidungsfindung und Problemlösung wider. Die Denkweise eines Testers sollte Neugier, professionellen Pessimismus, einen kritischen Blick, Detailgenauigkeit und die Motivation zu guter und positiver Kommunikation und Beziehungen beinhalten. Die Denkweise eines Testers wächst und reift mit steigender Erfahrung.

Die Denkweise eines Entwicklers kann einige der Elemente der Denkweise eines Testers enthalten, aber erfolgreiche Entwickler sind häufig stärker am Entwurf und an der Erstellung von Lösungen interessiert als daran, darüber nachzudenken, was an diesen Lösungen falsch sein könnte. Darüber hinaus erschwert ihnen das Phänomen Bestätigungsfehler, sich der von ihnen selbst begangenen Fehlhandlungen bewusst zu werden.

Mit der richtigen Denkweise können Entwickler ihren eigenen Code testen. Unterschiedliche Softwareentwicklungslebenszyklus-Modelle weisen oft verschiedene Möglichkeiten auf, wie die Tester und Testaktivitäten organisiert werden. Die Durchführung einiger der Testaktivitäten von unabhängigen Testern erhöht die Effektivität der Fehlerfindung, was insbesondere bei großen, komplexen oder sicherheitskritischen Systemen von hoher Bedeutung ist. Unabhängige Tester bringen eine Sichtweise ein, die sich von denen der Autoren der Arbeitsergebnisse (d.h. Business-Analysten, Product Owner, Designer und Entwickler) unterscheidet, da sie andere kognitive Voreingenommenheiten als die Autoren haben.

2. Testen im Softwareentwicklungslebenszyklus – 100 Minuten

Schlüsselbegriffe

Abnahmetest, änderungsbezogenes Testen, Alpha-Test, Auswirkungsanalyse, Benutzerabnahmetest, Beta-Test, betrieblicher Abnahmetest, Fehlernachtest, funktionaler Test, Integrationstest, kommerzielle Standardsoftware, Komponentenintegrationstest, Komponententest, nicht-funktionaler Test, Regressionstest, regulatorischer Abnahmetest, sequenzielles Entwicklungsmodell, Systemintegrationstest, Systemtest, Testart, Testbasis, Testfall, Testobjekt, Teststufe, Testumgebung, Testziel, vertraglicher Abnahmetest, Wartungstest, White-Box-Test

Lernziele für das Testen im Softwareentwicklungslebenszyklus

2.1 Softwareentwicklungslebenszyklus-Modelle

FL-2.1.1 (K2) Die Beziehungen zwischen Softwareentwicklungsaktivitäten und Testaktivitäten im Softwareentwicklungslebenszyklus erklären können

FL-2.1.2 (K1) Gründe identifizieren können, warum Softwareentwicklungslebenszyklus-Modelle an den Kontext des Projekts und an die Produktmerkmale angepasst werden müssen

2.2 Teststufen

FL-2.2.1 (K2) Die unterschiedlichen Teststufen unter den Aspekten der Testziele, Testbasis, Testobjekte, typischen Fehlerzustände und Fehlerwirkungen sowie der Testvorgehensweise und Verantwortlichkeiten vergleichen können

2.3 Testarten

FL-2.3.1 (K2) Funktionale, nicht-funktionale und White-Box-Tests vergleichen können

FL-2.3.2 (K1) Erkennen können, dass funktionale, nicht-funktionale und White-Box-Tests auf jeder Teststufe eingesetzt werden können

FL-2.3.3 (K2) Den Zweck von Fehlernachtests und Regressionstests vergleichen können

2.4 Wartungstest

FL-2.4.1 (K2) Auslöser für Wartungstests zusammenfassen können

FL-2.4.2 (K2) Den Einsatz der Auswirkungsanalyse im Wartungstest beschreiben können

2.1 Softwareentwicklungslebenszyklus-Modelle

Ein Softwareentwicklungslebenszyklus-Modell beschreibt Arten von Aktivitäten, die in jeder Phase eines Softwareentwicklungsprojekts durchgeführt werden, und wie diese Aktivitäten logisch und zeitlich zueinander in Beziehung stehen. Es gibt eine Reihe von unterschiedlichen Softwareentwicklungslebenszyklus-Modellen, von denen jedes Modell andere Ansätze für das Testen erfordert.

2.1.1 Softwareentwicklung und Softwaretesten

Damit Testaktivitäten angemessen erfolgen können, ist es in der Rolle eines Testers eine wichtige Kompetenz, sich mit den gängigen Softwareentwicklungslebenszyklus-Modellen auszukennen.

In jedem Softwareentwicklungslebenszyklus-Modell gibt es mehrere Merkmale für gutes Testen:

- Für jede Entwicklungsaktivität gibt es eine zugehörige Testaktivität.
- Jede Teststufe hat ihre stufenspezifischen Ziele für den Test.
- Für eine vorgegebene Teststufe beginnen Testanalyse und Testentwurf bereits während der zugehörigen Entwicklungsaktivität.
- Tester nehmen an Diskussionen zur Definition und Verfeinerung von Anforderungen und des Entwurfs teil. Darüber hinaus sind sie am Review von Arbeitsergebnissen (z.B. Anforderungen, Architekturdesign, User-Stories usw.) beteiligt, sobald erste Entwürfe dafür vorliegen.

Unabhängig davon, welches Softwareentwicklungslebenszyklus-Modell gewählt wird, sollten Testaktivitäten in Übereinstimmung mit dem Grundsatz des frühen Testens schon in frühen Phasen des Lebenszyklus beginnen.

Dieser Lehrplan kategorisiert verbreitete Softwareentwicklungslebenszyklus-Modelle wie folgt:

- Sequenzielle Entwicklungsmodelle
- Iterative und inkrementelle Entwicklungsmodelle

Ein sequenzielles Entwicklungsmodell beschreibt den Softwareentwicklungsprozess als linearen, sequenziellen Ablauf von Aktivitäten. Das bedeutet, dass jede Phase im Entwicklungsprozess erst beginnen sollte, wenn die vorherige Phase abgeschlossen ist. Theoretisch gibt es keine Überlappung der Phasen. In der Praxis ist es aber von Nutzen, aus der folgenden Phase eine frühe Rückmeldung zu erhalten.

Im Wasserfallmodell werden die Entwicklungsaktivitäten (z.B. Anforderungsanalyse, Entwurf, Implementierung, Test) nacheinander abgeschlossen. In diesem Modell findet das Testen erst statt, nachdem alle anderen Entwicklungsaktivitäten abgeschlossen sind.

Anders als das Wasserfallmodell integriert das (allgemeine) V-Modell den Testprozess in den Entwicklungsprozess und realisiert so den Grundsatz des frühen Testens. Außerdem beinhaltet das V-Modell Teststufen, die zu jeder zugehörigen Entwicklungsphase in Bezug stehen, was das frühe Testen noch weiter unterstützt (für eine Beschreibung der Teststufen siehe Abschnitt *2.2 Teststufen*). In diesem Modell findet die Durchführung der Tests, die zu der jeweiligen Teststufe gehören, sequenziell statt. In manchen Fällen kann es aber zu einer Überlappung kommen.

Sequenzielle Entwicklungsmodelle liefern Software, die das komplette Feature-Set enthält und benötigen typischerweise Monate oder Jahre für die Auslieferung an Stakeholder und Benutzer.

In der inkrementellen Entwicklung erfolgt die Festlegung der Anforderungen, der Entwurf, die Implementierung und das Testen eines Systems in Teilen, d.h., dass die Software-Features inkrementell anwachsen. Die Größe der Inkremente kann variieren. Einige Methoden haben größere und andere kleinere Inkremen-

te. Ein Inkrement kann dabei so klein sein wie eine einzige Veränderung in einer Maske der Benutzungsoberfläche oder wie eine neue Abfrageoption.

Iterative Entwicklung findet insbesondere dann statt, wenn Gruppen von Features zusammen in einer Reihe von Zyklen, oft mit einer festgelegten Dauer, spezifiziert, entworfen, implementiert und getestet werden können. Iterationen können sowohl Änderungen von Leistungsmerkmalen umfassen, die in früheren Iterationen entwickelt wurden, als auch Änderungen am eigentlichen Projektumfang. Jede Iteration liefert hierbei so lange eine lauffähige Software, die eine wachsende Teilmenge des gesamten Feature-Sets darstellt, bis die endgültige Software geliefert oder die Entwicklung beendet ist.

Beispiele hierfür sind u.a.:

- Rational Unified Process: Jede Iteration ist tendenziell relativ lang (z.B. zwei bis drei Monate) und die Inkremente der Features sind entsprechend groß (z.B. zwei oder drei Gruppen zusammengehörender Features).
- Scrum: Jede Iteration ist tendenziell eher kurz (z.B. Stunden, Tage oder einige Wochen) und die Inkremente der Features sind entsprechend klein (z.B. einige Verbesserungen und/oder zwei oder drei neue Features).
- Kanban: Implementierung mit Iterationen mit oder ohne festgelegten Länge. Sie liefert entweder eine einzige Verbesserung oder ein einziges vollständiges Feature oder Gruppen von Features zusammengefasst in einem Release.
- Spiralmodell: Beinhaltet das Schaffen von experimentellen Inkrementen, von denen einige in den folgenden Iterationen stark überarbeitet oder sogar aufgegeben werden müssen.

Komponenten oder Systeme, die unter Nutzung dieser Methoden entwickelt werden, beinhalten in der gesamten Entwicklung häufig Überlappung und iterative Teststufen. Idealerweise wird jedes Feature auf dem Weg zur Auslieferung auf mehreren Teststufen getestet. In manchen Fällen nutzen Teams eine kontinuierliche Auslieferung (continuous delivery) oder eine kontinuierliche Bereitstellung (continuous deployment) der Software. Beide beinhalten die Automatisierung von Tests auf mehreren Teststufen als Teil ihrer Delivery-Pipelines. Viele Entwicklungsbemühungen, die diese Methoden nutzen, beinhalten auch das Konzept der selbstorganisierenden Teams. Dieses Konzept verändert auch die Art und Weise, wie Tests organisiert werden, und die Beziehung zwischen Testern und Entwicklern.

Diese Methoden formen ein wachsendes System, das für die Endanwender Feature für Feature, Iteration für Iteration oder traditionell als Hauptrelease freigegeben werden kann. Je mehr das System wächst, desto wichtiger sind Regressionstests; unabhängig davon, ob die Softwareinkremente für Anwender freigegeben werden.

Im Gegensatz zu sequenziellen Modellen können iterative und inkrementelle Modelle in Wochen oder Tagen nutzbare Software liefern. Die vollständige Menge an Anforderungen können diese aber erst nach einer Reihe von Monaten oder sogar Jahren liefern.

Für weitere Informationen zum Softwaretesten im Kontext der agilen Entwicklung siehe ISTQB-AT, Black 2017, Crispin 2008 und Gregory 2015.

2.1.2 Softwareentwicklungslebenszyklus-Modelle im Kontext

Softwareentwicklungslebenszyklus-Modelle müssen im Kontext der Projekt- und Produktmerkmale ausgewählt und angepasst werden. Ein geeignetes Softwareentwicklungslebenszyklus-Modell sollte auf Grundlage des Projektziels, der Art des zu entwickelnden Produkts, der Geschäftsprioritäten (z.B. Time-to-Market) und der identifizierten Produkt- und Projektrisiken ausgewählt und angepasst werden. Zum Beispiel sollten sich die Entwicklung und das Testen eines kleinen internen Verwaltungssystems von der Entwicklung und dem Testen eines sicherheitskritischen Systems wie dem Bremssteuerungssystem eines Autos unterscheiden. Ein weiteres Beispiel sind organisatorische und kulturelle Probleme, die die Kommunikation zwischen Teammitgliedern erschweren können, was eine iterative Entwicklung behindert.

Abhängig vom Kontext des Projekts kann es notwendig sein, Teststufen und/oder Testaktivitäten zu kombinieren oder neu zu organisieren. Für die Integration einer kommerziellen Standardsoftware (commercial off-the-shelf, COTS) in ein größeres System kann der Käufer der Software zum Beispiel Interoperabilitätstests in der Systemintegrationsteststufe (z.B. Integration in die Infrastruktur und mit anderen Systemen) und auch in der Abnahmeteststufe (funktionale und nicht-funktionale Tests im Rahmen der Benutzer- und der betrieblichen Abnahmetests) durchführen (siehe Abschnitt *2.2 Teststufen* zur Beschreibung von Teststufen und Abschnitt *2.3 Testarten* zur Beschreibung von Testarten).

Darüber hinaus können Softwareentwicklungslebenszyklus-Modelle miteinander kombiniert werden. Beispielsweise kann ein V-Modell für die Entwicklung und das Testen eines Backend-Systems und seiner Integrationen genutzt werden, während ein agiles Entwicklungsmodell zur Entwicklung und zum Testen der Frontend-Benutzungsschnittstelle (UI) und ihrer Funktionalität eingesetzt wird. Ebenso kann Prototyping in einer frühen Projektphase verwendet werden und dann, sobald die experimentelle Phase abgeschlossen ist, ein inkrementelles Entwicklungsmodell.

„Internet der Dinge“-Systeme (Internet of Things, IoT), die aus vielen verschiedenen Objekten wie Geräten, Produkten und Diensten bestehen, wenden in der Regel eigenständige Softwareentwicklungslebenszyklus-Modelle für jedes Objekt an. Das stellt eine besondere Herausforderung für die Entwicklung von Versionen eines solchen Systems dar. Zusätzlich betont der Softwareentwicklungslebenszyklus solcher Projekte stärker die späten Phasen des Softwareentwicklungslebenszyklus nach deren Übergang in die betriebliche Nutzung (z.B. in den Phasen des Betriebs, der Aktualisierung und der Außerbetriebnahme).

Gründe, warum Softwareentwicklungsmodelle an den Kontext von Projekt- und Produkteigenschaften angepasst werden müssen, können sein:

- Unterschiedliche Produktrisiken von Systemen (komplexes oder einfaches Projekt)
- Viele Geschäftseinheiten können Teil eines Projekts oder Programms sein (Kombination aus sequentieller und agiler Entwicklung).
- Kurze Zeit bis zur Markteinführung eines Produkts (Zusammenführung von Teststufen und/oder Integration von Testarten in Teststufen)

2.2 Teststufen

Teststufen sind Gruppen von Testaktivitäten, die gemeinsam organisiert und verwaltet werden. Jede Teststufe ist eine Instanz des Testprozesses, der aus Aktivitäten besteht, die in Abschnitt *1.4 Testprozess* beschrieben wurden. Diese Testaktivitäten beziehen sich auf die Software einer festgelegten Entwicklungsstufe von einzelnen Einheiten oder Komponenten bis hin zu vollständigen Systemen oder, falls zutreffend, von Systemen von Systemen. Die Teststufen stehen mit anderen Aktivitäten innerhalb des Softwareentwicklungslebenszyklus in Verbindung. In diesem Lehrplan werden die folgenden Teststufen verwendet:

- Komponententest
- Integrationstest
- Systemtest
- Abnahmetest

Teststufen sind durch die folgenden Eigenschaften gekennzeichnet:

- Spezifische Ziele
- Testbasis, um mit Bezug darauf Testfälle abzuleiten
- Testobjekt (d.h., was wird getestet)
- Typische Fehlerzustände und Fehlerwirkungen

- Spezifische Ansätze und Verantwortlichkeiten

Für jede Teststufe ist eine passende Testumgebung erforderlich. Für den Abnahmetest ist beispielsweise eine produktionsähnliche Umgebung ideal, während die Entwickler im Komponententest üblicherweise ihre eigene Entwicklungsumgebung nutzen.

2.2.1 Komponententest

Ziele des Komponententests

Der Komponententest (auch Unit- oder Modultest genannt) konzentriert sich auf Komponenten, die einzeln testbar sind. Die Ziele des Komponententests beinhalten:

- Risikoreduktion
- Verifizierung, ob die funktionalen und nicht-funktionalen Verhaltensweisen der Komponente dem Entwurf und der Spezifikation entsprechen
- Schaffen von Vertrauen in die Qualität der Komponente
- Finden von Fehlerzuständen in der Komponente
- Verhindern, dass Fehlerzustände an höhere Teststufen weitergegeben werden

In manchen Fällen, insbesondere in inkrementellen und iterativen Entwicklungsmodellen (z.B. agilen Entwicklungsmodellen), in denen Codeänderungen stetig erfolgen, spielen automatisierte Komponentenregressionstests eine Schlüsselrolle. Diese sollen Vertrauen schaffen, dass Änderungen bestehende Komponenten nicht beschädigt haben.

Der Komponententest wird häufig isoliert vom Rest des Systems vorgenommen. In Abhängigkeit vom Softwareentwicklungslebenszyklus-Modell und vom System erfordert dies eventuell Mock-Objekte, Service-Virtualisierung, Rahmen, Platzhalter und Treiber. Der Komponententest kann die Funktionalität (z.B. die Korrektheit von Berechnungen), nicht-funktionale Eigenschaften (z.B. Suche nach Speicherengpässen) und strukturelle Merkmale (z.B. Entscheidungstests) abdecken.

Testbasis

Beispiele für Arbeitsergebnisse, die als Testbasis für Komponententests genutzt werden können, sind u.a.:

- Feinentwurf
- Code
- Datenmodelle
- Komponentenspezifikationen

Testobjekte

Gängige Testobjekte für Komponententests sind u.a.:

- Komponenten, Units oder Module
- Code und Datenstrukturen
- Klassen
- Datenbankmodule

Gängige Fehlerzustände und Fehlerwirkungen

Beispiele für gängige Fehlerzustände und Fehlerwirkungen für Komponententests sind u.a.:

- Nicht korrekte Funktionalität (z.B. nicht wie in den Entwurfsspezifikationen beschrieben)

- Datenflussprobleme
- Nicht korrekter Code und nicht korrekte Logik

Fehlerzustände werden in der Regel behoben, sobald sie gefunden werden, oftmals ohne formales Fehlermanagement. Wenn Entwickler allerdings Fehlerzustände berichten, liefert dies wichtige Informationen für die Grundursachenanalyse und die Prozessverbesserung.

Spezifische Ansätze und Verantwortlichkeiten

Komponententests werden üblicherweise von dem Entwickler durchgeführt, der den Code geschrieben hat. Zumindest aber erfordert dies den Zugang zum Code, der getestet werden soll. Entwickler können zwischen der Entwicklung einer Komponente und dem Finden und Beheben von Fehlerzuständen wechseln. Entwickler schreiben oft Tests und führen diese aus, nachdem sie den Code für eine Komponente entwickelt haben. Insbesondere in der agilen Entwicklung kann es jedoch auch sein, dass das Schreiben von automatisierten Komponententestfällen dem Schreiben von Anwendungscode vorangeht.

Nehmen wir zum Beispiel die testgetriebene Entwicklung (test driven development, TDD). Die testgetriebene Entwicklung ist hoch iterativ und basiert auf Zyklen zur Entwicklung automatisierter Testfälle, dann erfolgt die Entwicklung und Integration kleiner Teile von Code, gefolgt von der Durchführung von Komponententests, der Korrektur möglicher Probleme und der Restrukturierung (Refactoring) von Code. Dieser Prozess setzt sich fort, bis die Komponente vollständig erstellt und alle Komponententests erfolgreich abgeschlossen sind. Die testgetriebene Entwicklung ist ein Beispiel für den Ansatz „test first“. Obwohl die testgetriebene Entwicklung ihren Ursprung im eXtreme Programming (XP) hat, hat sie sich auch in andere Formen der agilen und in sequenzielle Lebenszyklen verbreitet (siehe ISTQB-AT).

2.2.2 Integrationstest

Ziele von Integrationstests

Integrationstests konzentrieren sich auf die Interaktion zwischen Komponenten oder Systemen. Ziele der Integrationstests sind u.a.:

- Risikoreduktion
- Verifizierung, ob die funktionalen und nicht-funktionalen Verhaltensweisen der Schnittstellen dem Entwurf und der Spezifikation entsprechen
- Vertrauen schaffen in die Qualität der Schnittstellen
- Fehlerzustände finden (die in den Schnittstellen selbst oder innerhalb der Komponenten oder des Systems liegen können)
- Verhindern, dass Fehlerzustände an höhere Teststufen weitergegeben werden

Wie beim Komponententest liefern automatisierte Integrationsregressionstests in manchen Fällen das Vertrauen, dass Änderungen bestehende Schnittstellen, Komponenten oder Systeme nicht beschädigt haben.

In diesem Lehrplan wird der Test von zwei Integrationsstufen beschrieben, der auf Testobjekte unterschiedlicher Größe ausgeübt werden kann:

- Komponentenintegrationstests konzentrieren sich auf die Interaktionen und die Schnittstellen zwischen integrierten Komponenten. Komponentenintegrationstests werden nach Komponententests durchgeführt und sind generell automatisiert. In der iterativen und inkrementellen Entwicklung sind Integrationstests in der Regel Teil des kontinuierlichen Integrationsprozesses (continuous integration).
- Systemintegrationstests konzentrieren sich auf die Interaktionen und Schnittstellen zwischen Systemen, Paketen und Microservices. Systemintegrationstests können auch Interaktionen und

Schnittstellen abdecken, die von externen Organisationen bereitgestellt werden (z.B. Webservices). In diesem Fall hat die entwickelnde Organisation keinen Einfluss auf die externen Schnittstellen. Dies kann zu verschiedenen Herausforderungen im Testen führen (z.B. Sicherstellen, dass testblockierende Fehlerzustände im Code der externen Organisation behoben sind, Bereitstellen von Testumgebungen usw.). Der Systemintegrationstest kann nach den Systemtests oder parallel zu andauernden Systemtestaktivitäten stattfinden (sowohl in der sequenziellen Entwicklung als auch in der iterativen und inkrementellen Entwicklung).

Testbasis

Beispiele für Arbeitsergebnisse, die als Testbasis für Integrationstests genutzt werden können, sind u.a.:

- Software- und Systementwurf
- Sequenzdiagramme
- Spezifikationen von Schnittstellen und Kommunikationsprotokollen
- Anwendungsfälle
- Architektur auf Komponenten- oder Systemebene
- Workflows
- Externe Schnittstellendefinitionen

Testobjekte

Gängige Testobjekte für Integrationstests sind u.a.:

- Subsysteme
- Datenbanken
- Infrastruktur
- Schnittstellen
- APIs
- Microservices

Typische Fehlerzustände und Fehlerwirkungen

Beispiele für typische Fehlerzustände und Fehlerwirkungen für Komponentenintegrationstests sind u.a.:

- Falsche Daten, fehlende Daten oder falsche Datenverschlüsselung
- Falsche Reihenfolge oder fehlerhafte zeitliche Abfolge von Schnittstellenaufrufen
- Falsch angepasste Schnittstellen
- Fehlerwirkungen in der Kommunikation zwischen Komponenten
- Nicht behandelte oder nicht ordnungsgemäß behandelte Fehlerwirkungen in der Kommunikation zwischen den Komponenten
- Nicht korrekte Annahmen über die Bedeutung, Einheiten oder Grenzen der Daten, die zwischen den Komponenten hin- und hergereicht werden

Beispiele typischer Fehlerzustände und Fehlerwirkungen für Systemintegrationstests sind u.a.:

- Inkonsistente Nachrichtenstrukturen zwischen den Systemen
- Falsche Daten, fehlende Daten oder falsche Datenverschlüsselung

- Falsch angepasste Schnittstellen
- Fehlerwirkungen in der Kommunikation zwischen den Systemen
- Nicht behandelte oder nicht ordnungsgemäß behandelte Fehlerwirkungen in der Kommunikation zwischen den Systemen
- Falsche Annahmen über die Bedeutung, Einheiten oder Grenzen der Daten, die zwischen den Systemen hin- und hergereicht werden
- Fehlende Konformität mit erforderlichen Richtlinien zur IT-Sicherheit

Spezifische Ansätze und Verantwortlichkeiten

Komponentenintegrationstests und Systemintegrationstests sollten sich auf die Integration selbst konzentrieren. Zum Beispiel, wenn das Modul A mit dem Modul B integriert wird, sollten sich die Tests auf die Kommunikation zwischen diesen Modulen konzentrieren und nicht auf die Funktionalität der einzelnen Module, da diese in den Komponententests abgedeckt sein sollte. Wenn System X mit System Y integriert wird, sollten sich die Tests auf die Kommunikation zwischen den Systemen konzentrieren, nicht auf die Funktionalität der einzelnen Systeme, da diese in den Systemtests abgedeckt sein sollte. Funktionale, nicht-funktionale und strukturelle Testarten sind geeignet.

Komponentenintegrationstests liegen häufig in der Verantwortung der Entwickler. Systemintegrationstests liegen im Allgemeinen in der Verantwortung der Tester. Idealerweise sollten Tester, die Systemintegrationstests durchführen, die Systemarchitektur verstehen und die Integrationsplanung beeinflusst haben.

Wenn Integrationstests und die Integrationsstrategie geplant werden, bevor Komponenten oder Systeme entwickelt werden, können diese Komponenten oder Systeme in der Reihenfolge entwickelt werden, die für das effizienteste Testen erforderlich ist. Systematische Integrationsstrategien können auf der Systemarchitektur (z.B. Top-down und Bottom-up), auf funktionellen Aufgaben, auf der Reihenfolge der Transaktionsverarbeitung oder auf anderen Aspekten des Systems oder der Komponenten basieren. Um die Isolation von Fehlerzuständen zu vereinfachen und Fehlerzustände früh zu erkennen, sollte die Integration normalerweise inkrementell erfolgen (d.h. nur eine kleine Anzahl von zusätzlichen Komponenten oder Systemen zur gleichen Zeit) statt in einer Art „Big Bang“ (d.h. Integration aller Komponenten oder Systeme in einem einzigen Schritt). Eine Risikoanalyse der komplexesten Schnittstellen kann dabei helfen, die Integrationstests zielgerichtet einzusetzen.

Je größer der Umfang der Integration, desto schwieriger wird es, die Fehlerzustände in einer spezifischen Komponente oder einem spezifischen System zu isolieren. Dies führt zu einem höheren Risiko und einem größeren Zeitaufwand für die Fehlerbehebung. Das ist ein Grund dafür, dass kontinuierliche Integration, bei der Software auf Komponentenbasis integriert wird (d.h. funktionale Integration), zur gängigen Vorgehensweise geworden ist. Eine derartige kontinuierliche Integration (continuous integration) beinhaltet häufig automatisierte Regressionstests, idealerweise auf mehreren Teststufen.

2.2.3 Systemtest

Ziele des Systemtests

Systemtests konzentrieren sich auf das Verhalten und die Fähigkeiten des Systems oder Produkts. Dies geschieht oft unter Berücksichtigung der End-to-End-Aufgaben, die das System leisten kann, und der nicht-funktionalen Verhaltensweisen, die bei der Verarbeitung dieser Aufgaben zu Tage treten. Ziele des Systemtests sind u.a.:

- Risikoreduktion
- Verifizierung, ob die funktionalen und nicht-funktionalen Verhaltensweisen des Systems dem Entwurf und den Spezifikationen entsprechen
- Validierung, dass das System vollständig ist und wie erwartet funktionieren wird

- Vertrauen in die Qualität des Systems als Ganzes schaffen
- Finden von Fehlerzuständen
- Verhindern, dass Fehlerzustände an höhere Teststufen oder in die Produktion weitergegeben werden

Für einige Systeme kann auch die Verifizierung der Datenqualität ein Ziel sein. Wie beim Komponententest und beim Integrationstest bieten in einigen Fällen automatisierte Systemregressionstests die Gewissheit, dass Änderungen nicht die bestehenden Features oder End-to-End-Fähigkeiten beeinträchtigt haben. Systemtests liefern häufig Informationen, die von Stakeholdern für Freigabeentscheidungen genutzt werden. Systemtests können auch zur Erfüllung rechtlicher oder regulatorischer Anforderungen oder Standards notwendig sein.

Die Testumgebung sollte idealerweise der finalen Ziel- oder der Produktivumgebung entsprechen.

Testbasis

Beispiele für Arbeitsergebnisse, die als Testbasis für Systemtests genutzt werden können, sind u.a.:

- System- und Softwareanforderungsspezifikationen (funktional und nicht-funktional)
- Risikoanalyseberichte
- Anwendungsfälle
- Epics und User-Stories
- Modelle des Systemverhaltens
- Zustandsdiagramme
- System- und Benutzeranleitungen

Testobjekte

Gängige Testobjekte für Systemtests sind u.a.:

- Anwendungen
- Hardware/Softwaresysteme
- Betriebssysteme
- Systeme unter Test (SUT)
- Systemkonfiguration und Konfigurationsdaten

Typische Fehlerzustände und Fehlerwirkungen

Beispiele für typische Fehlerzustände und Fehlerwirkungen für Systemtests sind u.a.:

- Falsche Berechnungen
- Falsche oder unerwartete funktionale oder nicht-funktionale Systemverhaltensweisen
- Falsche Kontroll- und/oder Datenflüsse innerhalb des Systems
- Versagen bei der korrekten oder vollständigen Ausführung von funktionalen End-to-End-Aufgaben
- Versagen des Systems bei der ordnungsgemäßen Arbeit in der/den Systemumgebung/en
- Das System funktioniert nicht wie in den System- oder Benutzeranleitungen beschrieben

Spezifische Ansätze und Verantwortlichkeiten

Systemtests sollten sich auf das allgemeine End-to-End-Verhalten des Systems als Ganzes konzentrieren, sowohl in funktionaler als auch in nicht-funktionaler Hinsicht. Systemtests sollten die Verfahren nutzen, die am besten für die zu testenden Aspekte des Systems geeignet sind (siehe Kapitel 4). Beispielsweise kann eine Entscheidungstabelle erstellt werden, um zu verifizieren, ob ein funktionales Verhalten den Beschreibungen in den Geschäftsregeln entspricht.

Systemtests werden in der Regel von unabhängigen Testern durchgeführt, die sich stark auf Spezifikationen stützen. Fehlerzustände in Spezifikationen (z.B. fehlende User-Stories, falsch benannte Fachanforderungen usw.) können zu einem Verständnisproblem oder Unstimmigkeiten über das erwartete Systemverhalten führen. Derartige Situationen können zu „falsch positiven“ und „falsch negativen“ Ergebnissen führen. Dies verschwendet Zeit und reduziert entsprechend die Effektivität der Fehleridentifikation. Das frühe Einbeziehen von Testern in User-Story-Verfeinerungen (Refinements) oder statische Testaktivitäten, wie Reviews, helfen dabei, das Auftreten solcher Situationen zu reduzieren.

2.2.4 Abnahmetest

Ziele des Abnahmetests

Der Abnahmetest konzentriert sich wie der Systemtest typischerweise auf das Verhalten und die Fähigkeiten eines gesamten Systems oder Produkts. Ziele des Abnahmetests sind u.a.:

- Vertrauen in die Qualität des Systems als Ganzen aufbauen
- Validieren, ob das System vollständig ist und wie erwartet funktionieren wird
- Verifizieren, ob funktionale und nicht-funktionale Verhaltensweisen des Systems den Spezifikationen entsprechen

Der Abnahmetest kann Informationen bereitstellen, mit denen sich die Bereitschaft des Systems für den Einsatz und die Nutzung durch den Kunden (Endanwender) beurteilen lässt. Im Abnahmetest können Fehlerzustände gefunden werden, aber das Finden von Fehlerzuständen ist häufig nicht das Ziel und das Finden einer großen Anzahl von Fehlerzuständen im Abnahmetest wird in manchen Fällen als großes Projektrisiko angesehen. Der Abnahmetest kann auch notwendig sein, um rechtliche oder regulatorische Anforderungen oder Standards zu erfüllen.

Häufige Ausprägungen von Abnahmetests sind u.a.:

- Benutzerabnahmetest
- Betrieblicher Abnahmetest
- Vertraglicher und regulatorischer Abnahmetest
- Alpha- und Beta-Test

Jeder dieser Ausprägungen wird in den folgenden vier Unterabschnitten genauer beschrieben.

Benutzerabnahmetest (User Acceptance Testing (UAT))

Der Abnahmetest des Systems ist üblicherweise darauf konzentriert, die Bereitschaft des Systems zur Nutzung durch Benutzer in einer realen oder simulierten Betriebsumgebung zu validieren. Das Hauptziel ist es, Vertrauen darin zu schaffen, dass die Benutzer das System so nutzen können, dass es ihre Bedürfnisse und Anforderungen erfüllt und ihre Geschäftsprozesse mit einem Minimum an Schwierigkeiten, Kosten und Risiken ausführt.

Betrieblicher Abnahmetest (Operational Acceptance Testing (OAT))

Der Abnahmetests des Systems durch Mitarbeiter des Betriebs oder Systemadministratoren wird üblicherweise in einer (simulierten) Produktivumgebung durchgeführt. Der Fokus des Tests liegt dabei auf betrieblichen Aspekten, u.a.:

- Testen von Backups und Wiederherstellungen
- Installieren, Deinstallieren und Aktualisieren
- Notfallwiederherstellung (Disaster-Recovery)
- Benutzerverwaltung
- Wartungsaufgaben
- Datenlade- und Migrationsaufgaben
- Prüfen von IT-Sicherheitsschwachstellen
- Performanztest

Das Hauptziel von betrieblichen Abnahmetests ist es, Vertrauen darin aufzubauen, dass die Betreiber oder Systemadministratoren das System in der betrieblichen Umgebung ordnungsgemäß für die Benutzer funktionsfähig halten können, selbst unter außergewöhnlichen oder schwierigen Bedingungen.

Vertraglicher oder regulatorischer Abnahmetest

Der vertragliche Abnahmetest wird aufgrund von vertraglichen Abnahmekriterien für die Herstellung von kundenspezifisch entwickelter Software durchgeführt. Abnahmekriterien sollten zu dem Zeitpunkt definiert werden, an dem sich die Vertragsparteien auf den Vertrag einigen. Vertragliche Abnahmetests werden häufig durch Benutzer oder unabhängige Tester durchgeführt.

Regulatorische Abnahmetests werden gegen Regularien durchgeführt, die eingehalten werden müssen, beispielsweise staatliche, gesetzliche oder Vorschriften zur funktionalen Sicherheit. Regulatorische Abnahmetests werden häufig von Benutzern oder unabhängigen Testern durchgeführt. Manchmal werden die Ergebnisse von Aufsichtsbehörden bestätigt oder auditiert.

Das Hauptziel von vertraglichen oder regulatorischen Abnahmetests ist es, Vertrauen darin aufzubauen, dass die vertragliche oder regulatorische Konformität gegeben ist.

Alpha- und Beta-Tests

Alpha- und Beta-Tests werden üblicherweise von Entwicklern von kommerzieller Standardsoftware genutzt, die Rückmeldungen von ihren potenziellen oder bestehenden Benutzern, Kunden und/oder Betreibern erhalten wollen, bevor die Software auf den Markt kommt. Alpha-Tests werden auf Seiten des entwickelnden Unternehmens vorgenommen, nicht vom Entwicklungsteam, sondern von potenziellen oder bestehenden Kunden und/oder Betreibern oder von einem unabhängigen Testteam. Beta-Tests werden von potenziellen oder bestehenden Kunden und/oder Betreibern an ihren eigenen Standorten durchgeführt. Beta-Tests können nach Alpha-Tests erfolgen oder auch ohne, dass ein vorausgehender Alpha-Test stattfand.

Ein Ziel des Alpha- und Beta-Testens ist es, Vertrauen bei potenziellen oder bestehenden Kunden und/oder Betreibern darüber zu schaffen, dass sie das System unter normalen alltäglichen Umständen in der Produktivumgebung(en) nutzen können, um ihre Ziele bequem und mit geringsten Kosten und Risiken zu erreichen. Ein weiteres Ziel kann das Finden von Fehlerzuständen sein, die sich auf die Bedingungen und Umgebungen beziehen, in denen das System genutzt werden wird, insbesondere dann, wenn diese Bedingungen und Umgebungen durch das Entwicklungsteam nur schwer nachzubilden sind.

Testbasis

Beispiele für Arbeitsergebnisse, die als Testbasis für jegliche Form von Abnahmetests verwendet werden können, sind u.a.:

- Geschäftsprozesse
- Benutzer- oder Fachanforderungen
- Vorschriften, rechtliche Verträge und Standards
- Anwendungsfälle und/oder User Stories
- Systemanforderungen
- System- oder Benutzerdokumentation
- Installationsverfahren
- Risikoanalyseberichte

Zusätzlich können eines oder mehrere der folgenden Arbeitsergebnisse als Testbasis für die Ableitung von Testfällen für betriebliche Abnahmetests genutzt werden:

- Sicherungs- und Wiederherstellungsverfahren
- Disaster-Recovery-Verfahren
- Nicht-funktionale Anforderungen
- Betriebsdokumentation
- Bereitstellungs- und Installationsanweisungen
- Performanzziele
- Datenbankpakete
- Standards oder Vorschriften bzgl. IT-Sicherheit

Typische Testobjekte

Typische Testobjekte für jegliche Form von Abnahmetests sind u.a.:

- System unter Test (SUT)
- Systemkonfigurationen und Konfigurationsdaten
- Geschäftsprozesse des vollintegrierten Systems
- Wiederherstellungssysteme und Hot Sites (für Tests zur Business-Continuity (Betriebskontinuität) und Notfallwiederherstellung)
- Betriebs- und Wartungsprozesse
- Formulare
- Berichte
- Bestehende und konvertierte Produktionsdaten

Typische Fehlerzustände und Fehlerwirkungen

Beispiele für typische Fehlerzustände für jegliche Form von Abnahmetests sind u.a.:

- Systemworkflows erfüllen nicht die Fach- oder Benutzeranforderungen

- Geschäftsregeln wurden nicht korrekt umgesetzt
- Das System erfüllt nicht die vertraglichen oder regulatorischen Anforderungen
- Nicht-funktionale Fehlerwirkungen wie IT-Sicherheitsschwachstellen, nicht angemessene Performanz unter hoher Last oder nicht ordnungsgemäßer Betrieb auf einer unterstützten Plattform

Spezifische Ansätze und Verantwortlichkeiten

Der Abnahmetest liegt häufig in der Verantwortung der Kunden, Fachanwender, Product Owner oder Betreiber eines Systems. Andere Stakeholder können ebenfalls mit einbezogen werden.

Der Abnahmetest wird oft als die letzte Stufe in einem sequenziellen Entwicklungslebenszyklus verstanden, er kann aber auch zu anderen Zeitpunkten stattfinden, z.B.:

- Der Abnahmetests eines kommerziellen Standardsoftwareprodukts kann zum Zeitpunkt der Installation oder Integration stattfinden
- Der Abnahmetest einer neuen funktionalen Verbesserung kann vor dem Systemtest stattfinden

In der iterativen Entwicklung können Projektteams verschiedene Formen der Abnahmetests während und am Ende einer Iteration vornehmen, wie die, die sich auf die Verifizierung eines neuen Features gegenüber seinen Abnahmekriterien konzentrieren, oder die, die sich darauf konzentrieren, zu validieren, dass ein neues Feature die Bedürfnisse der Benutzer erfüllt. Darüber hinaus können Alpha- und Beta-Tests entweder am Ende jeder Iteration, nach Abschluss jeder Iteration oder nach einer Serie von Iterationen stattfinden. Benutzerabnahmetests, betriebliche Abnahmetests, regulatorische Abnahmetests und vertragliche Abnahmetests können ebenfalls entweder zum Abschluss jeder Iteration, nach Abschluss jeder Iteration oder nach einer Serie von Iterationen stattfinden.

2.3 Testarten

Eine Testart ist eine Gruppe von Testaktivitäten, die darauf abzielt, spezifische Merkmale eines Softwaresystems oder eines Teils eines Systems auf der Grundlage spezifischer Testziele zu testen. Solche Ziele können u.a. sein:

- Bewertung funktionaler Qualitätsmerkmale wie Vollständigkeit, Korrektheit und Angemessenheit
- Bewertung nicht-funktionaler Qualitätsmerkmale wie Zuverlässigkeit, Performanz, IT-Sicherheit, Kompatibilität und Gebrauchstauglichkeit (Usability)
- Bewertung, ob die Struktur oder Architektur der Komponente oder des Systems korrekt und vollständig ist und den Spezifikationen entspricht
- Bewertung der Auswirkungen von Änderungen wie die Bestätigung, dass Fehlerzustände behoben wurden (Fehlernachtests), und die Suche nach nicht gewünschten Änderungen im Verhalten, die sich aus Software- oder Umgebungsänderungen ergeben (Regressionstests)

2.3.1 Funktionale Tests

Funktionales Testen eines Systems beinhaltet Tests, die Funktionen bewerten, die das System ausführen soll. Funktionale Anforderungen können in Arbeitsergebnissen wie fachliche Anforderungsspezifikation, Epics, User-Stories, Anwendungsfällen oder funktionalen Spezifikationen beschrieben sein. Sie können allerdings auch undokumentiert sein. Die Funktionen sind das, „was" das System tun sollte.

Funktionale Tests sollten in allen Teststufen durchgeführt werden (z.B. können Komponententests auf einer Komponentenspezifikation basieren), obwohl der Fokus auf jeder Stufe ein anderer ist (siehe Abschnitt *2.2 Teststufen*).

Funktionales Testen betrifft das Verhalten der Software. So können Black-Box-Verfahren genutzt werden, um Testbedingungen und Testfälle für die Funktionalität der Komponente oder des Systems abzuleiten (siehe Abschnitt *4.2 Black-Box-Testverfahren*).

Die Gründlichkeit von funktionalen Tests kann durch die funktionale Überdeckung gemessen werden. Die funktionale Überdeckung ist der Grad, in dem eine gewisse Funktionalität durch Testen ausgeführt wurde, und wird als Prozentsatz der Arten ausgedrückt, die abgedeckt sind. Durch die Nutzung der Verfolgbarkeit zwischen Tests und funktionalen Anforderungen kann beispielsweise der Prozentsatz dieser Anforderungen, die durch die Tests angesprochen werden, berechnet werden. So können potenzielle Überdeckungslücken identifiziert werden.

Funktionaler Testentwurf und die Durchführung funktionaler Tests können spezielle Fähigkeiten oder Wissen erfordern, wie die Kenntnis des bestimmten Fachproblems, das die Software löst (z.B. geologische Modellierungssoftware für die Öl- und Gasindustrie).

2.3.2 Nicht-funktionale Tests

Nicht-funktionale Tests eines Systems bewerten Merkmale von Systemen und Software wie Gebrauchstauglichkeit, Performanz oder IT-Sicherheit. In der ISO-Norm ISO/IEC 25010 findet sich eine Klassifizierung von Softwarequalitätsmerkmalen. Nicht-funktionale Tests sind die Tests, die zeigen, “wie gut“ das System sich verhält.

Im Gegensatz zu gängigen Fehleinschätzungen können und sollten nicht-funktionale Tests in den meisten Fällen in allen Teststufen und das so früh wie möglich durchgeführt werden. Das späte Entdecken von nicht-funktionalen Fehlerzuständen kann für den Erfolg eines Projekts extrem gefährlich sein.

Um Testbedingungen und Testfälle für nicht-funktionale Tests abzuleiten, können Black-Box-Verfahren (siehe Abschnitt *4.2 Black-Box-Testverfahren*) genutzt werden. Beispielsweise kann die Grenzwertanalyse genutzt werden, um die Stressbedingungen für Performanztests festzulegen.

Die Gründlichkeit von nicht-funktionalen Tests kann durch die nicht-funktionale Überdeckung gemessen werden. Nicht-funktionale Überdeckung ist der Grad, zu dem einige Arten von nicht-funktionalen Elementen durch Tests ausgeführt wurden, und wird als Prozentsatz der Arten der Elemente ausgedrückt, die abgedeckt sind. Beispielsweise lässt sich durch die Nutzung der Verfolgbarkeit zwischen Tests und unterstützten Endgeräten bei mobilen Applikationen der Prozentsatz der Endgeräte, die durch die Kompatibilitätstests betrachtet werden, berechnen, wodurch sich möglicherweise Überdeckungslücken aufdecken lassen.

Nicht-funktionaler Testentwurf und die Durchführung nicht-funktionaler Tests können spezielle Fähigkeiten oder Kenntnisse erfordern, wie die Kenntnis der zugrunde liegenden Schwächen eines Entwurfs oder einer Technologie (z.B. IT-Sicherheitsschwachstellen, die mit bestimmten Programmiersprachen verbunden sind) oder einer bestimmten Benutzergruppe (z.B. die Personas[3] von Verwaltungssystemen für Gesundheitseinrichtungen).

In den ISTQB-ATA, ISTQB-ATTA und ISTQB-SEC sowie in anderen ISTQB-Spezialistenmodulen finden sich weitere Details bezüglich des Testens von nicht-funktionalen Qualitätsmerkmalen.

2.3.3 White-Box-Tests

White-Box-Tests leiten Tests auf Basis der internen Struktur oder der Implementierung eines Systems ab. Die interne Struktur kann Code, Architektur, Workflows und/oder Datenflüsse innerhalb des Systems enthalten (siehe Abschnitt *4.3 White-Box-Testverfahren*).

[3] Anwenderprofil (Scrum)

Die Gründlichkeit von White-Box-Tests kann durch strukturelle Überdeckung gemessen werden. Strukturelle Überdeckung ist der Grad, zu dem einige Arten von strukturellen Elementen durch Tests ausgeführt wurden, und wird ausgedrückt durch den Prozentsatz der Elementart, die übergedeckt wurde.

Auf Stufe der Komponententests basiert die Codeüberdeckung auf dem Prozentsatz des Komponentencodes, der getestet wurde. Die Überdeckung kann in Bezug auf verschiedene Aspekte des Codes (Überdeckungselemente) gemessen werden, wie den Prozentsatz von ausführbaren Anweisungen oder den Prozentsatz von Entscheidungsergebnissen, die in der Komponente getestet wurden. Diese Überdeckungsarten werden zusammengefasst als Codeüberdeckung bezeichnet. Auf der Stufe der Komponentenintegrationstests kann White-Box-Testen auf der Architektur eines Systems basieren, wie den Schnittstellen zwischen den Komponenten. Die strukturelle Überdeckung kann hinsichtlich des Prozentsatzes der Schnittstellen gemessen werden, die durch die Tests ausgeführt wurden.

White-Box-Testentwurf und die Durchführung von White-Box-Tests können spezielle Fähigkeiten und Kenntnisse erfordern, wie Kenntnisse über die Art und Weise, wie der Code aufgebaut ist, wie Daten gespeichert werden (z.B. um mögliche Datenbankanfragen zu bewerten) und wie die Überdeckungswerkzeuge genutzt und ihre Ergebnisse korrekt interpretiert werden.

2.3.4 Änderungsbezogenes Testen

Wenn Änderungen an einem System vorgenommen werden, entweder um einen Fehlerzustand zu korrigieren oder aufgrund einer neuen oder geänderten Funktionalität, sollten Tests durchgeführt werden, um zu bestätigen, dass die Änderungen den Fehlerzustand korrigiert oder die Funktionalität korrekt umgesetzt haben und keine unvorhergesehenen nachteiligen Folgen hervorgerufen haben.

- Fehlernachtests: Nachdem ein Fehlerzustand korrigiert wurde, muss die Software mit allen Testfällen getestet werden, die aufgrund des Fehlerzustands fehlgeschlagen sind. Diese sollten mit der neuen Softwareversion erneut ausgeführt werden. Die Software kann auch mit neuen Tests getestet werden, um Änderungen abzudecken, die nötig waren, um einen Fehlerzustand zu beheben. Zumindest müssen die Schritte, die die Fehlerwirkung, die durch den Fehlerzustand entstanden ist, reproduzieren konnten, in der neuen Softwareversion erneut durchgeführt werden. Der Zweck eines Fehlernachtests ist es, zu bestätigen, dass der ursprüngliche Fehlerzustand erfolgreich behoben wurde.
- Regressionstests: Es ist möglich, dass eine Änderung, die in einem Teil des Codes gemacht wurde (egal, ob aufgrund einer Fehlerbehebung oder aufgrund einer anderen Art von Änderung) versehentlich das Verhalten anderer Teile des Codes beeinflusst. Sei es innerhalb der gleichen Komponente, in anderen Komponenten des gleichen Systems oder sogar in anderen Systemen. Änderungen können Änderungen in der Umgebung sein, wie z.B. eine neue Version eines Betriebssystems oder eines Datenbankmanagementsystems. Solche nicht gewünschten Nebeneffekte nennt man Regressionen. Regressionstests beinhalten die Durchführung von Tests, um solche unbeabsichtigten Nebeneffekte zu finden.

Fehlernachtests und Regressionstests werden in allen Teststufen durchgeführt.

Insbesondere in iterativen und inkrementellen Entwicklungslebenszyklen (z.B. agile) resultieren neue Features, Änderungen zu bestehenden Features und Code-Restrukturierung (Refactoring) in häufigen Änderungen am Code, was in gleicher Weise änderungsbezogenes Testen erfordert. Aufgrund der sich kontinuierlich entwickelnden Beschaffenheit des Systems sind Fehlernachtests und Regressionstests sehr wichtig. Dies gilt insbesondere für Systeme des Internets der Dinge, bei denen individuelle Objekte (z.B. Endgeräte) häufig aktualisiert oder ersetzt werden.

Regressionstestsuiten laufen viele Male und entwickeln sich in der Regel nur langsam. So sind Regressionstests ein starker Kandidat für Testautomatisierung. Die Automatisierung dieser Tests sollte früh im Projekt beginnen (siehe Kapitel 6).

2.3.5 Testarten und Teststufen

Es ist möglich, jede der oben genannten Testarten in jeder Teststufe durchzuführen. Um dies zu veranschaulichen, werden für eine Bankanwendung Beispiele für funktionale, nicht-funktionale, White-Box- und änderungsbezogene Tests über alle Teststufen hinweg gegeben, beginnend mit funktionalen Tests:

- Für Komponententests werden Tests auf der Grundlage dessen entworfen, wie eine Komponente Zinseszinsen berechnet.
- Für Komponentenintegrationstests werden Tests auf der Grundlage dessen entworfen, wie Kontoinformationen, die auf der Benutzeroberfläche erfasst werden, in die fachliche Logik übertragen werden.
- Für Systemtests werden Tests auf Grundlage dessen entworfen, wie Kontoinhaber eine Kreditlinie für ihr Konto beantragen können.
- Für Systemintegrationstests werden Tests auf Grundlage dessen entworfen, wie das System einen externen Microservice dafür nutzt, die Bonität eines Kontoinhabers zu prüfen.
- Für Abnahmetests werden Tests auf Grundlage dessen entworfen, wie der Bankmitarbeiter über die Annahme oder die Ablehnung einer Kreditanfrage entscheidet.

Die folgenden Tests sind Beispiele nicht-funktionaler Tests:

- Für Komponententests werden Tests entworfen, die die Anzahl von CPU-Zyklen bewerten, die erforderlich sind, um eine komplexe Gesamtverzinsung zu berechnen.
- Für Komponentenintegrationstests werden IT-Sicherheitstests für Speicherüberlauf-Schwachstellen aufgrund von Daten, die von der Benutzungsschnittstelle an die fachliche Logik übertragen werden, entworfen.
- Für Systemtests werden Übertragbarkeitstests entworfen, um zu prüfen, ob die Präsentationsschicht auf allen unterstützten Browsern und mobilen Endgeräten funktioniert.
- Für Systemintegrationstests werden Zuverlässigkeitstests entworfen, die die Robustheit des Systems bewerten, falls der Bonitäts-Microservice nicht antwortet.
- Für Abnahmetests werden Gebrauchstauglichkeitstests entworfen, die die Barrierefreiheit der Kreditbearbeitungsoberfläche des Bankmitarbeiters für Menschen mit Behinderung bewerten.

Folgende Tests sind Beispiele für White-Box-Tests:

- Für Komponententests werden Tests entworfen, die vollständige Anweisungs- und Entscheidungsüberdeckung (siehe Abschnitt *4.3 White-Box-Testverfahren*) für alle Komponenten, die Finanzberechnungen durchführen, erzielen sollen.
- Für Komponentenintegrationstests werden Tests entworfen, die testen, wie jeder Bildschirm der Browserbenutzungsoberfläche Daten an den nächsten Bildschirm und an die Fachlogik überträgt.
- Für Systemtests werden Tests entworfen, die Reihenfolgen von Webseiten abdecken, die während einer Kreditanfrage auftauchen können.
- Für Systemintegrationstests werden Tests entworfen, die alle möglichen Anfragearten ausführen, die an den Bonitäts-Microservice gesendet werden können.
- Für Abnahmetests werden Tests entworfen, die alle unterstützten Dateistrukturen der Finanzdaten und Wertebereiche für zwischenbankliche Banküberweisungen abdecken.

Die folgenden Tests sind schließlich Beispiele für änderungsbezogenes Testen:

- Für Komponententests werden automatisierte Regressionstests für jede Komponente entworfen und innerhalb des Continuous-Integration-Frameworks integriert.

- Für Komponentenintegrationstests werden Tests entworfen, die die Beseitigung von schnittstellenbezogenen Fehlerzuständen bestätigen, sobald die Korrekturen im Code-Repository eingecheckt werden.
- Für Systemtests werden alle Tests eines vorgegebenen Workflows wiederholt, falls ein Bildschirm in diesem Workflow sich geändert hat.
- Für Systemintegrationstests werden Tests der Anwendung, die mit dem Bonitäts-Microservice interagiert, täglich als Teil der kontinuierlichen Verteilung (continuous deployment) des Microservice wiederholt.
- Für Abnahmetests werden alle zuvor fehlgeschlagenen Tests wiederholt, nachdem ein Fehlerzustand, der bei Abnahmetests gefunden wurde, behoben worden ist.

Obwohl dieser Abschnitt Beispiele für jede Testart in jeder Teststufe bereitstellt, ist es nicht für jede Software notwendig, jede Testart in jeder Stufe abzubilden. Allerdings ist es wichtig, in jeder Stufe angemessene Tests laufen zu lassen, insbesondere auf der untersten Stufe, auf der die Testart vorkommt.

2.4 Wartungstest

Einmal in Produktionsumgebungen verteilt, benötigen Software und Systeme Wartung. Änderungen verschiedener Art in ausgelieferter Software und Systemen sind fast unvermeidbar, entweder um Fehlerzustände zu beheben, die in der betrieblichen Nutzung aufgetreten sind, um neue Funktionalitäten hinzuzufügen oder um bereits gelieferte Funktionalitäten zu löschen oder zu ändern. Wartung wird auch benötigt, um nicht-funktionale Qualitätsmerkmale der Komponente oder des Systems über seine Lebenszeit hinweg zu erhalten oder zu verbessern, insbesondere Performanz, Kompatibilität, Zuverlässigkeit, IT-Sicherheit und Übertragbarkeit.

Wenn Änderungen als Teil der Wartung vorgenommen werden, sollten Wartungstests durchgeführt werden, sowohl um den Erfolg zu bewerten, mit dem die Änderungen vorgenommen wurden, als auch um mögliche Seiteneffekte (z.B. Regressionen) in Teilen des Systems zu prüfen, die unverändert bleiben (was üblicherweise der Großteil des Systems ist). Wartung kann geplante Releases und ungeplante Releases (sog. Hot Fixes) beinhalten.

Ein Wartungsrelease kann, abhängig vom Umfang, Wartungstests in mehreren Teststufen erfordern und verschiedene Testarten nutzen. Der Umfang von Wartungstests hängt ab von:

- Der Risikohöhe der Änderung (z.B. der Grad, zu dem der geänderte Bereich der Software mit anderen Komponenten oder Systemen kommuniziert)
- Der Größe des bestehenden Systems
- Der Größe der Änderung

2.4.1 Auslöser für Wartung

Es gibt verschiedene Gründe, warum Softwarewartung und somit auch Wartungstests stattfinden, sowohl für geplante als auch für ungeplante Änderungen.

Wir können die Auslöser für Wartung wie folgt klassifizieren:

- Modifikation, wie geplante Verbesserungen (z.B. releasegesteuert), korrigierende Änderungen und Notfalländerungen, Änderungen der betrieblichen Umgebung (wie geplante Betriebssystem- oder Datenbankaktualisierungen), Aktualisierungen von kommerzieller Standardsoftware und Patches für Fehlerzustände und Schwachstellen

- Migration, wie von einer Plattform zu einer anderen, was betriebliche Tests der neuen Umgebung und auch der veränderten Software erfordert oder Tests der Datenkonvertierung, wenn Daten von einer anderen Anwendung in das System migriert werden, das gewartet wird
- Außerbetriebnahme, z.B. wenn eine Anwendung das Ende ihres Lebenszyklus erreicht. Wenn eine Anwendung oder ein System außer Betrieb genommen wird, kann dies Tests der Datenmigration oder, wenn lange Datenspeicherungsfristen erforderlich sind Tests der Datenarchivierung erfordern
- Das Testen von Wiederherstellungsverfahren nach der Archivierung bei langen Aufbewahrungsfristen kann ebenso notwendig sein
- Regressionstests können erforderlich sein, um sicherzustellen, dass jede Funktionalität, die in Betrieb bleibt, weiterhin funktioniert.

Für Systeme des Internets der Dinge können Wartungstests durch die Einführung von vollständig neuen oder modifizierten Dingen in das System ausgelöst werden, wie Hardwaregeräte und Softwaredienste. Die Wartungstests für derartige Systeme konzentrieren sich insbesondere auf Integrationstests auf verschiedenen Ebenen (z.B. Netzwerkebene, Anwendungsebene) und auf IT-Sicherheitsaspekte, insbesondere solche, die sich auf persönliche Daten beziehen.

2.4.2 Auswirkungsanalyse für Wartung

Die Auswirkungsanalyse bewertet die Veränderungen, die für ein Wartungsrelease gemacht wurden, um die gewünschten Folgen sowie die erwarteten als auch möglichen Nebeneffekte einer Veränderung zu identifizieren und um die Bereiche des Systems zu identifizieren, die von der Veränderung betroffen sind. Die Auswirkungsanalyse kann auch dabei helfen, die Wirkung einer Veränderung auf bestehende Tests zu identifizieren. Die Nebeneffekte und betroffenen Bereiche im System müssen für Regressionen getestet werden, möglicherweise nach der Aktualisierung aller bestehenden Tests, die von der Veränderung betroffen sind.

Die Auswirkungsanalyse kann vor der Veränderung durchgeführt werden, um bei der Entscheidung zu helfen, ob die Veränderung vorgenommen werden sollte. Dies erfolgt auf der Grundlage der potenziellen Folgen für andere Bereiche des Systems.

Die Auswirkungsanalyse kann schwierig sein, wenn:

- Spezifikationen (z.B. Fachanforderungen, User-Stories, Architektur) veraltet sind oder fehlen
- Testfälle nicht dokumentiert oder veraltet sind
- Bidirektionale Verfolgbarkeit zwischen Tests und der Testbasis nicht aufrechterhalten wurde
- Werkzeugunterstützung schwach ist oder nicht existiert
- Die beteiligten Personen keine Fach- oder Systemkenntnis haben
- Während der Entwicklung nur ungenügende Aufmerksamkeit auf die Wartbarkeit der Software gelegt wurde

3. Statischer Test – 135 Minuten

Schlüsselbegriffe

Ad-hoc-Review, checklistenbasiertes Review, dynamischer Test, formales Review, informelles Review, Inspektion, perspektivisches Lesen, Review, rollenbasiertes Review, statische Analyse, statischer Test, szenariobasiertes Review, technisches Review, Walkthrough

Lernziele für den statischen Test

3.1 Grundlagen des statischen Tests

FL-3.1.1 (K1) Arten von Softwarearbeitsergebnissen erkennen können, die durch die verschiedenen statischen Testverfahren geprüft werden können

FL-3.1.2 (K2) Beispiele nennen können, um den Wert des statischen Tests zu beschreiben

FL-3.1.3 (K2) Den Unterschied zwischen statischen und dynamischen Verfahren unter Berücksichtigung der Ziele, der zu identifizierenden Fehlerzustände und der Rolle dieser Verfahren innerhalb des Softwarelebenszyklus erklären können

3.2 Reviewprozess

FL-3.2.1 (K2) Die Aktivitäten des Reviewprozesses für Arbeitsergebnisse zusammenfassen können

FL-3.2.2 (K1) Die unterschiedlichen Rollen und Verantwortlichkeiten in einem formalen Review erkennen können

FL-3.2.3 (K2) Die Unterschiede zwischen den unterschiedlichen Reviewarten erklären können: informelles Review, Walkthrough, technisches Review und Inspektion

FL-3.2.4 (K3) Ein Reviewverfahren auf ein Arbeitsergebnis anwenden können, um Fehlerzustände zu finden

FL-3.2.5 (K2) Die Faktoren erklären können, die zu einem erfolgreichen Review beitragen

3.1 Grundlagen des statischen Tests

Im Gegensatz zu dynamischen Tests, die die Ausführung der Software, die getestet wird, erfordern, verlässt sich statischer Test auf die manuelle Prüfung der Arbeitsergebnisse (d.h. Reviews) oder auf die werkzeuggestützte Bewertung des Codes oder anderer Arbeitsergebnisse (d.h. statische Analyse). Beide Arten des statischen Tests beurteilen den Code oder andere Arbeitsergebnisse, die getestet werden, ohne den Code oder das Arbeitsergebnis, das getestet wird, tatsächlich auszuführen.

Die statische Analyse ist wichtig für sicherheitskritische Computersysteme (z.B. Luftfahrt, medizinische oder Kernkraftsoftware), aber statische Analysen sind auch in anderen Bereichen wichtig und üblich geworden. Zum Beispiel ist die statische Analyse ein wichtiger Teil der IT-Sicherheitstests. Die statische Analyse ist auch häufig eingebettet in Werkzeuge für automatisierte Software Build & Verteilung, z.B. in der agilen Entwicklung, in der kontinuierlichen Auslieferung (continuous delivery) und in der kontinuierlichen Bereitstellung in die Zielumgebung (continuous deployment).

3.1.1 Arbeitsergebnisse, die durch statische Tests geprüft werden können

Fast jedes Arbeitsergebnis kann durch statische Tests geprüft werden (Reviews und/oder statische Analysen), z.B.:

- Spezifikationen, u.a. Fachanforderungen, funktionale Anforderungen, und IT-Sicherheitsanforderungen
- Epics, User-Stories und Abnahmekriterien
- Architektur und Entwurfsspezifikationen
- Code
- Testmittel einschließlich Testkonzepten, Testfällen, Testablauf und automatisierten Testskripten
- Benutzeranleitungen
- Webseiten
- Verträge, Projektpläne, Zeitpläne und Budgetplanung
- Aufsetzen der Konfiguration u. Infrastruktur
- Modelle wie Aktivitätsdiagramme, die für modellbasiertes Testen genutzt werden können (siehe ISTQB-MBT und Kramer 2016)

Reviews können auf jedes Arbeitsergebnis angewendet werden, von dem die Teilnehmer wissen, wie es zu lesen und zu verstehen ist. Statische Analysen können effizient auf jedes Arbeitsergebnis mit einer formalen Struktur angewendet werden (üblicherweise Code oder Modelle), für die es ein geeignetes statisches Analysewerkzeug gibt. Statische Analysen können sogar mit Werkzeugen durchgeführt werden, die Arbeitsergebnisse bewerten, die wie Anforderungen in natürlicher Sprache geschrieben sind (z.B. Prüfung auf Rechtschreibung, Grammatik und Lesbarkeit).

3.1.2 Vorteile des statischen Tests

Statische Testverfahren haben eine Reihe von Vorteilen. Wenn sie früh im Softwareentwicklungslebenszyklus eingesetzt werden, können statische Tests das frühe Erkennen von Fehlerzuständen ermöglichen, noch bevor dynamische Tests durchgeführt werden (z.B. in Anforderungs- oder Entwurfsspezifikationsreviews, in Backlog-Refinements usw.). Fehlerzustände, die früh gefunden werden, lassen sich häufig kostengünstiger entfernen als Fehlerzustände, die später im Lebenszyklus gefunden werden, insbesondere im Vergleich zu Fehlerzuständen, die erst gefunden werden, nachdem die Software auf die Zielumgebung gebracht wurde und aktiv in Nutzung ist. Die Nutzung statischer Testverfahren zum Auffinden von Fehlerzuständen und die folgende prompte Behebung dieser Fehlerzustände ist für das Unternehmen fast immer

kostengünstiger als der Einsatz dynamischer Tests zum Auffinden von Fehlerzuständen im Testobjekt und deren Behebung. Dies gilt insbesondere unter Berücksichtigung der zusätzlichen Kosten, die mit der Aktualisierung anderer Arbeitsergebnisse sowie der Durchführung von Fehlernachtests und Regressionstests einhergehen.

Zusätzliche Vorteile statischer Tests können u.a. sein:

- Effizienteres Erkennen und Korrigieren von Fehlerzuständen schon vor der Durchführung dynamischer Tests
- Identifizieren von Fehlerzuständen, die in dynamischen Tests nicht leicht zu finden sind
- Verhindern von Fehlerzuständen im Entwurf oder der Kodierung durch das Aufdecken von Inkonsistenzen, Mehrdeutigkeiten, Widersprüchen, Auslassungen, Ungenauigkeiten und Redundanzen in Anforderungen
- Erhöhen der Entwicklungsproduktivität (z.B. durch verbesserte Entwürfe, mehr wartungsfähigen Code)
- Reduzieren von Entwicklungskosten und -zeit
- Reduzieren von Testkosten und -zeit
- Reduzieren der Gesamtkosten der Qualität über die Lebenszeit der Software hinweg, aufgrund von weniger Fehlerwirkungen zu einem späteren Zeitpunkt im Lebenszyklus oder nach Auslieferung in die Produktion
- Verbesserte Kommunikation zwischen Teammitgliedern durch die Teilnahme an Reviews

3.1.3 Unterschiede zwischen statischen und dynamischen Tests

Statischer und dynamischer Test können die gleichen Ziele verfolgen (siehe Abschnitt *1.1.1 Typische Ziele des Testens*), wie die Beurteilung der Qualität von Arbeitsergebnissen und das Identifizieren von Fehlerzuständen so früh wie möglich. Statischer und dynamischer Test ergänzen sich gegenseitig, da sie unterschiedliche Fehlerarten finden.

Ein Hauptunterschied ist, dass statische Tests Fehlerzustände in Arbeitsergebnissen direkt finden, anstatt erst die Fehlerwirkungen zu identifizieren, die durch die Fehlerzustände entstehen, wenn die Software läuft. Ein Fehlerzustand kann in einem Arbeitsergebnis über einen langen Zeitraum hinweg bestehen, ohne eine Fehlerwirkung zu verursachen. Der Pfad, auf dem der Fehlerzustand liegt, wird vielleicht selten genutzt oder ist schwer zu erreichen. Es ist dann nicht einfach, einen dynamischen Test zu konstruieren und durchzuführen, der den Fehlerzustand aufdeckt. Statische Tests können einen solchen Fehlerzustand unter Umständen mit wesentlich geringerem Aufwand finden.

Ein weiterer Unterschied ist, dass statische Tests genutzt werden können, um die Konsistenz und interne Qualität der Arbeitsergebnisse zu verbessern, während dynamische Tests in der Regel auf extern sichtbares Verhalten konzentriert sind.

Im Vergleich zu dynamischen Tests sind typische Fehlerzustände, die einfacher und kostengünstiger durch statische Tests zu finden und zu beheben sind, u.a. folgende:

- Anforderungsfehler (z.B. Inkonsistenzen, Mehrdeutigkeiten, Widersprüche, Auslassungen, Ungenauigkeiten und Redundanzen)
- Entwurfsfehler (z.B. ineffiziente Algorithmen oder Datenbankstrukturen, hohe Koppelung, geringe Kohäsion)
- Programmierfehler (z.B. Variablen mit nicht definierten Werten, Variablen, die deklariert, aber nie genutzt werden, unerreichbarer Code, doppelter Code)

- Abweichungen von Standards (z.B. fehlende Einhaltung von Programmierrichtlinien)
- Falsche Schnittstellenspezifikationen (z.B. unterschiedliche Maßeinheiten im aufrufenden und im aufgerufenen System)
- Schwachstellen in der IT-Sicherheit (z.B. Neigung zu Pufferüberlauf)
- Lücken oder Ungenauigkeiten in der Verfolgbarkeit oder die Überdeckung der Testbasis (z.B. fehlende Tests für ein Abnahmekriterium)

Darüber hinaus können die meisten Wartbarkeitsfehler nur durch statische Tests gefunden werden (z.B. ungeeignete Modularisierung, schlechte Wiederverwendbarkeit von Komponenten, schwer analysierbarer Code, der ohne das Einfügen neuer Fehlerzustände schwer zu modifizieren ist).

3.2 Reviewprozess

Reviews variieren von informellen bis zu formalen Reviews. Informelle Reviews sind dadurch gekennzeichnet, dass sie keinem definierten Prozess folgen und kein formal dokumentiertes Ergebnis haben. Formale Reviews sind durch die Teilnahme von Teams, dokumentierten Ergebnissen des Reviews und dokumentierten Vorgehensweisen für die Durchführung des Reviews gekennzeichnet. Die Formalität eines Reviewprozesses hängt von Faktoren wie dem Softwareentwicklungslebenszyklus-Modell, der Reife des Entwicklungsprozesses, der Komplexität der zu prüfenden Arbeitsergebnisse, rechtlichen oder regulatorischen Anforderungen und/oder der Notwendigkeit für einen Prüfnachweis ab.

Der Fokus eines Reviews hängt von den vereinbarten Zielen des Reviews ab (z.B. Finden von Fehlerzuständen, Verständnisgewinnung, Training von Teilnehmern wie Testern und neuen Teammitgliedern oder Diskussion und Entscheidung durch Konsens).

Die ISO-Norm (ISO/IEC 20246) enthält tiefergehende Beschreibungen des Reviewprozesses für Arbeitsergebnisse sowie Rollen und Reviewverfahren.

3.2.1 Reviewprozess für Arbeitsergebnisse

Der Reviewprozess umfasst die folgenden Hauptaktivitäten:

Planung

- Definition des Umfangs inkl. des Zwecks des Reviews, welche Dokumente oder Teile von Dokumenten im Review einbezogen werden sollen und der Qualitätsmerkmale, die bewertet werden sollen
- Schätzung von Aufwand und Zeitbedarf
- Identifizieren von Revieweigenschaften wie der Reviewart mit Rollen, Aktivitäten und Checklisten
- Auswahl der Personen, die am Review teilnehmen sollen, und Zuordnung der Rollen
- Definition der Eingangs- und Endekriterien für formalere Reviewarten (z.B. Inspektionen)
- Prüfung, ob Eingangskriterien erfüllt sind (für formalere Reviewarten)

Reviewbeginn

- Verteilen des Arbeitsergebnisses (physisch oder elektronisch) und anderer Materialien, wie Befund-Template, Checklisten und zugehörige Arbeitsergebnisse
- Erläutern des Umfangs, der Ziele, des Prozesses, der Rollen und der Arbeitsergebnisse gegenüber den Teilnehmern
- Beantwortung von Fragen, die die Teilnehmer zum Review haben könnten

Individuelles Review (d.h. individuelle Vorbereitung)

- Review des gesamten oder von Teilen des Arbeitsergebnisses
- Aufzeichnung potenzieller Fehlerzustände, Empfehlungen und Fragen

Befundkommunikation und -analyse

- Kommunikation identifizierter potenzieller Fehlerzustände (z.B. in einer Reviewsitzung)
- Analyse potenzieller Fehlerzustände, Zuweisung von Zuständigkeit und Status
- Bewertung und Dokumentation von Qualitätsmerkmalen
- Bewertung der Reviewbefunde gegenüber den Endekriterien, um eine Reviewentscheidung zu treffen (ablehnen, umfangreiche Änderungen notwendig, annehmen, vielleicht mit geringfügigen Änderungen)

Fehlerbehebung und Bericht

- Für die Befunde, die Änderungen an einem Arbeitsergebnis erfordern, Fehlerberichte erstellen
- Beheben von im geprüften Arbeitsergebnis gefundenen Fehlerzuständen (üblicherweise durch den Autor)
- Kommunikation von Fehlerzuständen an die zuständige Person oder das zuständige Team (wenn sie in einem Arbeitsergebnis gefunden wurden, das zu dem Arbeitsergebnis, welches geprüft wurde, in Beziehung steht)
- Aufzeichnung des aktualisierten Status der Fehlerzustände (in formalen Reviews), potenziell auch mit der Zustimmung der Person, die den Befund erstellte
- Sammeln von Metriken (für formalere Reviewarten)
- Prüfen, dass Endekriterien erfüllt sind (für formalere Reviewarten)
- Abnahme des Arbeitsergebnisses, wenn die Endekriterien erreicht sind

Die Reviewergebnisse eines Arbeitsergebnisses können in Abhängigkeit von der Reviewart und der Formalität variieren, wie in Abschnitt *3.2.3 Reviewarten* beschrieben.

3.2.2 Rollen und Verantwortlichkeiten in einem formalen Review

Ein typisches formales Review umfasst die folgenden Rollen:

Autor

- Erstellt das Arbeitsergebnis, das einem Review unterzogen wird
- Behebt (falls notwendig) Fehlerzustände im Arbeitsergebnis, das einem Review unterzogen wurde

Management

- Ist verantwortlich für die Reviewplanung
- Entscheidet über die Durchführung von Reviews
- Legt Mitarbeiter, Budget und Fristen fest
- Überwacht die stetige Kosteneffizienz
- Trifft Steuerungsentscheidungen im Fall von unangemessenen Ergebnissen

Reviewmoderator[4] (oft auch Moderator genannt)

- Stellt den erfolgreichen Ablauf von Reviewsitzungen sicher (falls solche stattfinden)
- Vermittelt, falls nötig, zwischen verschiedenen Standpunkten
- Ist oft die Person, von der der Erfolg des Reviews abhängt

Reviewleiter

- Übernimmt die Verantwortung für das Review
- Entscheidet, wer einbezogen wird, und bestimmt, wann und wo es stattfindet

Gutachter

- Können Fachexperten sein, Personen, die in dem Projekt arbeiten, Stakeholder mit einem Interesse an dem Arbeitsergebnis und/oder Einzelpersonen mit spezifischen technischen oder fachlichen Hintergründen
- Identifizieren potenzielle Fehlerzustände des im Review befindlichen Arbeitsergebnisses
- Können verschiedene Perspektiven vertreten (z.B. Tester, Entwickler, Benutzer, Betreiber, Businessanalysten, Experten für Gebrauchstauglichkeit)

Protokollant

- Erhebt potenzielle Fehlerzustände, die während der individuellen Reviewaktivitäten gefunden werden
- Erfasst neue potenzielle Fehlerzustände, offene Punkte und Entscheidungen aus der Reviewsitzung (falls eine stattfindet)

In einigen Reviewarten kann eine Person mehr als eine Rolle einnehmen und die Aktionen, die mit jeder Rolle in Verbindung stehen, können ebenso je nach Reviewart variieren. Darüber hinaus gibt es nach Einführung von Werkzeugen, die den Reviewprozess insbesondere bei der Aufzeichnung von Fehlerzuständen, offenen Punkten und Entscheidungen unterstützen, häufig keine Notwendigkeit mehr für einen Protokollanten.

Außerdem sind detailliertere Rollen möglich, wie in der ISO-Norm (ISO/IEC 20246) beschrieben.

3.2.3 Reviewarten

Obwohl Reviews für unterschiedliche Zwecke verwendet werden können, ist eines der Hauptziele, Fehlerzustände aufzudecken. Alle Reviewarten können beim Finden von Fehlerzuständen helfen. Die Reviewart sollte neben anderen Auswahlkriterien auf den Projektbedürfnissen, den verfügbaren Ressourcen, der Produktart und ihrer Risiken, dem Geschäftszweig und der Unternehmenskultur basieren.

Ein einziges Arbeitsergebnis kann Gegenstand mehrerer Arten von Reviews sein. Wenn mehr als eine Reviewart genutzt wird, kann die Reihenfolge variieren. Zum Beispiel kann ein informelles Review vor einem technischen Review durchgeführt werden, um sicherzustellen, dass das Arbeitsergebnis für ein technisches Review bereit ist.

Die Reviewarten, die oben beschrieben wurden, können als sogenannte Peer-Reviews durchgeführt werden, d.h. von einem Kollegen auf einer ähnlichen Ebene der Unternehmenshierarchie.

Die Fehlerarten, die in einem Review gefunden werden, variieren und hängen insbesondere vom Arbeitsergebnis ab, das Gegenstand des Reviews ist. In Abschnitt *3.1.3 Unterschiede zwischen statischen und dynamischen Tests* werden Beispiele für Fehlerzustände gegeben, die in Reviews von verschiedenen

[4] Oder Facilitator.

Arbeitsergebnissen gefunden werden können, und Gilb gibt weitere Informationen über formale Inspektionen (siehe Gilb 1993). Reviews können nach verschiedenen Eigenschaften klassifiziert werden. Die folgende Liste beinhaltet die vier gängigsten Arten von Reviews und ihre zugehörigen Eigenschaften.

Informelles Review (z.B. Buddy-Check, Pairing (paarweises Zusammenarbeiten), paarweises Review)

- Hauptzweck: Erkennen von potenziellen Fehlerzuständen
- Mögliche zusätzliche Zwecke: Generieren von neuen Ideen oder Lösungen, schnelle Lösung kleinerer Probleme
- Basiert nicht auf einem formalen (dokumentierten) Prozess
- Beinhaltet möglicherweise keine Reviewsitzung
- Kann von einem Kollegen des Autors durchgeführt werden (Buddy-Check) oder von mehreren Personen
- Ergebnisse können dokumentiert werden
- Der Nutzen variiert abhängig von den Gutachtern
- Die Nutzung von Checklisten ist optional
- Sehr verbreitet in der agilen Entwicklung

Walkthrough

- Hauptzwecke: Fehlerzustände finden, das Softwareprodukt verbessern, alternative Umsetzungen erwägen, Konformität mit Standards und Spezifikationen bewerten
- Mögliche zusätzliche Zwecke: Ideenaustausch über Verfahren oder Stilvariationen, Ausbildung der Teilnehmer, Konsenserzielung
- Individuelle Vorbereitung vor der Reviewsitzung ist optional
- Reviewsitzungen werden üblicherweise vom Autor des Arbeitsergebnisses geleitet
- Protokollant ist obligatorisch
- Die Nutzung von Checklisten ist optional
- Kann in Form von Szenarios, Dry Run (Probelauf) oder Simulationen durchgeführt werden
- Protokolle potenzieller Fehlerzustände und Reviewberichte werden erstellt
- Kann in der Praxis von recht informell bis hin zu sehr formal variieren

Technisches Review

- Hauptzwecke: Gewinnen von Konsens, finden von potenziellen Fehlerzuständen
- Mögliche weitere Zwecke: Qualität bewerten und Vertrauen in das Arbeitsergebnis schaffen, neue Ideen generieren, den Autor motivieren und befähigen, zukünftige Arbeitsergebnisse zu verbessern, alternative Umsetzungen bedenken
- Gutachter sollten fachspezifische Kollegen des Autors und fachspezifische Experten in der gleichen oder in anderen Disziplinen sein
- Individuelle Vorbereitung vor der Reviewsitzung ist erforderlich
- Eine Reviewsitzung ist optional, idealerweise wird sie von einem geschulten Reviewmoderator (üblicherweise nicht dem Autor) geleitet
- Protokollführung ist obligatorisch, idealerweise nicht vom Autor

- Die Nutzung von Checklisten ist optional
- Protokolle potenzieller Fehlerzustände und Reviewberichte werden erstellt

Inspektion

- Hauptzwecke: Erkennen potenzieller Fehlerzustände, bewerten der Qualität und Vertrauen in das Arbeitsergebnis schaffen, durch das Lernen des Autors und Grundursachenanalyse zukünftige ähnliche Fehlerzustände verhindern
- Mögliche weitere Zwecke: den Autor motivieren und befähigen, zukünftige Arbeitsergebnisse und den Softwareentwicklungsprozess zu verbessern, erzielen von Konsens
- Folgt einem definierten Prozess mit formalen dokumentierten Ergebnissen, basierend auf Regeln und Checklisten
- Nutzt klar definierte Rollen, wie die in Abschnitt 3.2.2 Rollen und Verantwortlichkeiten in einem formalen Review beschriebenen, die verpflichtend sind und es kann ein zur Verfügung gestellter Vorleser einbezogen werden, der das Arbeitsergebnis während der Reviewsitzung laut vorliest, häufig paraphrasiert (d.h. sinngemäß, in eigenen Worten) wiedergibt.
- Individuelle Vorbereitung vor der Reviewsitzung ist erforderlich
- Gutachter sind entweder gleichrangig mit dem Autor oder Experten in anderen Fachrichtungen, die für das Arbeitsergebnis relevant sind
- Festgelegte Eingangs- und Endekriterien werden genutzt
- Protokollant ist obligatorisch
- Die Reviewsitzung wird von einem geschulten Moderator geleitet (nicht vom Autor)
- Der Autor kann nicht als Reviewleiter, Vorleser oder Protokollant agieren
- Protokolle potenzieller Fehlerzustände und Reviewberichte werden erstellt
- Metriken werden gesammelt und genutzt, um den gesamten Softwareentwicklungsprozess zu verbessern, einschließlich des Inspektionsprozesses

3.2.4 Die Anwendung von Reviewverfahren

Es gibt eine Reihe von Reviewverfahren, die während des individuellen Reviews (d.h. der individuellen Vorbereitung) angewendet werden können, um Fehlerzustände zu erkennen. Diese Verfahren können für alle oben beschriebenen Reviewarten verwendet werden. Die Effektivität der Verfahren kann in Abhängigkeit von der genutzten Reviewart variieren. Beispiele für unterschiedliche individuelle Reviewverfahren für verschiedene Reviewarten werden unten aufgelistet.

Ad hoc

In einem Ad-hoc-Review wird Gutachtern nur wenig oder gar keine Anleitung an die Hand gegeben, wie die Aufgabe ausgeführt werden soll. Gutachter lesen häufig das Arbeitsergebnis sequenziell und identifizieren und dokumentieren Befunde, sobald sie auf sie stoßen. Ad-hoc-Reviews sind ein verbreitet genutztes Verfahren, das wenig Vorbereitung erfordert. Dieses Verfahren hängt stark von den Fähigkeiten der Gutachter ab und kann zu vielen doppelt berichteten Befunden führen.

Checklistenbasiert

Ein checklistenbasiertes Review ist ein systematisches Verfahren, bei dem die Gutachter Befunde auf Basis von Checklisten erkennen, die bei Reviewbeginn verteilt werden (z.B. durch den Reviewmoderator). Eine Reviewcheckliste besteht aus einem Set an Fragen, die auf potenziellen Fehlerzuständen basieren

und sich aus Erfahrungen ableiten lassen. Checklisten sollten spezifisch auf die Art des Arbeitsergebnisses, das Gegenstand des Reviews ist, zugeschnitten sein und regelmäßig aktualisiert werden, um Befundarten, die in früheren Reviews übersehen wurden, abzudecken. Der Hauptvorteil des checklistenbasierten Verfahrens ist die systematische Überdeckung typischer Fehlerarten. Es sollte darauf geachtet werden, nicht nur einfach der Checkliste im individuellen Review zu folgen, sondern auch ein Auge auf Fehlerzustände außerhalb der Checkliste zu haben.

Szenarien und Dry Runs (Probeläufe)

In einem szenariobasierten Review erhalten Gutachter strukturierte Richtlinien, wie sie ein Arbeitsergebnis durchlesen sollen. Ein szenariobasiertes Review unterstützt Gutachter dabei, Probeläufe mit dem Arbeitsergebnis auf Basis der erwarteten Nutzung des Arbeitsergebnisses durchzuführen (falls das Arbeitsergebnis in einem angemessenen Format, wie in Form von Anwendungsfällen, dokumentiert ist). Diese Szenarien geben Gutachtern bessere Richtlinien darüber an die Hand, wie spezifische Fehlerarten identifiziert werden können, als einfache Checklisteneinträge. Wie bei checklistenbasierten Reviews sollten Gutachter sich nicht ausschließlich auf die dokumentierten Szenarien beschränken, um andere Fehlerarten (z.B. fehlende Leistungsmerkmale) nicht zu übersehen.

Perspektivisch

Ähnlich wie im rollenbasierten Review (s.u.) nehmen beim perspektivischen Lesen Gutachter im individuellen Review unterschiedliche Standpunkte von unterschiedlichen Stakeholdern ein. Typische Standpunkte von Stakeholdern sind u.a. Endanwender, Marketing, Designer, Tester oder Betrieb. Die Nutzung unterschiedlicher Stakeholder-Standpunkte führt zu mehr Tiefe im individuellen Review mit weniger Doppelungen von Befunden unter den Gutachtern.

Darüber hinaus verlangt perspektivisches Lesen von den Gutachtern auch, dass sie versuchen, das Arbeitsergebnis, das Gegenstand des Reviews ist, zu nutzen, um das Produkt zu generieren, das sie daraus ableiten wollen. Zum Beispiel würde ein Tester versuchen, einen Entwurf für Abnahmetests zu generieren, wenn er perspektivisches Lesen bei einer Anforderungsspezifikation durchführt, um zu sehen, ob alle notwendigen Informationen enthalten sind. Außerdem wird erwartet, dass im perspektivischen Lesen Checklisten genutzt werden.

Empirische Studien haben gezeigt, dass perspektivisches Lesen das effektivste allgemeine Verfahren für das Review von Anforderungen und technischen Arbeitsergebnissen ist. Ein Schlüsselfaktor für den Erfolg ist der korrekte Einbezug und die angemessene Gewichtung der Sichtweisen der unterschiedlichen Stakeholder basierend auf den Risiken. Siehe Shull 2000 für Details über perspektivisches Lesen und Sauer 2000 für die Effektivität unterschiedlicher Reviewverfahren.

Rollenbasiert

Ein rollenbasiertes Review ist ein Verfahren, in dem die Gutachter das Arbeitsergebnis aus der Perspektive von individuellen Stakeholderrollen bewerten. Typische Rollen beinhalten spezifische Arten von Endanwendern (erfahren, unerfahren, Senioren, Kinder usw.) und spezifische Rollen innerhalb einer Organisation (Benutzeradministrator, Systemadministrator, Performanztester usw.). Es werden die gleichen Prinzipien wie beim perspektivischen Lesen angewendet, da die Rollen ähnlich sind.

3.2.5 Erfolgsfaktoren für Reviews

Für ein erfolgreiches Review müssen die geeignete Reviewart und die genutzten Verfahren in Betracht gezogen werden. Darüber hinaus gibt es eine Reihe von Faktoren, die das Ergebnis des Reviews beeinflussen.

Organisatorische Erfolgsfaktoren für Reviews sind u.a.:

- Jedes Review hat klare Ziele, die während der Reviewplanung definiert und als messbare Endekriterien genutzt werden.

- Es werden die Reviewarten genutzt, die passend sind, um die Ziele zu erreichen, und geeignet sind für Art und Stufe des Softwarearbeitsergebnisses sowie der Teilnehmer.
- Alle genutzten Reviewverfahren, wie checklistenbasiertes oder rollenbasiertes Review, sind geeignet für die effektive Identifizierung von Fehlerzuständen im Arbeitsergebnis, das Gegenstand des Reviews ist.
- Alle Checklisten, die genutzt werden, gehen auf Hauptrisiken ein und sind aktuell.
- Große Dokumente werden in kleinen Abschnitten geschrieben und in Reviews geprüft, damit die Qualitätssteuerung so erfolgt, dass Autoren frühe und häufige Rückmeldungen zu Fehlerzuständen erhalten.
- Teilnehmer haben ausreichend Zeit für die Vorbereitung.
- Reviews werden mit angemessener Vorankündigung geplant.
- Das Management unterstützt den Reviewprozess (z.B. durch Bereitstellen angemessener Zeiträume für Reviewaktivitäten innerhalb der Projektpläne).
- Reviews werden in die Qualitäts- und Testrichtlinien des Unternehmens integriert

Personenbezogene Erfolgsfaktoren für Reviews sind u.a.:

- Die richtigen Personen sind involviert, damit die Reviewziele erreicht werden, z.B. Personen mit unterschiedlichen Fähigkeitsprofilen oder Sichtweisen, die das Dokument als Basis für ihre Arbeit nutzen könnten.
- Tester werden als wertgeschätzte Gutachter gesehen, die zum Review beitragen und etwas über das Arbeitsergebnis lernen, was ihnen dabei hilft, effektivere Tests vorzubereiten und dies auch noch früher.
- Teilnehmer widmen den Details angemessene Zeit und Aufmerksamkeit.
- Reviews werden in kleinen Schritten vorgenommen, so dass Gutachter nicht die Konzentration während des individuellen Reviews und/oder der Reviewsitzung (falls eine stattfindet) verlieren.
- Gefundene Fehlerzustände werden objektiv anerkannt, bewertet und behandelt.
- Das Meeting ist gut geleitet, so dass die Teilnehmer es als wertvolle Nutzung ihrer Zeit ansehen.
- Das Review wird in einer vertrauensvollen Atmosphäre abgehalten, das Ergebnis wird nicht für die Bewertung der Teilnehmer herangezogen.
- Die Teilnehmer vermeiden Körpersprache und Verhaltensweisen, die Langeweile, Frust oder Feindseligkeit gegenüber anderen Teilnehmern ausdrücken.
- Angemessene Schulungen, insbesondere für formalere Reviewarten wie Inspektionen, werden bereitgestellt.
- Eine Kultur des Lernens und der Prozessverbesserungen wird gefördert.

(Siehe Gilb 1993, Wiegers 2002 und van Veenendaal 2004 für mehr Informationen über erfolgreiche Reviews.)

4. Testverfahren – 330 Minuten

Schlüsselbegriffe

Anweisungsüberdeckung, Anwendungsfallbasierter Test, Äquivalenzklassenbildung, Black-Box-Testverfahren, checklistenbasiertes Testen, Entscheidungstabellentest, Entscheidungsüberdeckung, erfahrungsbasierte Testverfahren, exploratives Testen, Grenzwertanalyse, intuitive Testfallermittlung, Testverfahren, Überdeckung, White-Box-Testverfahren, Zustandsübergangstest

Lernziele für Testverfahren

4.1 Kategorien von Testverfahren

FL-4.1.1 (K2) Die Eigenschaften, Gemeinsamkeiten und Unterschiede zwischen Black-Box-Testverfahren, White-Box-Testverfahren und erfahrungsbasierten Testverfahren erklären können

4.2 Black-Box-Testverfahren

FL-4.2.1 (K3) Die Äquivalenzklassenbildung anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.2 (K3) Die Grenzwertanalyse anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.3 (K3) Entscheidungstabellentests anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.4 (K3) Zustandsübergangstests anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.5 (K2) Erklären können, wie man Testfälle aus einem Anwendungsfall ableitet

4.3 White-Box-Testverfahren

FL-4.3.1 (K2) Anweisungsüberdeckung erklären können

FL-4.3.2 (K2) Entscheidungsüberdeckung erklären können

FL-4.3.3 (K2) Die Bedeutung von Anweisungs- und Entscheidungsüberdeckung erklären können

4.4 Erfahrungsbasierte Testverfahren

FL-4.4.1 (K2) Die intuitive Testfallermittlung erklären können

FL-4.4.2 (K2) Exploratives Testen erklären können

FL-4.4.3 (K2) Checklistenbasiertes Testen erklären können

4.1 Kategorien von Testverfahren

Zweck von Testverfahren, einschließlich der hier beschriebenen, ist es, eine Hilfe beim Bestimmen von Testbedingungen, Testfällen und Testdaten zu sein.

Die Wahl, welches Testverfahren zu nutzen ist, hängt von einer Reihe von Faktoren ab, darunter u.a.:

- Die Komplexität der Komponente oder des Systems
- Regulatorische Standards
- Kundenanforderungen oder vertragliche Anforderungen
- Risikostufe und Risikoarten
- Verfügbare Dokumentation
- Kenntnisse und Fähigkeiten des Testers
- Verfügbare Werkzeuge
- Zeit und Budget
- Softwareentwicklungslebenszyklus-Modell
- Die Arten von Fehlerzuständen, die in der Komponente oder dem System erwartet werden

Einige Verfahren sind in bestimmten Situationen und auf bestimmten Teststufen geeigneter, andere passen auf alle Teststufen. Wenn Testfälle erstellt werden, nutzen Tester in der Regel eine Kombination aus verschiedenen Testverfahren, um mit gegebenem Testaufwand die besten Ergebnisse zu erzielen.

Die Nutzung von Testverfahren in der Testanalyse, im Testentwurf und in der Testrealisierung kann von sehr informell (wenig bis gar keine Dokumentation) bis hin zu sehr formal reichen. Der passende Grad an Formalität hängt vom Kontext des Testens ab, u.a. von der Reife des Test- und Entwicklungsprozesses, zeitlichen Beschränkungen, Informationssicherheits- oder regulatorischen Anforderungen, den Kenntnissen und Fähigkeiten der involvierten Personen und dem eingesetzten Softwareentwicklungslebenszyklus-Modell.

4.1.1 Kategorien von Testverfahren und ihre Eigenschaften

In diesem Lehrplan werden Testverfahren als Black-Box-, White-Box- oder erfahrungsbasierte Testverfahren klassifiziert.

Black-Box-Testverfahren (auch spezifikationsbasierte Verfahren genannt) basieren auf einer Analyse der zugehörigen Testbasis (z.B. formale Anforderungsdokumente, Spezifikationen, Anwendungsfälle, User-Stories oder Geschäftsprozesse). Diese Verfahren können sowohl auf funktionale als auch auf nichtfunktionale Tests angewendet werden. Black-Box-Testverfahren konzentrieren sich auf die Eingaben und Ausgaben des Testobjekts, ohne seine interne Struktur zu berücksichtigen.

White-Box-Testverfahren (auch strukturelle oder strukturbasierte Verfahren genannt) basieren auf einer Analyse der Architektur, dem Feinentwurf, der internen Struktur oder dem Code des Testobjekts. Anders als Black-Box-Testverfahren konzentrieren sich White-Box-Testverfahren auf die Struktur und die Abläufe innerhalb des Testobjekts.

Erfahrungsbasierte Testverfahren nutzen die Erfahrung von Entwicklern, Testern und Benutzern, um Tests zu entwerfen, umzusetzen und auszuführen. Diese Verfahren werden oft mit Black-Box- und White-Box-Verfahren kombiniert.

Gängige Merkmale von Black-Box-Testverfahren sind u.a.:

- Testbedingungen, Testfälle und Testdaten werden aus einer Testbasis abgeleitet, die Softwareanforderungen, Spezifikationen, Anwendungsfälle und User-Stories beinhalten kann.
- Testfälle können genutzt werden, um Lücken zwischen den Anforderungen und der Realisierung der Anforderungen sowie Abweichungen von den Anforderungen zu erkennen.
- Die Überdeckung wird anhand der getesteten Elemente in der Testbasis gemessen und aufgrund der Verfahren, die auf die Testbasis angewendet wird.

Gängige Merkmale von White-Box-Testverfahren sind u.a.:

- Testbedingungen, Testfälle und Testdaten werden aus einer Testbasis abgeleitet, die Code, Softwarearchitektur, Feinentwurf oder andere Arten an Informationen zur Struktur der Software enthalten kann.
- Die Überdeckung wird auf Basis der getesteten Elemente innerhalb einer ausgewählten Struktur gemessen (z.B. dem Code oder den Schnittstellen) und das Verfahren auf die Testbasis angewendet.

Gängige Merkmale von erfahrungsbasierten Testverfahren sind u.a.:

- Testbedingungen, Testfälle und Testdaten werden aus einer Testbasis abgeleitet, die schlicht aus den Kenntnissen und Erfahrungen der Tester, Entwickler, Benutzer und anderer Stakeholder bestehen kann.
- Diese Kenntnisse und Erfahrungen beinhalten die erwartete Nutzung der Software, ihrer Umgebung, mögliche Fehlerzustände und die Verteilung dieser Fehlerzustände.

Der internationale Standard ISO/IEC/IEEE 29119-4 enthält Beschreibungen von Testverfahren und ihren zugehörigen Überdeckungsmaßen (siehe Craig 2002 und Copeland 2004 für weitere Informationen über Verfahren).

4.2 Black-Box-Testverfahren

4.2.1 Äquivalenzklassenbildung

Äquivalenzklassenbildung teilt Daten so in Äquivalenzklassen auf (auch als Partitionen bezeichnet), dass alle Elemente einer vorgegebenen Äquivalenzklasse erwartungsgemäß in derselben Art und Weise verarbeitet werden (siehe Kaner 2013 und Jorgensen 2014). Es gibt Äquivalenzklassen für gültige und ungültige Werte.

- Gültige Werte sind Werte, die von der Komponente oder dem System akzeptiert werden sollten. Eine Äquivalenzklasse, die gültige Werte enthält, wird “gültige Äquivalenzklasse” genannt.
- Ungültige Werte sind Werte, die von der Komponente oder dem System abgelehnt werden sollten. Eine Äquivalenzklasse, die ungültige Werte enthält, wird “ungültige Äquivalenzklasse” genannt.
- Äquivalenzklassen können für jedes Datenelement in Bezug auf das Testobjekt gebildet werden, u.a. für Eingaben, Ausgaben, interne Werte, zeitbezogene Werte (z.B. vor oder nach einem Ereignis) und für Schnittstellenparameter (z.B. integrierte Komponenten, die während Integrationstests getestet werden).
- Jede Klasse kann (wenn nötig) in Unterklassen unterteilt werden.
- Jeder Wert muss eindeutig einer einzigen Äquivalenzklasse zugeordnet werden können.
- Wenn ungültige Äquivalenzklassen in Testfällen verwendet werden, sollten sie individuell getestet werden, d.h. nicht in Kombination mit anderen ungültigen Äquivalenzklassen, um sicherzustellen,

dass Fehlerwirkungen nicht maskiert bleiben. Fehlerwirkungen können maskiert bleiben, wenn mehrere Fehlerwirkungen zur gleichen Zeit auftreten, aber nur eine sichtbar ist, so dass die anderen Fehlerwirkungen unentdeckt bleiben.

Um mit diesem Verfahren eine 100%ige Überdeckung zu erzielen, müssen die Testfälle alle festgelegten Klassen (auch ungültige Klassen) abdecken, indem sie mindestens einen Wert aus jeder Klasse nutzen. Die Überdeckung wird gemessen durch die Anzahl der Äquivalenzklassen, die durch mindestens einen Wert getestet wurden, dividiert durch die Gesamtzahl der identifizierten Äquivalenzklassen, üblicherweise in Prozent ausgedrückt. Äquivalenzklassenbildung kann auf allen Teststufen angewendet werden.

4.2.2 Grenzwertanalyse

Die Grenzwertanalyse ist eine Erweiterung der Äquivalenzklassenbildung, aber sie kann nur dann genutzt werden, wenn die Klasse geordnet ist und aus numerischen oder sequenziellen Daten besteht. Die Minimum- und Maximum-Werte (oder erste und letzte Werte) einer Klasse sind ihre Grenzwerte (siehe Beizer 1990).

Nehmen wir zum Beispiel an, ein Eingabefeld akzeptiert einen einstelligen Integer-Wert als Eingabe und nutzt eine Zifferntaste zur Einschränkung der Eingabe, damit nur Integer-Werte als Eingabe möglich sind. Der gültige Bereich reicht von 1 bis einschließlich 5. Es gibt also drei Äquivalenzklassen: ungültig (zu niedrig), gültig, ungültig (zu hoch). Für die gültige Äquivalenzklasse sind die Grenzwerte 1 und 5. Für die ungültige (zu hohe) Klasse ist der Grenzwert 6. Für die ungültige (zu niedrige) Klasse gibt es nur einen Grenzwert, 0, da dies eine Klasse mit nur einem Wert ist.

Im obigen Beispiel identifizieren wir zwei Grenzwerte pro Grenze. Die Grenze zwischen ungültig (zu niedrig) und gültig ergibt die Testwerte 0 und 1. Die Grenze zwischen gültig und ungültig (zu hoch) ergibt die Testwerte 5 und 6. Einige Varianten dieses Verfahrens identifizieren drei Grenzwerte pro Grenze: die Werte vor, auf und gerade über der Grenze. Im vorigen Beispiel würde die Nutzung von drei Grenzwerten zu den niedrigen Testwerten 0, 1 und 2 und zu den höheren Testwerten 4, 5 und 6 führen (siehe Jorgensen 2014).

Das Verhalten an den Grenzen der Äquivalenzklasse ist wahrscheinlich eher nicht so korrekt wie das Verhalten innerhalb der Klassen. Es ist wichtig, sich zu vergegenwärtigen, dass sowohl die spezifizierten als auch die implementierten Grenzen über oder unterhalb ihrer eigentlich vorzusehenden Position verschoben, ganz ausgelassen oder mit unerwünschten zusätzlichen Grenzen ergänzt sein können. Grenzwertanalysen und -tests decken fast alle diese Fehlerzustände auf, indem sie die Software provozieren, ein anderes Verhalten zu zeigen, als es für die Klasse zu erwarten wäre, zu der die Grenzwerte gehören.

Grenzwertanalysen können auf allen Teststufen angewandt werden. Dieses Verfahren wird üblicherweise dazu genutzt, Anforderungen zu testen, die eine Reihe von Zahlen (einschließlich Datum und Zeit) enthalten. Grenzwertüberdeckung für eine Klasse wird gemessen als die Anzahl der getesteten Grenzwerte dividiert durch die Gesamtzahl der identifizierten Grenzwerte – meist ausgedrückt als Prozentzahl.

4.2.3 Entscheidungstabellentests

Entscheidungstabellen sind sinnvoll, um komplexe Regeln in Geschäftsprozessen zu erfassen, die ein System umzusetzen hat. Bei der Erstellung der Entscheidungstabellen identifiziert der Tester Bedingungen (zumeist Eingaben) und die daraus folgenden Aktionen (zumeist Ausgaben) des Systems. Diese bilden die Zeilen einer Tabelle, üblicherweise dann auch mit den Bedingungen oben und den Aktionen unten. Jede Spalte bezieht sich auf eine Entscheidungsregel, bestehend aus einer spezifischen Kombination von Bedingungen und den damit verbundenen Aktionen. Die Werte der Bedingungen und Aktionen werden üblicherweise als boolesche Werte dargestellt (wahr oder falsch) oder als eigenständige Werte (z.B. rot, grün, blau), können aber auch Zahlen oder Zahlenreihen sein. Derlei unterschiedliche Arten von Bedingungen und Aktionen können auch zusammen in der gleichen Tabelle auftreten.

Die gängige Notation in Entscheidungstabellen sieht wie folgt aus:

Für Bedingungen:

- J bedeutet, die Bedingung ist richtig (kann auch als Ja oder 1 dargestellt werden).
- N bedeutet, die Bedingung ist falsch (kann auch als Nein oder 0 bezeichnet werden).
- – bedeutet, der Wert der Bedingung hat für die Aktionen keine Bedeutung (kann auch als N/A bezeichnet werden).

Für Aktionen:

- X bedeutet, die Aktion sollte stattfinden (kann auch als Ja oder J oder 1 bezeichnet werden).
- Leer bedeutet, die Aktion sollte nicht stattfinden (kann auch als – oder Nein oder N oder 0 bezeichnet werden).

Eine vollständige Entscheidungstabelle hat genau so viele Spalten (Testfälle), dass jede Kombination von Bedingungen abgedeckt ist. Durch das Löschen von Spalten mit Bedingungskombinationen, die das Ergebnis nicht beeinflussen, kann sich die Anzahl der Testfälle erheblich verringern. Zum Beispiel durch das Entfernen von unmöglichen Kombinationen von Bedingungen. Für weitere Informationen dazu, wie Entscheidungstabellen optimiert werden können, siehe ISTQB-ATA).

Der übliche Mindestüberdeckungsstandard für Entscheidungstabellentests besteht darin, pro Entscheidungsregel der Tabelle mindestens einen Testfall zu haben. Das beinhaltet üblicherweise das Abdecken aller Kombinationen von Bedingungen. Die Überdeckung wird gemessen als die Anzahl der Entscheidungsregeln, die durch wenigstens einen Testfall getestet wurden, dividiert durch die Gesamtzahl der Entscheidungsregeln – üblicherweise als Prozentzahl dargestellt.

Die Stärke von Entscheidungstabellentests besteht darin, dass sie dabei helfen, alle wichtigen Kombinationen von Bedingungen zu identifizieren, die ansonsten leicht übersehen werden könnten. Sie helfen auch, Lücken in den Anforderungen zu finden. Sie können in jeder Teststufe auf alle Situationen angewendet werden, in denen das Verhalten der Software von einer Kombination von Bedingungen abhängt.

4.2.4 Zustandsübergangstest

Komponenten oder Systeme können unterschiedlich auf ein Ereignis reagieren, abhängig vom gegenwärtigen Zustand oder von Abläufen in der Vergangenheit (z.B. den Vorgängen, die seit Inbetriebnahme des Systems stattgefunden haben). Die Abläufe in der Vergangenheit können durch das Modell von Zuständen abgebildet werden. Ein Zustandsübergangsdiagramm zeigt zum einen alle möglichen Softwarezustände selbst und dann auch, wie die Software in diesen Zustand eintritt, austritt und zwischen den Zuständen wechselt. Ein Übergang wird durch ein Ereignis hervorgerufen (z.B. Benutzereingabe eines Werts in ein Feld). Das Ereignis führt zu einem Übergang. Das gleiche Ereignis kann zu zwei oder mehreren Übergängen aus dem gleichen Zustand heraus führen. Die Zustandsänderung kann häufig Reaktionen der Software zur Folge haben (z.B. Ausgabe einer Berechnung oder einer Fehlermeldung).

Eine Zustandsübergangstabelle zeigt alle gültigen Übergänge und auch alle ungültigen, aber möglichen Übergänge zwischen Zuständen auf. Außerdem enthält sie alle Ereignisse und zu erwartende Reaktionen der Software für gültige Übergänge. Zustandsübergangsdiagramme zeigen üblicherweise nur die gültigen Übergänge und schließen ungültigen Übergänge aus.

Es werden Tests entworfen, um eine übliche Sequenz von Zuständen abzubilden, um alle Zustände herbeizuführen, um jeden Übergang auszuführen, um spezifische Sequenzen von Übergängen auszuführen oder schließlich um ungültige Übergänge zu testen.

Zustandsübergangstest werden für menügeführte Anwendungen genutzt und häufig in der Softwarebranche für eingebettete Systeme (embedded systems) eingesetzt. Das Verfahren ist auch gut für die Modellerstellung eines Geschäftsszenarios mit spezifischen Zuständen geeignet oder für das Testen der Bildschirmnavigation. Das Konzept eines Zustands ist abstrakt – es kann sich dabei um einige Zeilen Code oder gar um einen gesamten Geschäftsprozess handeln.

Die Überdeckung wird üblicherweise gemessen als die Anzahl der ermittelten Zustände oder Übergänge, die getestet wurden, dividiert durch die Gesamtzahl der ermittelten Zustände oder Übergänge im Testobjekt überhaupt – üblicherweise als Prozentzahl dargestellt. Für weitere Informationen zu Überdeckungskriterien für Zustandsübergangstest siehe ISTQB-ATA.

4.2.5 Anwendungsfallbasierter Test

Anwendungsfälle (Use Cases) sind eine spezielle Form, um Interaktionen mit Softwareelementen darzustellen. Sie enthalten Anforderungen an die Softwarefunktionalität. Anwendungsfälle haben Akteure (menschliche Benutzer, externe Hardware oder andere Komponenten oder Systeme) und Objekte (eine Komponente oder das System, auf das der Anwendungsfall angewendet wird).

Jeder Anwendungsfall definiert ein bestimmtes Verhalten, das ein Objekt in Zusammenarbeit mit einem oder mehreren Akteuren ausführen kann (UML 2.5.1 2017). Ein Anwendungsfall kann durch Interaktionen und Aktivitäten sowie durch Vorbedingungen, Nachbedingungen und falls angemessen auch in natürlicher Sprache beschrieben werden. Interaktionen zwischen den Akteuren und dem Objekt können zu Veränderungen des Objektzustands führen. Interaktionen können grafisch durch Workflows, Aktivitätsdiagramme oder Geschäftsprozessmodelle dargestellt werden.

Ein Anwendungsfall besteht aus mehreren möglichen Varianten seines grundlegenden Verhaltens, was u.a. Sonder- und Fehlerbehandlungen einschließt (Antwort- und Wiederherstellungsmechanismen des Systems nach Programmier-, Anwendungs- und Kommunikationsfehler, die z.B. zu Fehlermeldungen führen). Tests werden entworfen, um das definierte Verhalten nachzuweisen (grundlegendes, außergewöhnliches oder alternatives Verhalten und die Fehlerbehandlungsroutinen). Die Überdeckung kann durch den Anteil der getesteten Anwendungsfallvarianten bezogen auf die Gesamtzahl der Anwendungsfallvarianten gemessen werden – und wird üblicherweise als Prozentsatz dargestellt.

Für weitere Informationen zu den Überdeckungskriterien für anwendungsfallbasierte Tests siehe ISTQB-ATA.

4.3 White-Box-Testverfahren

White-Box-Tests basieren auf der internen Struktur des Testobjekts. White-Box-Testverfahren können in allen Teststufen genutzt werden, aber die beiden codebasierten Verfahren, die in diesem Abschnitt beschrieben sind, werden am häufigsten in der Komponententeststufe genutzt. Es gibt weiterführende Verfahren, die zur genaueren Überdeckung in einigen sicherheitskritischen, einsatzkritischen oder hochintegrierten Umgebungen eingesetzt werden, aber diese werden hier nicht weiter behandelt. Für weitere Informationen zu diesen Verfahren siehe ISTQB-TTA.

4.3.1 Anweisungstests und -überdeckung

Anweisungstests untersuchen die potenziell ausführbaren Anweisungen im Code. Die Überdeckung wird an der Anzahl der im Test ausgeführten Anweisungen dividiert durch die Gesamtzahl aller ausführbaren Anweisungen insgesamt im Testobjekt gemessen – üblicherweise als Prozentsatz dargestellt.

4.3.2 Entscheidungstest und -überdeckung

Entscheidungstests untersuchen die Entscheidungen im Code und testen den Code, der auf Grundlage des Entscheidungsergebnisses ausgeführt wird. Dafür folgen die Testfälle bestimmten Kontrollflüssen, die von einem Entscheidungspunkt ausgehen (z.B. „wahr“ und „falsch“ bei einer IF-Anweisung, für eine CASE-Anweisung wären Testfälle für alle möglichen Ergebnisse nötig, auch für das Standardergebnis).

Die Überdeckung wird gemessen anhand der Anzahl der Entscheidungsergebnisse, die durch die Tests ausgeführt werden, dividiert durch die Gesamtzahl an möglichen Entscheidungsergebnissen im Testobjekt – üblicherweise als Prozentsatz dargestellt.

4.3.3 Der Beitrag von Anweisungs- und Entscheidungstests

100% Anweisungsüberdeckung bedeutet, dass alle potenziell ausführbaren Anweisungen im Code auch mindestens einmal ausgeführt wurden. Es stellt aber nicht sicher, dass die komplette Entscheidungslogik getestet wurde. Von den zwei in diesem Lehrplan beschriebenen White-Box-Verfahren kann der Anweisungstest weniger Überdeckung als der Entscheidungstest erzielen.

Wenn 100% Entscheidungsüberdeckung erzielt wird, führt dies alle Entscheidungsergebnisse aus, inklusive eines Tests des „wahr“-Ergebnisses und auch des „falsch“-Ergebnisses, selbst wenn es keine ausdrückliche Anweisung für den „falsch“-Fall gibt (z.B. für den Fall, dass eine IF-Anweisung keine ELSE-Anweisung im Code enthält). Anweisungsüberdeckung hilft Fehlerzustände im Code zu finden, der durch andere Tests nicht ausgeführt wurde. Entscheidungsüberdeckung hilft Fehlerzustände im Code zu finden, für den andere Tests nicht alle "wahr"/"falsch"-Fälle ausgeführt haben.

Das Erzielen von 100% Entscheidungsüberdeckung garantiert 100% Anweisungsüberdeckung (aber nicht umgekehrt).

4.4 Erfahrungsbasierte Testverfahren

Bei der Anwendung von erfahrungsbasierten Testverfahren werden die Testfälle aus den Kenntnissen und der Intuition des Testers heraus entwickelt sowie aus seiner Erfahrung mit ähnlichen Anwendungen und Technologien. Diese Verfahren können Testfälle identifizieren, die durch andere systematischere Verfahren nicht so leicht zu identifizieren wären. Abhängig vom Ansatz und der Erfahrung des Testers können diese Verfahren stark unterschiedliche Überdeckungen und Effizienz erreichen. Die Überdeckung ist bei diesen Verfahren schwer zu beurteilen und möglicherweise nicht messbar.

Gängige erfahrungsbasierte Testverfahren werden in den nächsten Abschnitten beschrieben.

4.4.1 Intuitive Testfallermittlung

Intuitive Testfallermittlung ist ein Verfahren, das das Auftreten von Fehlhandlungen, Fehlerzuständen und Fehlerwirkungen aufgrund des Wissens des Testers vermutet, u.a.:

- Wie hat die Anwendung früher funktioniert?
- Welche Arten von Fehlhandlungen werden üblicherweise gemacht?
- Fehlerwirkungen, die in anderen Anwendungen aufgetreten sind

Ein methodischer Ansatz für das Verfahren der intuitiven Testfallermittlung ist das Erstellen einer Liste möglicher Fehlhandlungen, Fehlerzustände und Fehlerwirkungen, zu der anschließend Tests entworfen werden, die die erwarteten Fehlerwirkungen und die Fehlerzustände offenlegen. Diese Fehlhandlungs-, Fehlerzustands- und Fehlerwirkungslisten können aufgrund von Erfahrungen oder Informationen über Fehlerzustands- und Fehlerwirkungen oder auch aufgrund allgemeiner Kenntnis darüber, warum Software fehlschlägt, erstellt werden.

4.4.2 Exploratives Testen

In explorativen Tests werden informelle (nicht vordefinierte) Tests während der Testdurchführung dynamisch entworfen, ausgeführt, aufgezeichnet und ausgewertet. Die Testergebnisse werden genutzt, um mehr über die Komponente oder das System zu erfahren und um Tests für die Bereiche zu vertiefen, die mehr Tests erfordern.

Exploratives Testen wird manchmal mit Hilfe von sitzungsbasiertem Testen (session-based testing) durchgeführt, um die Ausführung zu strukturieren. Beim sitzungsbasierten Testen werden explorative Tests innerhalb eines definierten Zeitfensters durchgeführt und der Tester nutzt eine Test-Charta mit Testzielen, die ihn beim Testen anleiten. Der Tester kann Sitzungsblätter (Session-Sheets) verwenden, um die ausgeführten Schritte und die Ergebnisse zu dokumentieren.

Exploratives Testen ist dort am nützlichsten, wo es wenig oder ungenügende Spezifikationen oder einen besonderen Zeitdruck für das Testen gibt. Exploratives Testen ist ebenso nützlich, um andere formalere Testverfahren zu ergänzen.

Exploratives Testen steht im starken Zusammenhang mit reaktiven Testverfahren (siehe Abschnitt *5.2.2 Teststrategie und Testvorgehensweise*). Exploratives Testen kann die Nutzung anderer Black-Box-, White-Box- oder erfahrungsbasierter Verfahren beim Testen mit einbeziehen.

4.4.3 Checklistenbasiertes Testen

Beim checklistenbasierten Testen entwerfen, realisieren und führen Tester Tests mit dem Ziel durch, alle Testbedingungen aus einer Checkliste abzudecken. Als Teil der Analyse erstellen Tester eine neue Checkliste oder erweitern die bestehende Checkliste. Tester können aber auch eine bestehende Checkliste ohne Änderungen verwenden. Solche Checklisten können auf Grundlage von Erfahrungen, Wissen über das Wesentliche in der Nutzung oder einem guten Verständnis darüber, warum und wie Software fehlschlägt, erstellt werden.

Checklisten können erstellt werden, um verschiedene Testarten zu unterstützen, einschließlich funktionaler und nicht-funktionaler Tests. Beim Fehlen detaillierter Testfälle können checklistenbasierte Tests eine gute Hilfestellung geben und ein bestimmtes Maß an Konsistenz liefern. Da es sich hier um Listen auf einer eher allgemeineren Ebene handelt, ist es wahrscheinlich, dass eine gewisse Variabilität beim eigentlichen Testen auftritt. Diese Variabilität führt einerseits zu einer potenziell höheren Überdeckung, aber andererseits zu einer geringeren Wiederholbarkeit.

5. Testmanagement – 225 Minuten

Schlüsselbegriffe

Eingangskriterien, Endekriterien, Fehlerbericht, Fehlermanagement, Konfigurationsmanagement, Produktrisiko, Projektrisiko, Risiko, risikobasiertes Testen, Risikostufe, Testabschlussbericht, Tester, Testfortschrittsbericht, Testkonzept, Testmanager, Testplanung, Testschätzung, Teststeuerung, Teststrategie, Testüberwachung, Testvorgehensweise

Lernziele für Testmanagement

5.1 Testorganisation

FL-5.1.1 (K2) Vor- und Nachteile unabhängigen Testens erklären können

FL-5.1.2 (K1) Die Aufgaben eines Testmanagers und eines Testers benennen können

5.2 Testplanung und -schätzung

FL-5.2.1 (K2) Den Zweck und Inhalt eines Testkonzepts zusammenfassen können

FL-5.2.2 (K2) Zwischen verschiedenen Teststrategien unterscheiden können

FL-5.2.3 (K2) Beispiele für mögliche Eingangs- und Endekriterien geben können

FL-5.2.4 (K3) Wissen über Priorisierung sowie technische und logische Abhängigkeiten anwenden können, um die Testdurchführung für ein gegebenes Testfallset zu planen

FL-5.2.5 (K1) Faktoren benennen können, die den Testaufwand beeinflussen

FL-5.2.6 (K2) Den Unterschied zwischen zwei Schätzverfahren erklären können: das metrikbasierte Verfahren und das expertenbasierte Verfahren

5.3 Testüberwachung und -steuerung

FL-5.3.1 (K1) Testmetriken wiedergeben können

FL-5.3.2 (K2) Zweck, Inhalte und Zielgruppen für Testberichte zusammenfassen können

5.4 Konfigurationsmanagement

FL-5.4.1 (K2) Zusammenfassen können, wie Konfigurationsmanagement das Testen unterstützt

5.5 Risiken und Testen

FL-5.5.1 (K1) Risikostufe anhand der Wahrscheinlichkeit (des Eintritts) und Auswirkung (im Schadensfall) definieren können

FL-5.5.2 (K2) Zwischen Projekt- und Produktrisiken unterscheiden können

FL-5.5.3 (K2) Anhand von Beispielen beschreiben können, wie die Produktrisikoanalyse Intensität und Umfang des Testens beeinflussen kann

5.6 Fehlermanagement

FL-5.6.1 (K3) Einen Fehlerbericht schreiben können, der einen während des Testens gefundenen Fehler enthält

5.1 Testorganisation

5.1.1 Unabhängiges Testen

Testaufgaben können von Personen in einer spezifischen Testrolle oder von Personen in einer anderen Rolle (z.B. Kunden) durchgeführt werden. Ein gewisser Grad an Unabhängigkeit erhöht die Effektivität des Testers beim Finden von Fehlerzuständen, da Autoren und Tester unterschiedliche kognitive Ausrichtungen haben (siehe Abschnitt *1.5 Die Psychologie des Testens*). Unabhängigkeit ist jedoch kein Ersatz für Vertrautheit und Entwickler können in ihrem eigenen Code in effizienter Art und Weise viele Fehlerzustände finden.

Die folgenden beispielhaften Testorganisationen unterscheiden sich im Grad der Unabhängigkeit (von gering bis hoch):

- Keine unabhängigen Tester; die einzig verfügbare Form von Tests sind Entwicklertests, bei denen Entwickler ihren eigenen Code testen
- Unabhängige Entwickler oder Tester innerhalb des Entwicklungsteams oder des Projektteams; dies können Entwickler sein, die die Arbeitsergebnisse ihrer Kollegen testen
- Ein unabhängiges Testteam oder eine Gruppe innerhalb des Unternehmens, die an das Projektmanagement oder die Unternehmensleitung berichtet
- Unabhängige Tester innerhalb des Unternehmens aus dem Fachbereich oder aus der Gruppe der Anwender oder mit Spezialisierungen auf bestimmte Testarten wie Gebrauchstauglichkeit, IT-Sicherheit, Performanz, Regulatorik/Compliance oder Übertragbarkeit
- Unabhängige, nicht zum Unternehmen gehörende Tester, die entweder vor Ort (Inhouse) oder außerhalb des Betriebs (Outsourcing) arbeiten

Für die meisten Projektarten ist es in der Regel von Vorteil, mehrere Teststufen zu durchlaufen, bei denen einige dieser Stufen von unabhängigen Testern durchgeführt werden. Entwickler sollten sich am Test beteiligen, besonders auf den unteren Teststufen. Somit können Entwickler die Qualität ihrer eigenen Arbeit kontrollieren.

Die Art und Weise, wie unabhängiges Testen realisiert wird, variiert in Abhängigkeit des Softwareentwicklungslebenszyklus-Modells. In der agilen Entwicklung können Tester zum Beispiel Teil des Entwicklungsteams sein. In einigen Unternehmen, die agile Methoden verwenden, können diese Tester auch zu einem größeren unabhängigen Testteam gehören. Darüber hinaus können Product Owner in solchen Unternehmen Abnahmetests durchführen, um User-Stories am Ende jeder Iteration zu validieren.

Mögliche Vorteile von Testunabhängigkeit sind u.a.:

- Unabhängige Tester werden wahrscheinlich andere Fehlerwirkungen erkennen als Entwickler, da sie unterschiedliche Hintergründe, unterschiedliche technische Sichtweisen und unterschiedliche Voreingenommenheit haben.
- Ein unabhängiger Tester kann Annahmen, die Stakeholder während der Spezifikation und Realisierung des Systems machten, verifizieren, in Frage stellen oder widerlegen.
- Unabhängige Tester eines Anbieters können auf korrekte und objektive Weise über das zu testende System berichten, ohne dass das Unternehmen, das sie eingestellt hat (politischen) Druck ausübt.

Mögliche Nachteile von Testunabhängigkeit sind u.a.:

- Die Isolation vom Entwicklungsteam kann zu fehlender Zusammenarbeit, verzögerten Rückmeldungen an das Entwicklungsteam oder zu einem feindlichen Verhältnis zum Entwicklungsteam führen.

- Entwickler können das Verantwortungsbewusstsein für Qualität verlieren.
- Unabhängige Tester können als Engpass angesehen werden.
- Unabhängigen Testern fehlen unter Umständen wichtige Informationen (z.B. über das Testobjekt).

Viele Unternehmen sind fähig, die Vorteile der Testunabhängigkeit zu verwirklichen und die Nachteile zu vermeiden.

5.1.2 Aufgaben eines Testmanagers und eines Testers

In diesem Lehrplan werden zwei Testrollen behandelt, und zwar Testmanager und Tester. Die Aktivitäten und Aufgaben dieser beiden Rollen hängen vom Projekt- und Produktkontext, von den Fähigkeiten der Menschen in diesen Rollen und dem Unternehmen ab.

Der Testmanager ist verantwortlich für den allgemeinen Testfortschritt und die erfolgreiche Leitung der Test-aktivitäten. Die Testmanagementrolle kann von einem professionellen Testmanager oder von einem Projektmanager, einem Entwicklungsmanager oder einem Qualitätssicherungsmanager eingenommen werden. In größeren Projekten oder Unternehmen können mehrere Testteams an einen Testmanager, Testcoach oder Testkoordinator berichten und jedes Team wird durch einen Testleiter oder den leitenden Tester geführt.

Typische Aufgaben eines Testmanagers sind u.a.:

- Eine Testrichtlinie und Teststrategie für das Unternehmen entwickeln oder prüfen
- Die Testaktivitäten unter Berücksichtigung des Kontexts und des Verständnisses der Testziele und Risiken planen. Das kann die Auswahl von Testvorgehensweisen, die Schätzung der Testdauer, des Aufwands und der Kosten, die Beschaffung von Ressourcen, die Festlegung von Teststufen und Testzyklen und die Planung des Fehlermanagements beinhalten
- Testkonzepte schreiben und aktualisieren
- Testkonzepte mit den Projektmanagern, Product Ownern und weiteren Beteiligten abstimmen
- Einbringen der Testperspektive in andere Projektaktivitäten, beispielsweise in die Integrationsplanung
- Die Analyse, den Entwurf, die Realisierung und die Durchführung von Tests anstoßen, den Testfortschritt und die Testergebnisse überwachen und den Stand der Endekriterien (oder der Definition-of-Done) prüfen sowie die Testabschlussaktivitäten ermöglichen
- Testfortschrittsberichte und Testabschlussberichte auf der Grundlage der gesammelten Informationen erstellen und verteilen
- Die Planung auf der Grundlage von Testergebnissen und Fortschritt anpassen (manchmal dokumentiert in Testfortschrittsberichten und/oder Testabschlussberichten für andere Tests, die im Projekt bereits abgeschlossen sind) und notwendige Maßnahmen zur Teststeuerung in die Wege leiten
- Das Aufsetzen des Fehlermanagementsystems und eines angemessenen Konfigurationsmanagements für Testmittel unterstützen
- Geeignete Metriken für die Messung des Testfortschritts und die Bewertung der Qualität des Testens und des Produkts einführen
- Unterstützung bei der Auswahl und dem Einsatz von Werkzeugen für den Testprozess, einschließlich der Empfehlung des Budgets für die Werkzeugauswahl (und möglicherweise Kauf und/oder Support), der Zuordnung von Zeit und Aufwand für Pilotprojekte und der Bereitstellung von kontinuierlicher Unterstützung bei der Nutzung der Werkzeuge

- Über die Realisierung von Testumgebungen entscheiden
- Die Tester, das Testteam und das Berufsbild Tester innerhalb des Unternehmens entwickeln und fördern
- Die Fähigkeiten und Aufstiegsmöglichkeiten von Testern weiterentwickeln (bspw. durch Schulungspläne, Leistungsbeurteilungen, Coaching usw.)

Die Art und Weise, in der die Rolle des Testmanagers umgesetzt wird, variiert in Abhängigkeit vom Softwareentwicklungslebenszyklus. In der agilen Entwicklung beispielsweise werden einige der oben genannten Aufgaben, insbesondere solche, die die täglichen Tests innerhalb des Teams betreffen, vom agilen Team übernommen – häufig von einem im Team integrierten Tester. Einige der Aufgaben, die mehrere Teams oder das ganze Unternehmen betreffen oder die mit Personalmanagement zu tun haben, können von Testmanagern außerhalb des Entwicklungsteams wahrgenommen werden, die manchmal Test Coaches genannt werden. Siehe Black 2009 für weitere Informationen zum Management des Testprozesses.

Typische Aufgaben eines Testers sind u.a.:

- Testkonzepte prüfen und zu diesen beitragen
- Anforderungen, User-Stories und Abnahmekriterien, Spezifikationen und Modelle (d.h. die Testbasis) auf Testbarkeit analysieren, prüfen und beurteilen
- Die Testbedingungen identifizieren und dokumentieren und die Verfolgbarkeit zwischen Testfällen, Testbedingungen und der Testbasis erfassen
- Die Testumgebung(en) entwerfen, einrichten und verifizieren, oft in Abstimmung mit der Systemadministration und dem Netzwerkmanagement
- Testfälle und Testabläufe entwerfen und realisieren
- Testdaten vorbereiten und beschaffen
- Den detaillierten Testausführungsplan erstellen
- Die Tests durchführen, die Ergebnisse bewerten und Abweichungen von den erwarteten Ergebnissen dokumentieren
- Geeignete Werkzeuge zur Unterstützung des Testprozesses verwenden
- Bei Bedarf Tests automatisieren (dies kann durch einen Entwickler oder Testautomatisierungsexperten unterstützt werden)
- Die nicht-funktionalen Eigenschaften wie Performanz, Zuverlässigkeit, Gebrauchstauglichkeit, IT-Sicherheit, Kompatibilität und Übertragbarkeit bewerten
- Tests prüfen, die von anderen entwickelt wurden

Personen, die an der Testanalyse, dem Testentwurf, spezifischen Testarten oder der Testautomatisierung arbeiten, können Spezialisten in diesen Rollen sein. Abhängig von den Risiken des Produkts oder des Projekts und dem ausgewählten Softwareentwicklungslebenszyklus-Modell können unterschiedliche Personen die Rolle des Testers in unterschiedlichen Teststufen wahrnehmen. Zum Beispiel übernimmt in der Komponententeststufe und der Komponentenintegrationsteststufe häufig ein Entwickler die Rolle des Testers. In der Abnahmeteststufe wird die Rolle des Testers oft von Businessanalysten, Fachexperten und Benutzern eingenommen. In der Systemtest- und Systemintegrationsteststufe wird die Rolle des Testers meist von einem unabhängigen Testteam übernommen. In der betrieblichen Abnahmeteststufe wird die Rolle des Testers häufig vom Betrieb und/oder von Mitarbeitern der Systemadministration wahrgenommen.

5.2 Testplanung und -schätzung

5.2.1 Zweck und Inhalt eines Testkonzepts

Ein Testkonzept legt die Testaktivitäten für Entwicklungs- und Wartungsprojekte fest. Die Planung wird durch die Testrichtlinie und die Teststrategie des Unternehmens sowie durch die verwendeten Entwicklungslebenszyklen und Verfahren (siehe Abschnitt *2.1 Softwareentwicklungslebenszyklus-Modelle*), den Testumfang, die Ziele, Risiken, Einschränkungen, Kritikalität, Testbarkeit und die Verfügbarkeit von Ressourcen beeinflusst.

Mit dem Fortschritt des Projekts und der Testplanung sind mehr Informationen verfügbar und mehr Details können in das Testkonzept einfließen. Die Testplanung ist eine kontinuierliche Aktivität und wird über den gesamten Produktlebenszyklus hinweg durchgeführt. (Beachten Sie, dass der Produktlebenszyklus über den Umfang eines Projekts hinausgehen kann, um auch die Wartungsphase einzubeziehen). Rückmeldungen aus Testaktivitäten sollten genutzt werden, um sich ändernde Risiken zu erkennen, damit die Planung angepasst werden kann. Die Planung kann in einem Mastertestkonzept und in einzelnen Testkonzepten für die jeweiligen Teststufen, wie Systemtest und Abnahmetest, oder für die einzelnen Testarten, wie Gebrauchstauglichkeitstests und Performanztests, dokumentiert werden. Testplanungsaktivitäten können die folgenden Aktivitäten enthalten und einige davon können in einem Testkonzept dokumentiert sein:

- Bestimmung des Umfangs, der Ziele und der Risiken der Tests
- Festlegen der allgemeinen Testvorgehensweise
- Integrieren und Koordinieren der Testaktivitäten in die Softwarelebenszyklusaktivitäten
- Treffen von Entscheidungen darüber, was zu testen ist, sowie über die Personen und andere Ressourcen, die notwendig sind, um die verschiedenen Testaktivitäten durchzuführen, und darüber, wie die Testaktivitäten durchgeführt werden
- Planen der Aktivitäten zur Testanalyse, zum Entwurf, zur Realisierung, zur Durchführung und zur Bewertung, entweder in festgelegten Zeiträumen (bspw. in der sequenziellen Entwicklung) oder im Kontext jeder Iteration (bspw. in der iterativen Entwicklung)
- Auswählen der Metriken zur Testüberwachung und -steuerung
- Festlegen des Budgets für die Testaktivitäten
- Bestimmung des Detaillierungsgrads und der Struktur für die Testdokumentation (bspw. durch Vorlagen oder Beispieldokumente)

Der Inhalt von Testkonzepten variiert und kann über die oben genannten Themen hinausgehen. Eine Musterstruktur für Testkonzepte und ein Beispieltestkonzept kann der ISO-Norm ISO/IEC/IEEE 29119-3 entnommen werden.

5.2.2 Teststrategie und Testvorgehensweise

Eine Teststrategie liefert eine verallgemeinernde Beschreibung des Testprozesses, üblicherweise auf Produkt- oder Organisationsebene. Gängige Arten von Teststrategien sind u.a.:

- **Analytisch:** Diese Art von Teststrategie basiert auf einer Analyse eines Faktors (z.B. Anforderung oder Risiko). Risikobasiertes Testen ist ein Beispiel für eine analytische Vorgehensweise, bei der Tests auf Grundlage der Risikostufe entworfen und priorisiert werden.
- **Modellbasiert:** In dieser Art von Teststrategie werden die Tests auf der Grundlage eines Modells eines geforderten Produktaspekts entworfen, z.B. einer Funktion, eines Geschäftsprozesses, einer internen Struktur oder einer nicht-funktionalen Eigenschaft (z.B. Zuverlässigkeit). Beispiele für sol-

che Modelle sind u.a. Geschäftsprozessmodelle, Zustandsmodelle und Zuverlässigkeitswachstumsmodelle.

- **Methodisch:** Diese Art der Teststrategie baut auf der systematischen Nutzung vordefinierter Sets von Tests oder Testbedingungen auf, wie einer Systematik von gängigen oder wahrscheinlichen Arten von Fehlerwirkungen, einer Liste von wichtigen Qualitätsmerkmalen oder unternehmensweiten Look-and-Feel-Standards für mobile Anwendungen oder Webseiten.
- **Prozesskonforme (oder standardkonforme):** Diese Art der Teststrategie beinhaltet die Analyse, den Entwurf und die Realisierung von Tests auf der Grundlage von externen Vorschriften und Standards, wie sie durch branchenspezifische Standards, durch Prozessdokumentation oder gründliche Identifizierung und Nutzung der Testbasis vorgegeben werden oder durch Prozesse oder Standards, die für das oder vom Unternehmen aufgestellt worden sind.
- **Angeleitete (oder beratende):** Diese Art der Teststrategie wird vorrangig durch Beratung, Anleitung oder Anweisungen von Stakeholdern, Fachexperten oder Technologieexperten bestimmt, die von außerhalb des Testteams oder sogar von außerhalb des Unternehmens kommen können.
- **Regressionsvermeidend (Regression-avers):** Diese Art der Teststrategie wird durch den Wunsch motiviert, **keinen** Verlust an vorhandener Leistungsfähigkeit im Test hinzunehmen. Diese Teststrategie beinhaltet die Wiederverwendung vorhandener Testmittel (insbesondere Testfälle und Testdaten), weitgehende Automatisierung von Regressionstests und die Nutzung von Standardtestsuiten.
- **Reaktiv:** Mit dieser Art der Teststrategie wird auf die Komponente oder das System und die während der Testdurchführung auftretenden Ereignisse reagiert im Gegensatz zu den vorherigen eher vorausgeplanten Strategien. Tests werden entworfen und realisiert und können umgehend als Reaktion auf Erkenntnisse, die durch vorherige Testergebnisse erworben wurden, durchgeführt werden. Exploratives Testen ist eine gängige Vorgehensweise in reaktiven Strategien.

Eine angemessene Teststrategie wird häufig durch die Kombination dieser verschiedenen Arten von Teststrategien erstellt. Zum Beispiel kann risikobasiertes Testen (eine analytische Strategie) mit explorativem Testen (einer reaktiven Strategie) kombiniert werden; sie ergänzen sich gegenseitig und können zusammengenommen effektiveres Testen gewährleisten.

Während die Teststrategie eine allgemeine Beschreibung des Testprozesses liefert, passt die Testvorgehensweise die Teststrategie für ein bestimmtes Projekt oder ein Release an. Die Testvorgehensweise ist der Ausgangspunkt für die Auswahl der Testverfahren, Teststufen und Testarten, für die Definition der Eingangs- und Endekriterien (bzw. Definition-of-Ready und Definition-of-Done). Die Anpassung der Strategie basiert auf Entscheidungen in Bezug auf die Komplexität und die Ziele des Projekts, die Art des zu entwickelnden Produkts und die Produktrisikoanalyse. Die ausgewählte Vorgehensweise hängt vom Kontext ab und kann Faktoren wie Risiken, funktionale Sicherheit, verfügbare Ressourcen und Fähigkeiten, Technologie, die Art des Systems (bspw. kundenspezifisch angefertigt gegenüber kommerzieller Standardsoftware), Testziele und Richtlinien einbeziehen.

5.2.3 Eingangs- und Endekriterien (Definition-of-Ready und Definition-of-Done)

Um effektive Kontrolle über die Qualität der Software und der Tests zu erzielen, ist es ratsam, Kriterien zu haben, die definieren, wann eine vorgegebene Testaktivität anfangen sollte und wann sie abgeschlossen ist. Eingangskriterien (in der agilen Entwicklung üblicherweise als Definition-of-Ready bezeichnet) definieren Vorbedingungen für die Durchführung einer bestimmten Testaktivität. Wenn Eingangskriterien nicht erfüllt sind, ist es wahrscheinlich, dass sich die Aktivität als schwieriger, zeitaufwendiger, kostspieliger und risikoreicher erweisen wird. Endekriterien (in der agilen Entwicklung üblicherweise Definition-of-Done genannt) definieren, welche Bedingungen erfüllt sein müssen, damit eine Teststufe oder ein Set von Testfällen als abgeschlossen bezeichnet werden kann. Eingangs- und Endekriterien sollten für jede Teststufe und jede Testart definiert sein und unterscheiden sich nach den Testzielen.

Übliche Eingangskriterien sind u.a.:

- Verfügbarkeit von testbaren Anforderungen, User-Stories und/oder Modellen (bspw., wenn eine modellbasierte Teststrategie verfolgt wird)
- Verfügbarkeit von Testelementen, die die Endekriterien für vorangegangene Teststufen erfüllt haben
- Verfügbarkeit einer Testumgebung
- Verfügbarkeit der notwendigen Testwerkzeuge
- Verfügbarkeit von Testdaten und anderen notwendigen Ressourcen

Übliche Endekriterien sind u.a.:

- Geplante Tests wurden durchgeführt
- Eine festgelegte Überdeckung (z.B. von Anforderungen, User-Stories, Abnahmekriterien, Risiken, Code) wurde erreicht
- Die Anzahl ungelöster Fehlerzustände bewegt sich innerhalb einer vereinbarten Grenze
- Die Anzahl von geschätzten verbleibenden Fehlerzuständen ist ausreichend gering
- Die bewerteten Niveaus an Zuverlässigkeit, Performanz, Gebrauchstauglichkeit, IT-Sicherheit und anderer relevanter Qualitätsmerkmale sind ausreichend

Selbst ohne die Erfüllung von Endekriterien ist es üblich, dass Testaktivitäten aufgrund eines ausgeschöpften Budgets, des Ablaufs der geplanten Zeitdauer und/oder aufgrund des Drucks, das Produkt auf den Markt zu bringen, begrenzt werden. Es kann zulässig sein, dass das Testen unter derartigen Umständen beendet wird, wenn die Stakeholder des Projekts und die Fachverantwortlichen das Risiko, ohne weitere Tests die Freigabe zu erteilen, geprüft und akzeptiert haben.

5.2.4 Testausführungsplan

Sobald die verschiedenen Testfälle und Testabläufe erstellt (eventuell mit einigen automatisierten Testabläufen) und in Testsuiten aufgenommen sind, können die Testsuiten in einem Testausführungsplan zusammengefasst werden, der die Reihenfolge definiert, in der sie ablaufen sollen. Der Testausführungsplan sollte Faktoren wie Priorisierung, Abhängigkeiten, Fehlernachtests, Regressionstests und die effizienteste Abfolge für die Durchführung der Tests berücksichtigen.

Idealerweise werden die Testfälle auf Basis ihrer Priorität geordnet, normalerweise indem die Testfälle mit der höchsten Priorität als Erstes durchgeführt werden. Wenn Testfälle oder die Features, die getestet werden, Abhängigkeiten aufweisen, kann es allerdings sein, dass diese Vorgehensweise nicht funktioniert. Wenn ein Testfall mit einer höheren Priorität von einem Testfall mit geringerer Priorität abhängt, muss der Testfall mit der geringeren Priorität zuerst ausgeführt werden. Ähnlich ist es, wenn Abhängigkeiten über Testfälle hinweg bestehen. Dann müssen diese, unabhängig von ihrer jeweiligen Priorität, in eine entsprechende Reihenfolge gebracht werden. Fehlernachtests und Regressionstests müssen ebenfalls priorisiert werden basierend auf der Bedeutung von schnellen Rückmeldungen nach Änderungen. Allerdings können auch hier wieder Abhängigkeiten bestehen.

Manchmal sind unterschiedliche Reihenfolgen von Tests möglich, die jeweils einen unterschiedlichen Grad an Effizienz haben. In diesen Fällen müssen Kompromisse zwischen der Effizienz der Testdurchführung und der Einhaltung von Priorisierungen geschlossen werden.

5.2.5 Den Testaufwand beeinflussende Faktoren

Testaufwandschätzung beinhalt die Voraussage über die Menge an testrelevanter Arbeit, die benötigt wird, um die Ziele des Testens für ein bestimmtes Projekt, ein Release oder eine Iteration zu erfüllen. Faktoren,

die den Testaufwand beeinflussen, können u.a. die Eigenschaften des Produkts, die Eigenschaften des Entwicklungsprozesses, Eigenschaften der beteiligten Personen und die Testergebnisse sein, wie unten dargestellt.

Produkteigenschaften

- Die Risiken, die mit dem Produkt einhergehen
- Die Qualität der Testbasis
- Die Größe des Produkts
- Die Komplexität des Produkteinsatzbereichs
- Die Anforderungen bezüglich Qualitätsmerkmalen (z.B. IT-Sicherheit, Zuverlässigkeit)
- Der geforderte Detaillierungsgrad der Testdokumentation
- Anforderungen an gesetzliche und regulatorische Konformität

Entwicklungsprozesseigenschaften

- Die Stabilität und Reife der Organisation
- Das genutzte Entwicklungsmodell
- Die Testvorgehensweise
- Die genutzten Werkzeuge
- Der Testprozess
- Zeitdruck

Menschliche Eigenschaften

- Die Fähigkeiten und Erfahrungen der beteiligten Personen, insbesondere mit ähnlichen Projekten und Produkten (z.B. Fachkenntnisse)
- Teamzusammenhalt und Leitung

Testergebnisse

- Die Anzahl und der Schweregrad von gefundenen Fehlerzuständen
- Die Menge an erforderlichen Nachbesserungen

5.2.6 Testschätzverfahren

Es gibt eine Reihe von Schätzverfahren, die genutzt werden, um den erforderlichen Aufwand für angemessenes Testen zu bestimmen. Zwei der am häufigsten verwendeten Verfahren sind die folgenden:

- Das metrikbasierte Verfahren: Schätzung des Testaufwands auf Basis der Metriken früherer ähnlicher Projekte oder auf Basis von typischen Werten
- Das expertenbasierte Verfahren: Schätzung des Testaufwands basierend auf der Erfahrung der für die Testaufgaben zuständigen Person oder von Experten

In der agilen Entwicklung sind zum Beispiel Burndown-Charts Beispiele eines metrikbasierten Vorgehens. Der verbleibende Aufwand wird bestimmt und berichtet und dann zusammen mit der Velocity (Maß für Produktivität im agilen Projekt) des Teams genutzt, um die Menge an Arbeit zu ermitteln, die das Team in der nächsten Iteration erfüllen kann. Das Planungspoker dagegen ist ein Beispiel für das expertenbasierte Verfahren, da Teammitglieder den Aufwand schätzen, der ihrer Erfahrung nach notwendig ist, um eine Funktionalität bereitzustellen (ISTQB-AT).

Innerhalb von sequenziellen Projekten sind Modelle der Fehlerbehebung Beispiele für das metrikbasierte Vorgehen, bei dem Mengen von Fehlerzuständen und die notwendige Zeit, sie zu entfernen, erfasst und berichtet werden, was dann eine Grundlage für die Schätzung zukünftiger Projekte ähnlicher Art bietet. Dagegen ist das Breitband-Delphi-Schätzverfahren ein Beispiel für ein expertenbasiertes Verfahren, in dem eine Gruppe von Experten Schätzungen auf Basis ihrer Erfahrung liefern (ISTQB-ATM).

5.3 Testüberwachung und -steuerung

Der Zweck der Testüberwachung ist es, Informationen zu sammeln sowie Feedback und eine Sicht über Testaktivitäten zu liefern. Zu überwachende Informationen können manuell oder automatisch gesammelt werden und sollten genutzt werden, um den Testfortschritt zu beurteilen und zu messen, ob die Testendekriterien oder die Testaufgaben, die mit einer Definition-of-Done in einem agilen Projekt einhergehen, erfüllt sind, also ob z.B. die Ziele für die Überdeckung von Produktrisiken, Anforderungen oder Abnahmekriterien erreicht sind.

Teststeuerung beschreibt alle leitenden oder korrigierenden Maßnahmen, die als Ergebnis der gesammelten und (möglicherweise) berichteten Informationen und Metriken unternommen werden. Maßnahmen können Testaktivitäten umfassen und jede andere Aktivität im Softwarelebenszyklus beeinflussen.

Beispiele für Teststeuerungsmaßnahmen sind u.a.:

- Neupriorisieren von Tests, wenn ein identifiziertes Risiko eintritt (z.B. Software wird spät geliefert)
- Anpassen des Testplans aufgrund von Verfügbarkeit oder Nichtverfügbarkeit einer Testumgebung oder anderer Ressourcen
- Neubewerten, ob ein Testelement nach Nachbesserungen ein Eingangs- oder Endekriterium erfüllt

5.3.1 Beim Testen verwendete Metriken

Metriken können während und am Ende von Testaktivitäten gesammelt werden, um Folgendes zu beurteilen:

- Fortschritt gegenüber dem geplanten Zeitplan und Budget
- Gegenwärtige Qualität des Testobjekts
- Angemessenheit der Testvorgehensweise
- Effektivität der Testaktivitäten in Bezug auf die Ziele

Gängige Testmetriken sind u.a.:

- Prozentsatz der durchgeführten, geplanten Arbeit in der Testfallvorbereitung (oder Prozentsatz der realisierten, geplanten Testfälle)
- Prozentsatz der durchgeführten, geplanten Arbeit in der Testumgebungsvorbereitung
- Testfalldurchführung (z.B. Anzahl der ausgeführten/nicht ausgeführten Testfälle, der erfolgreichen/fehlgeschlagenen Testfälle und/oder erfüllten/nicht erfüllten Testbedingungen)
- Fehlerinformation (z.B. Fehlerdichte, gefundene und behobene Fehlerzustände, Ausfallrate und Ergebnisse von Fehlernachtests)
- Testüberdeckung von Anforderungen, User-Stories, Abnahmekriterien, Risiken oder Code
- Aufgabenfertigstellung, Ressourcenverteilung und Nutzung und Aufwand

- Kosten des Testens, darunter auch die Kosten gegenüber dem Nutzen durch Auffinden des nächsten Fehlerzustands oder die Kosten im Vergleich zum Nutzen der Durchführung der nächsten Tests

5.3.2 Zwecke, Inhalte 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 (bspw. in einer Teststufe). Der Testbericht, der während einer Testaktivität erstellt wird, kann als Testfortschrittsbericht bezeichnet werden, während der Testbericht, der am Ende einer Testaktivität erstellt wird, Testabschlussbericht genannt werden kann.

Während der Testüberwachung und -steuerung erstellt der Testmanager regelmäßig Testfortschrittsberichte für die Stakeholder. Zusätzlich zu üblichen Inhalten für Testfortschrittsberichte und Testabschlussberichte sind die folgenden weiteren Inhalte durchaus gebräuchlich:

- Der Status der Testaktivitäten und der Fortschritt gegenüber dem Testkonzept
- Faktoren, die den Fortschritt behindern
- Tests, die für die nächste Berichtsperiode geplant sind
- Die Qualität des Testobjekts

Wenn die Endekriterien erreicht sind, erstellt der Testmanager den Testabschlussbericht. Dieser Bericht liefert eine Zusammenfassung der durchgeführten Tests auf Basis des letzten Testfortschrittsberichts und anderer relevanter Informationen.

Typische Testabschlussberichte können u.a. enthalten:

- Zusammenfassung der durchgeführten Tests
- Information darüber, was während einer Testperiode vorgefallen ist
- Abweichungen vom Plan einschließlich Abweichungen im Zeitplan, in der Dauer oder dem Aufwand von Testaktivitäten
- Stand der Tests und Produktqualität in Bezug auf die Endekriterien oder die Definition-of-Done
- Faktoren, die den Fortschritt blockiert haben oder weiterhin blockieren
- Metriken über Fehlerzustände, Testfälle, Testüberdeckung, Aktivitätsfortschritt und Ressourcenverbrauch (bspw. wie in Abschnitt *5.3.1 Beim Testen verwendete Metriken* beschrieben)
- Restrisiken (siehe Abschnitt 5.5 Risiken und Testen)
- Erzeugte wiederverwendbare Testarbeitsergebnisse

Die Inhalte eines Testberichts variieren in Abhängigkeit vom Projekt, den organisatorischen Anforderungen und dem Softwareentwicklungslebenszyklus. Beispielsweise wird ein komplexes Projekt mit vielen Stakeholdern oder ein regulatorisches Projekt detailliertere und genauere Berichte erfordern als eine schnelle Softwareaktualisierung. Als weiteres Beispiel ist die agile Entwicklung anzuführen, in der Testfortschrittsberichte in Task Boards, Fehlerzusammenfassungen und Burndown-Charts eingebunden sein können, die in einem täglichen Stand-up-Meeting besprochen werden können (siehe ISTQB-CTFL-AT).

Zusätzlich zur Anpassung von Testberichten auf Basis des Projektkontexts sollten Testberichte auch an die Zielgruppe des Berichts angepasst sein. Die Art und die Menge an Informationen, die für eine technische Zielgruppe oder ein Testteam zu berücksichtigen sind, können sich unterscheiden von dem, was in einer Zusammenfassung für das Management enthalten sein sollte. Im ersteren Fall können detaillierte Informationen über Fehlerarten und Trends von Bedeutung sein. Im letzteren Fall kann ein Bericht auf

abstrakter Ebene (z.B. ein Fehlerstatusreport nach Priorität, Budget, Zeitplan und erfüllten/nicht erfüllten/nicht getesteten Testbedingungen) angemessener sein.

Die ISO-Norm ISO/IEC/IEEE 29119-3 bezieht sich auf zwei Arten von Testberichten, die Testfortschrittsberichte und die Testabschlussberichte, und beinhaltet für beide Arten Strukturen und Beispiele.

5.4 Konfigurationsmanagement

Der Zweck des Konfigurationsmanagements ist es, die Integrität der Komponente oder des Systems, der Testmittel und ihrer Beziehungen untereinander durch das Projekt und den Produktlebenszyklus herzustellen und zu erhalten.

Um Tests angemessen zu unterstützen, kann das Konfigurationsmanagement Folgendes sicherstellen:

- Alle Testelemente sind eindeutig identifiziert, versionskontrolliert, werden in ihren Änderungen nachverfolgt und stehen in Verbindung zueinander.
- Alle Elemente der Testmittel sind eindeutig identifiziert, versionskontrolliert, werden in ihren Änderungen nachverfolgt, stehen in Verbindung zueinander und in Verbindung zu den Versionen der Testelemente, so dass die Verfolgbarkeit im Testprozess gewährleistet werden kann.
- Alle identifizierten Dokumente und Softwareelemente sind unmissverständlich in der Testdokumentation benannt.

Während der Testplanung sollten Konfigurationsmanagementverfahren und Infrastruktur (Werkzeuge) identifiziert und eingeführt werden.

5.5 Risiken und Testen

5.5.1 Definition des Risikos

Ein Risiko ist ein möglicherweise vorkommendes Ereignis in der Zukunft, das negative Auswirkungen hat. Die Höhe des Risikos wird durch die Eintrittswahrscheinlichkeit des Ereignisses und dessen Wirkung (der Schadenhöhe) bestimmt.

5.5.2 Produkt- und Projektrisiken

Produktrisiken beziehen sich auf eine mögliche Nichterfüllung der berechtigten Bedürfnisse seiner Benutzer und/oder Stakeholder durch ein Arbeitsergebnis (z.B. eine Spezifikation, eine Komponente, ein System oder ein Test). Wenn die Produktrisiken mit spezifischen Qualitätsmerkmalen eines Produkts in Verbindung stehen (z.B. funktionale Eignung, Zuverlässigkeit, Performanz, Gebrauchstauglichkeit, IT-Sicherheit, Kompatibilität, Wartbarkeit und Übertragbarkeit), werden Produktrisiken auch Qualitätsrisiken genannt. Beispiele für Produktrisiken sind u.a.:

- Die Software führt vielleicht nicht die gewünschten Funktionen gemäß ihrer Spezifikation aus.
- Die Software führt vielleicht nicht die gewünschten Funktionen gemäß den Bedürfnissen von Benutzern, Kunden und/oder Stakeholdern aus.
- Eine Systemarchitektur könnte einige nicht-funktionale Anforderungen nicht angemessen unterstützen.
- Eine bestimmte Berechnung könnte unter bestimmten Bedingungen nicht korrekt durchgeführt werden.
- Eine Schleifenkontrollstruktur ist möglicherweise nicht korrekt kodiert.

- Die Antwortzeiten könnten für ein hochleistungsfähiges Abwicklungssystem nicht angemessen sein.
- Rückmeldungen zum Benutzererlebnis könnten die Produkterwartungen nicht erfüllen.

Projektrisiken beziehen sich auf Situationen, die, falls sie eintreffen, einen negativen Effekt auf die Fähigkeit des Projekts haben, seine Ziele zu erfüllen. Beispiele für Projektrisiken sind u.a.:

- Projektprobleme:
  - Verzögerungen können in der Lieferung, Aufgabenerfüllung oder Erfüllung von Endekriterien oder der Definition-of-Done auftreten.
  - Ungenaue Schätzungen, Neuverteilung von Mitteln an höher priorisierte Projekte oder generelle Kosteneinsparungen im Unternehmen können zu nicht ausreichender Finanzierung führen.
  - Späte Änderungen können zu erheblichen Überarbeitungen führen.
- Unternehmensprobleme:
  - Fähigkeiten, Schulungen und Mitarbeiter sind möglicherweise nicht ausreichend.
  - Personalprobleme können Konflikte und Probleme auslösen.
  - Benutzer, Mitarbeiter der Fachabteilung oder Fachexperten sind unter Umständen aufgrund von gegenläufigen Geschäftsprioritäten nicht verfügbar.
- Politische Probleme:
  - Tester könnten ihre Bedürfnisse und/oder die Testergebnisse nicht ausreichend kommunizieren.
  - Entwickler und/oder Tester könnten Informationen, die in Tests und Reviews gefunden werden, nicht weiterverfolgen (z.B. verbessern nicht die Entwicklungs- und Testpraktiken).
  - Es kann eine ungeeignete Haltung oder Erwartung gegenüber dem Testen geben (z.B. keine Wertschätzung der Bedeutung der Fehlerfindung während des Testens).
- Technische Probleme:
  - Anforderungen könnten nicht gut genug definiert sein.
  - Die Anforderungen könnten unter den gegebenen Beschränkungen nicht erfüllt werden.
  - Die Testumgebung könnte nicht rechtzeitig verfügbar sein.
  - Datenkonvertierung, Migrationsplanung und ihre Werkzeugunterstützung könnten zu spät kommen.
  - Schwächen im Entwicklungsprozess können Auswirkungen auf die Konsistenz oder die Qualität von Projektergebnissen wie Entwurf, Code, Konfiguration, Testdaten und Testfälle haben.
  - Schlechtes Fehlermanagement und ähnliche Probleme können zu kumulierten Fehlerzuständen und anderen technischen Schulden führen.
- Lieferantenprobleme:
  - Ein Drittanbieter liefert möglicherweise ein notwendiges Produkt oder eine notwendige Dienstleistung nicht oder wird insolvent.
  - Vertragliche Probleme können Probleme für das Projekt auslösen.

Projektrisiken können sowohl Entwicklungsaktivitäten als auch Testaktivitäten beeinflussen. In einigen Fällen sind Projektmanager für den Umgang mit allen Projektrisiken verantwortlich, aber es ist nicht ungewöhnlich, dass Testmanager die Verantwortung für testbezogene Projektrisiken tragen.

5.5.3 Risikobasiertes Testen und Produktqualität

Risiko wird genutzt, um den Aufwand während des Testens zu bündeln. Das heißt, um zu entscheiden, wo und wann mit dem Testen begonnen wird, und um Bereiche zu identifizieren, die mehr Aufmerksamkeit benötigen. Testen wird genutzt, um die Eintrittswahrscheinlichkeit eines unerwünschten Ereignisses oder um die Auswirkungen eines solchen Ereignisses zu reduzieren. Testen wird als Verfahren zur Risikominderung eingesetzt, um Informationen sowohl über identifizierte Risiken als auch über die Restrisiken (nicht gelöste Risiken) zu geben.

Ein risikobasiertes Verfahren des Testens liefert proaktive Möglichkeiten, die Stufen des Produktrisikos zu reduzieren. Es beinhaltet die Produktrisikoanalyse, die die Identifizierung von Produktrisiken und die Beurteilung der Wahrscheinlichkeit und Auswirkung jedes Risikos einbezieht. Die ermittelte Produktrisikoinformation wird genutzt, um die Testplanung, die Spezifikation, Vorbereitung und Durchführung von Testfällen und die Testüberwachung und Steuerung zu lenken. Die frühe Analyse der Produktrisiken trägt zum Erfolg eines Projekts bei.

In einem risikobasierten Verfahren werden die Ergebnisse der Produktrisikoanalyse wie folgt genutzt:

- Bestimmen der Testverfahren, die genutzt werden sollen
- Bestimmen der einzelnen Teststufen und Testarten, die durchzuführen sind (z.B. IT-Sicherheitstests, Testen der Barrierefreiheit)
- Bestimmen des Umfangs von durchzuführenden Tests
- Priorisieren der Tests, um dadurch die kritischen Fehlerzustände so früh wie möglich zu finden
- Bestimmen, ob zusätzlich zum Testen Aktivitäten durchgeführt werden sollen, um das Risiko zu reduzieren (z.B. Schulung von unerfahrenen Designern)

Risikobasiertes Testen greift auf das gemeinsame Wissen und den Einblick der Projekt-Stakeholder zurück, um die Produktrisikoanalyse durchzuführen. Um sicherzustellen, dass die Wahrscheinlichkeit eines Produktfehlers minimiert wird, liefern Risikomanagementaktivitäten einen disziplinierten Ansatz für Folgendes:

- Die Analyse (und regelmäßige Neubewertung) dessen, was falsch laufen kann (Risiken)
- Die Festlegung, welche Risiken unbedingt behandelt werden müssen
- Das Ergreifen von Maßnahmen, um diese Risiken zu reduzieren
- Die Schaffung von Notfallplänen, um mit den Risiken für den Fall des Eintretens umzugehen

Darüber hinaus können Tests neue Risiken identifizieren und dabei helfen, zu entscheiden, welche Risiken eingedämmt werden sollten, sowie die Unsicherheit über Risiken reduzieren.

5.6 Fehlermanagement

Da eines der Ziele des Testens das Finden von Fehlerzuständen ist, sollten Fehlerzustände, die während der Tests gefunden werden, aufgezeichnet werden. Die Art und Weise, in der die Fehlerzustände aufgezeichnet werden, kann variieren, abhängig vom Kontext der zu testenden Komponente oder des zu testenden Systems, der Teststufe und dem Softwareentwicklungslebenszyklus-Modell. Alle identifizierten Fehlerzustände sollten untersucht werden und von der Entdeckung über die Klassifizierung bis hin zur Lösung nachverfolgt werden (z.B. Korrektur der Fehlerzustände und erfolgreiche Fehlernachtests der Lösung, Übertragung auf ein folgendes Release, Akzeptanz als dauerhafte Produkteinschränkung usw.). Um alle Fehlerzustände bis hin zur Lösung zu verwalten, sollte ein Unternehmen einen Fehlermanagementprozess etablieren, der einen Workflow und Regeln für die Klassifizierung enthält. Auf diesen Prozess müssen sich alle am Fehlermanagement beteiligten Parteien einigen u.a. Architekten, Designer, Entwickler, Tester und Product Owner. In einigen Organisationen kann die Aufzeichnung und Nachverfolgung von Fehlerzuständen sehr informell sein.

Während des Fehlermanagementprozesses können einige Berichte falsch positive Ergebnisse enthalten, die keine echten Fehlerwirkungen aufgrund von Fehlerzuständen sind. Zum Beispiel kann ein Test fehlschlagen, wenn eine Netzwerkverbindung unterbrochen ist oder ein Time-out eingetreten ist. Dieses Verhalten resultiert nicht aus einem Fehlerzustand im Testobjekt, sondern ist eine Anomalie, die untersucht werden muss. Tester sollten versuchen, die Anzahl der falsch positiven Ergebnisse, die als Fehlerzustände gemeldet werden, zu reduzieren.

Fehlerzustände können während der Kodierung, der statischen Analyse, der Reviews oder während dynamischer Tests gemeldet werden. Fehlerzustände können für Probleme im Code oder bei funktionierenden Systemen oder für jede Art von Dokumentation einschließlich Anforderungen, User-Stories und Abnahmekriterien, Entwicklungsdokumente, Testdokumente, Benutzerhandbücher oder Installationsanleitungen gemeldet werden. Um einen effektiven und effizienten Fehlermanagementprozess zu gewährleisten, können Organisationen Standards für die Merkmale, Klassifizierungen und den Workflow von Fehlern definieren.

Gängige Fehlerberichte haben die folgenden Ziele:

- Entwicklern und anderen Beteiligten Informationen über jedes aufgetretene unerwünschte Ereignis zur Verfügung zu stellen, um sie in die Lage zu versetzen, spezifische Auswirkungen zu identifizieren, das Problem mit minimalem Test zu reproduzieren und zu isolieren und die möglichen Fehlerzustände wie erforderlich zu korrigieren oder anderweitig das Problem zu lösen
- Testmanagern eine Möglichkeit zu geben, die Qualität der Arbeitsergebnisse und die Auswirkungen auf das Testen nachzuverfolgen (z.B. falls eine große Anzahl von Fehlerzuständen gemeldet wird, haben die Tester mehr Zeit damit verbracht, die Fehlerzustände zu melden, statt Tests durchzuführen, und es werden mehr Fehlernachtests notwendig sein)
- Ideen liefern für die Verbesserung der Entwicklung und des Testprozesses

Ein Fehlerbericht, der während dynamischer Tests erstellt wird, enthält üblicherweise Folgendes:

- Eine Kennung
- Einen Titel und eine kurze Zusammenfassung des gemeldeten Fehlerzustands
- Das Datum des Fehlerberichts, die ausstellende Organisation und den Verfasser
- Die Identifikation des Testelements (des zu testenden Konfigurationselements) und der Testumgebung
- Die Phase des Entwicklungslebenszyklus, in der der Fehlerzustand beobachtet wurde
- Eine Beschreibung des Fehlerzustands, um die Reproduzierbarkeit und Lösung zu ermöglichen, einschließlich Protokollen, Datenbank-Dumps, Bildschirmfotos oder Aufnahmen (falls während der Testdurchführung gefunden)
- Erwartete Ergebnisse und Istergebnisse
- Den Umfang oder Auswirkungsgrad (Fehlerschweregrad) des Fehlerzustands in Bezug auf die Interessen der Stakeholder
- Die Dringlichkeit/Priorität für die Behebung
- Den Status des Fehlerberichts (z.B. offen, aufgeschoben, doppelt, wartet auf Behebung, wartet auf Fehlernachtest, wieder geöffnet, geschlossen)
- Schlussfolgerungen, Empfehlungen und Freigaben
- Allgemeine Probleme, wie andere Bereiche, die durch die Änderung, die sich aus dem Fehlerzustand ergibt, betroffen sind

- Die Änderungshistorie, wie die Reihenfolge der Aktionen, die von Projektmitarbeitern in Bezug auf den Fehlerzustand ausgeführt wurden, um diesen zu isolieren, zu reparieren und als behoben zu bestätigen
- Referenzen, einschließlich des Testfalls, der das Problem aufgedeckt hat

Einige dieser Details können automatisch einbezogen und/oder verwaltet werden, wenn Fehlermanagementwerkzeuge genutzt werden, z.B. automatische Zuweisung einer Kennung, Zuweisung und Aktualisierung des Fehlerberichtsstatus während des Workflows usw. Fehler, die während statischer Tests, insbesondere in Reviews, gefunden werden, werden üblicherweise anders dokumentiert, z.B. in Review-Besprechungsprotokollen.

Ein Beispiel für den Inhalt eines Fehlerberichts ist im ISO-Standard ISO/IEC/IEEE 29119-3 zu finden, der Fehlerberichte als Abweichungsberichte bezeichnet.

6. Werkzeugunterstützung für das Testen – 40 Minuten

Schlüsselbegriffe

Datengetriebenes Testen, schlüsselwortgetriebener Test, Testautomatisierung, Testausführungswerkzeug, Testmanagementwerkzeug

Lernziele für Testwerkzeuge

6.1 Überlegungen zu Testwerkzeugen

FL-6.1.1 (K2) Testwerkzeuge gemäß ihrem Zweck und den Testaktivitäten, die sie unterstützen, klassifizieren können

FL-6.1.2 (K1) Nutzen und Risiken der Testautomatisierung identifizieren können

FL-6.1.3 (K1) Sich an besondere Gesichtspunkte von Testdurchführungs- und Testmanagementwerkzeugen erinnern können

6.2 Effektive Nutzung von Werkzeugen

FL-6.2.1 (K1) Die Hauptprinzipien für die Auswahl eines Werkzeugs identifizieren können

FL-6.2.2 (K1) Sich an Ziele für die Nutzung von Pilotprojekten zur Einführung von Werkzeugen erinnern können

FL-6.2.3 (K1) Erfolgsfaktoren für die Evaluierung, Implementierung, Bereitstellung und kontinuierliche Unterstützung von Testwerkzeugen in einem Unternehmen identifizieren können

6.1 Überlegungen zu Testwerkzeugen

Testwerkzeuge können genutzt werden, um eine oder mehrere Testaktivitäten zu unterstützen. Solche Werkzeuge sind u.a.:

- Werkzeuge, die direkt in Tests genutzt werden, wie Testausführungswerkzeuge und Testdateneditoren und -generatoren
- Werkzeuge, die helfen, die Anforderungen, Testfälle, Testabläufe, automatisierte Testskripte, Testergebnisse, Testdaten und Fehlerzustände zu verwalten, sowie Werkzeuge für die Berichterstattung und Überwachung der Testdurchführung
- Werkzeuge, die zur Analyse und Bewertung verwendet werden
- Jedes Werkzeug, das das Testen unterstützt (in diesem Sinn ist ein Tabellenkalkulationsprogramm auch ein Testwerkzeug)

6.1.1 Klassifizierung von Testwerkzeugen

Testwerkzeuge können, abhängig vom Kontext, einen oder mehrere der folgenden Zwecke erfüllen:

- Die Effizienz der Testaktivitäten durch die Automatisierung sich wiederholender Aufgaben verbessern oder von Aufgaben, die erhebliche Ressourcen verbrauchen, wenn sie manuell durchgeführt werden (z.B. Testdurchführung, Regressionstests)
- Die Effizienz von Testaktivitäten durch die Unterstützung von manuellen Testaktivitäten während des Testprozesses verbessern (siehe Abschnitt *1.4 Testprozess)*
- Verbesserung der Qualität der Testaktivitäten durch konsistenteres Testen und eine höhere Fehlerreproduzierbarkeit
- Aktivitäten automatisieren, die nicht manuell ausgeführt werden können (z.B. groß angelegte Performanztests)
- Die Zuverlässigkeit des Testens erhöhen (z.B. durch automatisierten Vergleich großer Datenmengen oder die Simulation von Verhaltensweisen)

Werkzeuge können auf Basis verschiedener Kriterien wie Zweck, Preis, Lizenzmodell (z.B. kommerzielle Standardsoftware oder Open Source) und genutzter Technologie klassifiziert werden. In diesem Lehrplan sind Werkzeuge in Bezug auf Testaktivitäten klassifiziert, die sie unterstützen.

Einige Werkzeuge unterstützen nur oder hauptsächlich eine Aktivität, andere können mehr als eine Aktivität unterstützen, aber sie sind unter der Aktivität klassifiziert, mit der sie am wahrscheinlichsten in Verbindung gebracht werden. Werkzeuge eines einzigen Anbieters, insbesondere diejenigen, die entworfen wurden, um zusammenzuarbeiten, können als eine integrierte Suite zur Verfügung gestellt werden.

Einige Arten von Testwerkzeugen können intrusiv sein, d.h., sie können das tatsächliche Ergebnis des Tests beeinflussen. Beispielsweise können die tatsächlichen Antwortzeiten einer Anwendung aufgrund der zusätzlichen Anweisungen, die von einem Performanztestwerkzeug ausgeführt werden, differieren oder die erreichte Codeüberdeckung kann durch die Verwendung eines Überdeckungswerkzeugs verzerrt sein. Die Folge des Einsatzes von intrusiven Werkzeugen wird Untersuchungseffekt genannt.

Einige Werkzeuge sind als Unterstützung insbesondere für Entwickler geeignet (z.B. Werkzeuge, die während Komponenten- und Integrationstests genutzt werden). Derartige Werkzeuge sind in den folgenden Abschnitten mit „(E)“ markiert.

Werkzeugunterstützung für das Management des Testens und für Testmittel

Managementwerkzeuge können auf jede Testaktivität über den gesamten Softwareentwicklungslebenszyklus angewendet werden. Beispiele für Werkzeuge, die das Management des Testens und Testmittel unterstützen, sind u.a.:

- Testmanagementwerkzeuge und Application-Lifecycle-Management-Werkzeuge (ALM)
- Anforderungsmanagementwerkzeuge (z.B. Verfolgbarkeit zu Testobjekten)
- Fehlermanagementwerkzeuge
- Konfigurationsmanagementwerkzeuge
- Werkzeuge zur kontinuierlichen Integration (E)

Werkzeugunterstützung für statische Tests

Statische Testwerkzeuge stehen in Verbindung mit den Aktivitäten und Nutzen, die in Kapitel 3 beschrieben wurden. Beispiele für ein solches Werkzeug sind u.a.:

- Statische Analysewerkzeuge (E)

Werkzeugunterstützung für Testentwurf und -realisierung

Testentwurfswerkzeuge helfen beim Erstellen von wartbaren Arbeitsergebnissen im Testentwurf und der Testrealisierung, u.a. beim Erstellen von Testfällen, Testabläufen und Testdaten. Beispiele für solche Werkzeuge sind u.a.:

- Testwerkzeuge für das modellbasierte Testen
- Testdateneditoren und -generatoren

In einigen Fällen können Werkzeuge, die Testentwurf und -realisierung unterstützen, auch die Testdurchführung und -aufzeichnung unterstützen oder ihre Ausgaben direkt an andere Werkzeuge übertragen, die bei der Testdurchführung und -aufzeichnung helfen.

Werkzeugunterstützung für Testdurchführung und -protokollierung

Viele Werkzeuge existieren, um die Testdurchführung und -protokollierung zu unterstützen und zu verbessern. Beispiele für diese Werkzeuge sind u.a.:

- Testausführungswerkzeuge (z.B. um Regressionstests durchzuführen)
- Überdeckungswerkzeuge (z.B. Anforderungsüberdeckung, Codeüberdeckung (E))
- Testrahmen (E)

Werkzeugunterstützung zur Performanzmessung und dynamischen Analyse

Werkzeuge zur Performanzmessung und dynamischen Analyse sind wesentlich für die Unterstützung von Performanz- und Lasttestaktivitäten, da diese Aktivitäten nicht effektiv manuell durchführbar sind. Beispiele für solche Werkzeuge sind u.a.:

- Performanztestwerkzeuge
- Dynamische Analysewerkzeuge (E)

Werkzeugunterstützung für spezielle Testbedürfnisse

Zusätzlich zu Werkzeugen, die den allgemeinen Testprozess unterstützen, gibt es viele andere Werkzeuge, die spezifische Aspekte des Testens nichtfunktionaler Merkmale unterstützen.

6.1.2 Nutzen und Risiken der Testautomatisierung

Einfach ein Werkzeug zu kaufen, garantiert noch keinen Erfolg. Jedes neu eingeführte Werkzeug in einem Unternehmen erfordert Aufwand, um einen echten und anhaltenden Nutzen zu erzielen. Durch die Nutzung von Testwerkzeugen entstehen potenziell Nutzen und Chancen, aber auch Risiken. Dies gilt insbesondere für Testausführungswerkzeuge (was häufig auch als Testautomatisierung bezeichnet wird).

Potenzieller Nutzen durch die Verwendung von Werkzeugen für die Unterstützung der Testdurchführung beinhaltet u.a.:

- Reduktion von sich wiederholender manueller Arbeit (z.B. Durchführung von Regressionstests, Aufsetzen oder Abbau von Umgebungen, wiederholte Eingabe der gleichen Testdaten und Prüfung gegen Programmierrichtlinien) und dadurch Zeiteinsparung
- Höhere Konsistenz und Wiederholbarkeit (z.B. werden Testdaten in kohärenter Art und Weise erstellt, Tests werden durch ein Werkzeug in der gleichen Reihenfolge und der gleichen Frequenz durchgeführt und Tests werden konsistent aus Anforderungen abgeleitet)
- Objektivere Beurteilung (z.B. statische Maße, Überdeckung)
- Vereinfachter Zugang zu Informationen über das Testen (z.B. Statistiken und Grafiken über den Testfortschritt, Fehlerraten und Performanz)

Mögliche Risiken der Nutzung von Werkzeugen zur Testunterstützung sind u.a.:

- Erwartungen an das Werkzeug können unrealistisch sein (u.a. bezüglich Funktionalität oder Leichtigkeit der Nutzung).
- Zeit, Kosten und der Aufwand für die Einführung des Werkzeugs können unterschätzt werden (u.a. Schulung und externe Fachkenntnis).
- Zeit und Aufwand, um wesentliche und andauernde Nutzen aus dem Werkzeug zu ziehen, können unterschätzt werden (u.a. die Notwendigkeit für Änderungen im Testprozess und kontinuierliche Verbesserungen in der Art und Weise, wie das Werkzeug genutzt wird).
- Der notwendige Aufwand für die Wartung der Testarbeitsergebnisse, der durch das Werkzeug erstellt wurde, wird unterschätzt.
- Man verlässt sich zu stark auf das Werkzeug (es wird als Ersatz für den Testentwurf oder die Testdurchführung angesehen oder Testautomatisierung wird genutzt, obwohl manuelle Tests besser geeignet wären).
- Die Versionskontrolle der Testarbeitsergebnisse wird vernachlässigt.
- Beziehungen und Interoperabilitätsprobleme zwischen kritischen Werkzeugen werden vernachlässigt, wie Anforderungsmanagementwerkzeugen, Konfigurationsmanagementwerkzeugen, Fehlermanagementwerkzeugen und Werkzeugen von verschiedenen Anbietern.
- Der Werkzeuganbieter kann seine Geschäftstätigkeit einstellen, das Werkzeug vom Markt nehmen oder das Werkzeug an einen anderen Werkzeuganbieter verkaufen.
- Der Anbieter kann auf Unterstützungsanfragen, Aktualisierungen oder Fehlerbehebungen mangelhaft reagieren.
- Ein Open-Source-Projekt kann ausgesetzt werden.
- Eine neue Plattform oder Technologie kann durch das Werkzeug nicht mehr unterstützt werden.
- Es könnte keine klaren Eigentumsverhältnisse an dem Werkzeug geben (z.B. für Betreuung, Aktualisierungen usw.).

6.1.3 Besondere Betrachtung der Testausführungs- und Testmanagementwerkzeuge

Für eine reibungslose und erfolgreiche Realisierung gibt es eine Reihe von Aspekten, die bei der Auswahl und Integration von Testausführungs- und Testmanagementwerkzeugen in einem Unternehmen berücksichtigt werden sollten.

Testausführungswerkzeuge

Testausführungswerkzeuge führen Testobjekte aus, indem sie automatisierte Skripte nutzen. Testwerkzeuge dieser Art erfordern oft einen hohen Aufwand, um signifikante Nutzen zu erzielen. Es gibt verschiedene Ansätze zur automatisierten Testausführung:

- **Mitschnitt**: Die Erfassung von Tests durch die Aufnahme der Aktionen eines manuellen Testers mag sinnvoll erscheinen, aber dieses Verfahren eignet sich nicht für eine große Anzahl von Testskripten. Ein aufgezeichnetes Skript ist eine lineare Darstellung mit spezifischen Daten und Aktionen als Teil jedes Skripts. Diese Art Skript kann instabil werden, wenn unerwartete Ereignisse eintreten und erfordert eine stetige Wartung, da sich die Benutzungsschnittstelle des Systems über die Zeit hinweg weiterentwickelt.
- **Datengetrieben**: Dieser Testansatz legt die Testeingaben und erwartete Ergebnisse üblicherweise in einer Tabelle getrennt ab und nutzt ein generischeres Testskript, das die Eingabedaten auslesen und den gleichen Test mit verschiedenen Daten durchführen kann.
- **Schlüsselwortgetrieben**: Dieser Testansatz verarbeitet ein generisches Skript Schlüsselworte (auch Aktionsworte genannt), welche die Aktionen beschreiben, die durchzuführen sind. Diese rufen dann Schlüsselwortskripte auf, um die zugehörigen Testdaten zu verarbeiten.

Die oben genannten Ansätze erfordern jemanden, der in der Skriptsprache Erfahrung hat (Tester, Entwickler oder Spezialisten in Testautomatisierung). Bei der Verwendung von daten- oder schlüsselwortgetriebenen Testansätzen können auch Tester, die mit der Skriptsprache nicht vertraut sind, etwas dazu beitragen, indem sie Testdaten und/oder Schlüsselwörter für diese vordefinierten Skripte erstellen. Unabhängig vom genutzten Skriptverfahren müssen die erwarteten Ergebnisse für jeden Test mit den Istergebnissen des Tests entweder dynamisch verglichen werden (wenn der Test läuft) oder für spätere Vergleiche (nach Durchführung) gespeichert werden.

Weitere Details und Beispiele für datengetriebene und schlüsselwortgetriebene Testansätze werden im ISTQB-TAE sowie in Fewster 1999 und Buwalda 2001 gegeben.

Werkzeuge für das modellbasierte Testen (MBT) ermöglichen die Erfassung einer funktionalen Spezifikation in Form eines Modells wie zum Beispiel eines Aktivitätsdiagramms. Diese Aufgabe wird üblicherweise von einem Systemdesigner ausgeführt. Das MBT-Werkzeug interpretiert das Modell, um Testfallspezifikationen zu erstellen, die dann in einem Testmanagementwerkzeug gespeichert und/oder von einem Testausführungswerkzeug ausgeführt werden können (siehe ISTQB-MBT).

Testmanagementwerkzeuge

Testmanagementwerkzeuge benötigen aus unterschiedlichen Gründen Schnittstellen mit anderen Werkzeugen oder Tabellenkalkulationsprogrammen, z.B.:

- Um nützliche Informationen in einem Format bereitzustellen, das den Bedürfnissen der Organisation entspricht
- Um konsistente Verfolgbarkeit zu Anforderungen in einem Anforderungsmanagementwerkzeug zu erhalten
- Um eine Verbindung der Versionsinformation des Testobjekts zum Konfigurationsmanagementwerkzeug herzustellen

Dies ist insbesondere zu beachten, wenn ein integriertes Werkzeug genutzt wird (z.B. Application Lifecycle Management), das ein Testmanagementmodul und auch andere Module (z.B. Projektplanungs- und Budgetinformationen) enthält, die von unterschiedlichen Gruppen innerhalb des Unternehmens genutzt werden.

6.2 Effektive Nutzung von Werkzeugen

6.2.1 Hauptgrundsätze für die Auswahl von Werkzeugen

Grundsätzliche Überlegungen bei der Auswahl eines Werkzeugs für ein Unternehmen sind u.a.:

- Bewertung der Reife des eigenen Unternehmens, Analyse der Stärken und Schwächen
- Identifizierung von Möglichkeiten für die Verbesserung des Testprozesses, unterstützt durch Werkzeuge
- Verständnis der Technologien, die von den Testobjekten genutzt werden, um ein Werkzeug auszuwählen, das mit dieser Technologie kompatibel ist
- Verständnis der Werkzeuge für Build und kontinuierliche Integration, die im Unternehmen bereits im Einsatz sind, um Werkzeugkompatibilität und Integration zu gewährleisten
- Bewertung des Werkzeugs gegen klar spezifizierte Anforderungen und objektive Kriterien (für Nutzen und Anwendung)
- Überlegung, ob das Werkzeug für eine kostenfreie Testperiode verfügbar ist (und wie lange)
- Bewertung des Anbieters (einschließlich Trainings-, Unterstützungs- und kommerzieller Aspekte) oder der Unterstützung für nichtkommerzielle Werkzeuge (z.B. Open Source)
- Identifizierung interner Anforderungen für Coaching und Anleitung zur Nutzung des Werkzeugs
- Bewertung des Schulungsbedarfs unter Berücksichtigung der Test- (und Testautomatisierungs-) Kenntnisse derjenigen, die direkt mit dem Werkzeug arbeiten werden
- Überlegung zu Vor- und Nachteilen verschiedener Lizenzmodelle (z.B. kommerzieller Standardsoftware oder Open Source)
- Schätzung des Kosten-Nutzen-Verhältnisses auf Basis eines konkreten Business Case (falls benötigt)

Als letzter Schritt sollte eine Machbarkeitsstudie (Proof-of-Concept) durchgeführt werden, um zu beurteilen, ob das Werkzeug effektiv mit der zu testenden Software und innerhalb der aktuellen Infrastruktur arbeitet oder wenn nötig, um Änderungen zu identifizieren, die für diese Infrastruktur notwendig sind, um das Werkzeug effektiv nutzen zu können.

6.2.2 Pilotprojekte für die Einführung eines Werkzeugs in einem Unternehmen

Nach Abschluss der Auswahl eines Werkzeugs und der erfolgreichen Machbarkeitsstudie beginnt dessen Einführung in einem Unternehmen in der Regel mit einem Pilotprojekt, das die folgenden Ziele hat:

- Tiefgehende Kenntnis über das Werkzeug und Verständnis seiner Stärken und Schwächen erzielen
- Evaluierung, wie das Werkzeug in bestehende Prozesse und Methoden passt, und bestimmen, was ggf. geändert werden müsste
- Über die Standardisierung des Werkzeugeinsatzes hinsichtlich Nutzung, Verwaltung, Speicherung und Wartung des Werkzeugs und der Testarbeitsergebnisse (z.B. Entscheidung über Namens-

konventionen für Dateien und Tests, Auswahl von Programmierrichtlinien, Neuanlage von Bibliotheken und die Festlegung von modularen Testsuiten) entscheiden

- Beurteilen, ob der Nutzen mit vertretbaren Kosten erreicht werden kann
- Die Metriken, die das Werkzeug sammeln und aufzeichnen soll, verstehen und das Werkzeug konfigurieren, um sicherzustellen, dass diese Metriken erfasst und aufgezeichnet werden können

6.2.3 Erfolgsfaktoren für Werkzeuge

Erfolgsfaktoren für die Evaluierung, Implementierung, Bereitstellung und die andauernde Unterstützung von Werkzeugen innerhalb eines Unternehmens sind u.a.:

- Schrittweise Einführung des Werkzeugs im ganzen Unternehmen
- Anpassung und Verbesserung von Prozessen, so dass sie zur Nutzung des Werkzeugs passen
- Anbieten von Schulungsmaßnahmen, Coaching und Anleitung der Benutzer des Werkzeugs
- Definition von Richtlinien für die Nutzung des Werkzeugs (z.B. interne Standards für die Automatisierung)
- Einführung eines Verfahrens zur Sammlung von Nutzungsdaten über den derzeitigen Gebrauch
- Überwachung der Nutzung und des Nutzens des Werkzeugs
- Unterstützung für die Benutzer eines eingeführten Werkzeugs bieten
- Sammeln von gewonnenen Erkenntnissen von allen Benutzern

Außerdem ist es wichtig, sicherzustellen, dass das Werkzeug technisch und organisatorisch in den Softwareentwicklungslebenszyklus integriert ist, was unterschiedliche Organisationen betreffen kann, die für den Betrieb und/oder Drittanbieter verantwortlich sind.

Siehe Graham 2012 für Erfahrungen und Rat zur Nutzung von Testausführungswerkzeugen.

7. Referenzen

7.1 Normen/Standards

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering - Software testing - Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering - Software testing - Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering - Software testing - Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246: (2017) Software and systems engineering — Work product reviews

UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

7.2 ISTQB-Dokumente

ISTQB® -Glossar

ISTQB® Foundation Level - Überblick 2018

ISTQB-AT Foundation Level Syllabus Agile Tester

ISTQB-ATA Advanced Level Syllabus Test Analyst

ISTQB-TTA Advanced Level Syllabus Technical Test Analyst

ISTQB-ATM Advanced Level Syllabus Testmanager

ISTQB-ETM Expert Level Test Management Syllabus (englischsprachig)

ISTQB-EITP Expert Level Improving the Test Process Syllabus (englischsprachig)

Spezialistenmodule:

ISTQB-SEC Advanced Level Syllabus Security Tester (englischsprachig)

ISTQB-MBT Foundation Level Certified Model-Based Tester Syllabus (englischsprachig)

ISTQB-TAE Advanced Level Syllabus Test Automation Engineer (englischsprachig)

7.3 Bücher und Artikel

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

Black, R. (2009): Managing the Testing Process (3e), John Wiley & Sons, New York NY

Black, R. (2017): Agile Testing Foundations, BCS Learning & Development Ltd, Swindon UK

Buwalda, H. et al. (2001): Integrated Test Design and Automation, Addison Wesley, Reading MA

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

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

Crispin, L.; Gregory, J. (2008): Agile Testing, Pearson Education, Boston MA

Fewster, M.; Graham, D. (1999): Software Test Automation, Addison Wesley, Harlow UK

Gilb, T.; Graham, D. (1993): Software Inspection, Addison Wesley, Reading MA

Graham, D.; Fewster, M. (2012): Experiences of Test Automation, Pearson Education, Boston MA

Gregory, J.; Crispin, L. (2015): More Agile Testing, Pearson Education, Boston MA

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

Kaner, C.; Bach, J.; Pettichord, B. (2002): Lessons Learned in Software Testing, John Wiley & Sons, New York NY

Kaner, C.; Padmanabhan, S.; Hoffman, D. (2013): The Domain Testing Workbook, Context-Driven Press, New York NY

Kramer, A.; Legeard, B. (2016): Model-Based Testing Essentials: Guide to the ISTQB Certified Model- Based Tester: Foundation Level, John Wiley & Sons, New York NY

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

Sauer, C. (2000): "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research," IEEE Transactions on Software Engineering, Volume 26, Issue 1, pp 1-14

Shull, F.; Rus, I.; Basili, V. (2000): "How Perspective-Based Reading can Improve Requirement Inspections." IEEE Computer, Volume 33, Issue 7, pp 73-79

van Veenendaal, E. (ed.) (2004): The Testing Practitioner (Chapters 8 - 10), UTN Publishers, The Netherlands

Weinberg, G. (2008): Perfect Software and Other Illusions about Testing, Dorset House, New York NY

Wiegers, K. (2002): Peer Reviews in Software, Pearson Education, Boston MA

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

Bucsics, Th.; Baumgartner, M.; Seidl, R.; Gwihs, St. (2015): Basiswissen Testautomatisierung; Konzepte, Methoden und Techniken, 2., aktualisierte und überarbeitete Auflage, 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. (2016): Testen in Scrum-Projekten Leitfaden für Softwarequalität in der agilen Welt. Aus- und Weiterbildung zum ISTQB® Certified Agile Tester – Foundation Extension, 2., aktualisierte und überarbeitete Auflage, dpunkt.verlag, Heidelberg

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. (2019): Basiswissen Softwaretest:, Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB®-Standard, 6., überarbeitete u. aktualisierte Auflage, dpunkt.verlag, Heidelberg

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

7.5 Andere Quellen (in diesem Lehrplan nicht direkt referenziert)

Black, R.; van Veenendaal, E.; Graham, D. (2019): Foundations of Software Testing: ISTQB Certification (4e), Cengage Learning, London UK

Hetzel, W. (1993): Complete Guide to Software Testing (2e), QED Information Sciences, Wellesley MA

8. Anhang A – Hintergrundinformation zum Lehrplan

Zur Geschichte dieses Dokuments

Dieses Dokument stellt den Lehrplan für das internationale Basiszertifikat für Softwaretesten dar, die erste Ausbildungs- und Qualifizierungsstufe des ISTQB® (www.istqb.org).

Dieses Dokument wurde von Juni bis August 2019 von einer Arbeitsgruppe erstellt, die sich aus Mitgliedern des International Software Testing Qualifications Board (ISTQB) zusammensetzt. Updates wurden auf Basis von Review-Ergebnissen von Mitglieds-Boards, die den Foundation Syllabus 2018 verwendet haben, hinzugefügt.

Der vorherige Foundation Lehrplan 2018 wurde in den Jahren 2014 bis 2018 durch eine Arbeitsgruppe, deren Mitglieder durch das International Software Testing Qualifications Board (ISTQB®) benannt wurden, erstellt. Die Version 2018 wurde zunächst durch Vertreter aller ISTQB-Mitglieds-Boards und anschließend durch Repräsentanten der internationalen Softwaretestwelt geprüft.

Ziele der Basiszertifikat-Qualifizierung

- Anerkennung des Testens als einer essenziellen und professionellen Software-Engineering-Disziplin
- Darstellung eines Standardrahmens für die Laufbahn von Softwaretestern
- Verbesserung der Anerkennung von professionellen und qualifizierten Testern sowie deren Stellenprofil durch Arbeitgeber, Kunden, Berufskollegen
- Förderung konsistenter und praxisgerechter Vorgehensweisen in allen Software-Engineering-Disziplinen
- Erkennen von Themen des Testens, die für die Industrie relevant und wertvoll sind
- Softwarelieferanten zu befähigen, zertifizierte Tester anzustellen und mit dem Darstellen dieser Beschäftigungspolitik einen Wettbewerbsvorteil zu erzielen
- Schaffen einer Möglichkeit für Tester und am Testen Interessierte, eine international anerkannte Qualifizierung zu erlangen

Zielsetzungen einer internationalen Qualifizierung

- Wissen im Bereich Softwaretesten länderübergreifend vergleichen zu können
- Tester zu befähigen, über die Landesgrenzen hinaus einfacher Arbeit zu finden
- Durch ein gemeinsames Verständnis des Testens internationale Projekte zu unterstützen bzw. zu ermöglichen
- Die Anzahl qualifizierter Tester weltweit zu erhöhen
- Durch ein internationales Vorgehen mehr Einfluss/Bedeutung zu erlangen als durch ein nationales Vorgehen
- Ein gemeinsames internationales Verständnis und Wissen über das Testen durch einen Lehrplan und ein Glossar zu entwickeln und so das Wissen über das Testen bei allen Beteiligten zu erhöhen
- Testen als Beruf in weiteren Ländern zu verbreiten
- Testern zu ermöglichen, dass sie eine international anerkannte Qualifikation in ihrer Muttersprache erlangen

- Wissens- und Ressourcenaustausch über Ländergrenzen hinweg zu ermöglichen
- Testern und diesem Ausbildungsgang internationale Anerkennung durch die Beteiligung möglichst vieler Länder zu verschaffen

Voraussetzungen für die Basisstufe

Voraussetzung für die Prüfung zum Basiszertifikat ISTQB® Certified Tester ist das Interesse der Kandidaten am Softwaretesten. Es wird den Kandidaten jedoch dringend empfohlen,

- zumindest ein minimales Hintergrundwissen im Bereich Softwareentwicklung oder Softwaretesten zu haben (zum Beispiel sechs Monate Erfahrung als System- oder Abnahmetester oder als Entwickler)
- oder einen Kurs besucht zu haben, der durch ein ISTQB®-Mitglieds-Board nach ISTQB-Standard akkreditiert ist.

Hintergrund und Geschichte des Basiszertifikats „Softwaretesten“

Die unabhängige Zertifizierung von Softwaretestern begann in Großbritannien mit dem Information Systems Examination Board (ISEB) der British Computer Society, als 1998 ein Softwaretestgremium (www.bcs.org.uk/iseb) eingerichtet wurde. Im Jahre 2002 begann der Arbeitskreis Software-Qualität und -Fortbildung (ASQF) in Deutschland mit einer deutschen Softwaretesterausbildung (www.asqf.de). Dieser Lehrplan basiert auf der Grundlage der beiden Lehrpläne von ISEB und ASQF; er schließt neu strukturierte, aktualisierte und zusätzliche Themen mit ein, ausgerichtet auf die möglichst praktische Hilfe für den Tester.

Eine bereits bestehende Basiszertifizierung im Bereich Softwaretesten (von ISEB, ASQF oder einem anderen ISTQB®-Mitglieds-Board), die vor der Einführung der internationalen Zertifizierung erreicht wurde, wird als dem internationalen Zertifikat gleichwertig anerkannt. Das Basiszertifikat besitzt kein Ablaufdatum und muss nicht erneuert werden. Auf dem Zertifikat findet man das Datum, an dem es vergeben wurde.

In jedem teilnehmenden Land werden die lokalen Aspekte durch das jeweilige vom ISTQB® anerkannte nationale oder regionale Software Testing Board kontrolliert. Die Pflichten der Mitglieds-Boards sind durch das ISTQB® spezifiziert, müssen jedoch durch die einzelnen Mitglieds-Boards selbst umgesetzt werden. Dazu gehören auch die Akkreditierung von Schulungsanbietern und die Durchführung der Prüfungen.

9. Anhang B – Lernziele/Kognitive Stufen des Wissens

Die folgende Taxonomie für Lernziele bildet die Grundlage des Lehrplans. Jeder Inhalt wird entsprechend den zugeordneten Lernzielen geprüft.

Taxonomiestufe 1: Kennen (K1)

Der Lernende kann einen Begriff oder ein Konzept erkennen, erinnern und abrufen.

Schlüsselworte: identifizieren, erinnern, abfragen, abrufen, erkennen, kennen

Beispiele:

Erkennt die Definition von Fehlerwirkung (engl. „Failure“) als

- „Nicht-Erfüllen einer definierten Leistung gegenüber einem Anwender oder sonstigen Stakeholder“ oder
- „die Abweichung einer Komponente oder eines Systems von der erwarteten bzw. vereinbarten Lieferung, einer Dienstleistung oder einem Ergebnis.“

Taxonomiestufe 2: Verstehen (K2)

Der Lernende begründet oder erläutert Aussagen zum Thema. Typische beobachtbare Leistungen sind beschreiben, zusammenfassen, vergleichen, klassifizieren, begründen, erklären, Beispiele für Testkonzepte nennen.

Schlüsselworte: zusammenfassen, verallgemeinern, abstrahieren, klassifizieren, vergleichen, auf etwas übertragen, etwas gegenüberstellen, erläutern, interpretieren, übersetzen, darstellen, rückschließen, folgern, kategorisieren, Modelle konstruieren

Beispiele:

Kann erklären, warum Testanalyse und -entwurf so früh wie möglich erfolgen sollen:

- Um Fehlerzustände zu finden, da die Behebung billiger ist
- Um die wichtigsten Fehlerzustände zuerst zu finden

Beschreibt Gemeinsamkeiten und Unterschiede zwischen Integration- und Systemtest:

- Gemeinsamkeiten: Testobjekte für beide, Integrationstest und Systemtest, beinhalten mehr als eine Komponente und beide, Integrationstest und Systemtest, können nicht-funktionale Aspekte umfassen.
- Unterschiede: Der Integrationstest konzentriert sich auf Schnittstellen zwischen und Interaktion von Komponenten; der Systemtest ist auf Aspekte des ganzen Systems und End-to-End-Verarbeitung ausgerichtet.

Taxonomiestufe 3: Anwenden (K3)

Der Lernende überträgt erworbenes Wissen auf gegebene neue Situationen oder wendet es zur Problemlösung an.

Schlüsselworte: anwenden, einsetzen, nutzen, einem Verfahren folgen, ein Verfahren anwenden

Beispiele:

- Identifiziert Grenzwerte für gültige bzw. ungültige Äquivalenzklassen
- Selektiert aus einem Zustandsdiagramm die notwendigen Testfälle zur Überdeckung aller Statusübergänge

Referenz (Für die kognitiven Stufen der Lernziele)

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

10. Anhang C – Veröffentlichungshinweise

Der ISTQB-Foundation-Syllabus 2018 3.1 ist ein Minor Update des Release 2018. Es wurde eine separate Release-Note 2019 mit einer Übersicht pro Kapitel erstellt. Zusätzlich wurde eine Version des Foundation-Lehrplans 2018 mit Änderungsmodus veröffentlicht.

Der ISTQB-Foundation-Lehrplan 2018 ist eine umfassende Aktualisierung und Neufassung der Veröffentlichung aus dem Jahr 2011. Aus diesem Grund gibt es keine detaillierten Veröffentlichungshinweise pro Kapitel und Abschnitt. Allerdings wird an dieser Stelle eine Zusammenfassung der wesentlichen Änderungen gegeben. Darüber hinaus liefert das ISTQB® in einer separaten Release-Note die Verfolgbarkeit zwischen den Lernzielen des Foundation-Level-Lehrplans aus dem Jahr 2011 und der Version aus dem Jahr 2018 und zeigt darin, welche hinzugefügt, aktualisiert oder entfernt wurden.

Zu Beginn des Jahres 2017 hatten über 550.000 Menschen in über 100 Ländern die Basisprüfung abgelegt und es gibt weltweit über 500.000 zertifizierte Tester. In der Erwartung, dass sie alle den Foundation-Lehrplan gelesen haben, um die Prüfung zu bestehen, lässt sich sagen, dass der Foundation-Lehrplan wahrscheinlich das meistgelesene Softwaretestdokument ist, das es je gegeben hat!

Diese groß angelegte Aktualisierung wurde verfasst, um dieses Erbe zu würdigen und den Wert, den das ISTQB® den nächsten 500.000 Menschen der globalen Testcommunity liefert, noch zu verbessern.

In dieser Version wurden alle Lernziele angepasst, um sie atomar zu machen und eine Eins-zu-eins-Verfolgbarkeit vom Lernziel zu den Inhaltsabschnitten (und Prüfungsfragen) zu gewährleisten, die mit dem Lernziel in Verbindung stehen, und um umgekehrt auch eine klare Verfolgbarkeit von den Inhaltsabschnitten (und Prüfungsfragen) zu den zugehörigen Lernzielen zu gewährleisten. Darüber hinaus sind die Zeitaufwände für die Kapitel gegenüber der Version aus dem Jahr 2011 realistischer gestaltet worden, indem nachgewiesene Heuristiken und Formeln aus anderen ISTQB-Lehrplänen genutzt wurden, die auf einer Analyse der Lernziele, die in jedem Kapitel erreicht werden sollen, basieren.

Obwohl dieser Lehrplan die besten Vorgehensweisen und Verfahren wiedergibt, die die Zeit überdauert haben, wurden Änderungen vorgenommen, um die Darstellung des Materials zu modernisieren, insbesondere im Hinblick auf Softwareentwicklungsmethoden (z.B. Scrum und kontinuierliche Bereitstellung) und Technologien (z.B. das Internet der Dinge). Die referenzierten Standards wurden wie folgt aktualisiert, um sie auf den neuesten Stand zu bringen:

1. ISO/IEC/IEEE 29119 ersetzt IEEE-Standard 829.
2. ISO/IEC 25010 ersetzt ISO 9126.
3. ISO/IEC 20246 ersetzt IEEE 1028.

Darüber hinaus ist das ISTQB-Portfolio über die letzten 10 Jahre hinweg enorm gewachsen und daher wurden, wo nötig, weitreichende Querverweise zu verwandtem Material in anderen ISTQB-Lehrplänen hinzugefügt und es wurde eine genaue Prüfung auf Angleichung mit allen Lehrplänen und dem ISTQB-Glossar vorgenommen. Ziel ist es, die Lesbarkeit, Verständlichkeit, Erlernbarkeit und Übersetzbarkeit dieser Version zu erhöhen und auf steigende, praktische Nutzbarkeit und das Gleichgewicht zwischen Wissen und Fähigkeiten zu fokussieren.

Für eine detaillierte Analyse der Änderungen, die im Foundation-Lehrplan 2018 3.1 gemacht wurden, siehe ISTQB-Certified-Tester-Foundation-Level Änderungshinweise 2018 3.1.

11. Anhang D – Übersetzung von Fachbegriffen

Englisch	Deutsch
application lifecycle management	Application Lifecycle Management (ALM)
backup and restore procedures	Sicherungs- und Wiederherstellungsverfahren
burndown charts	Burndown-Charts
business analyst	Businessanalyst
business application	Fachanwendung
business continuity	Business-Continuity (Betriebskontinuität)
business owner	Fachverantwortlicher
business process	Geschäftsprozess
business requirements	Fachanforderungen
business user	Fachanwender
confirmation bias	Bestätigungsfehler
continuous delivery	kontinuierliche Auslieferung (continuous delivery)
daily stand-up meeting	tägliches Stand-up-Meeting
data encoding	Datenverschlüsselung
definition of done	Definition-of-Done
definition of ready	Definition-of-Ready
deployment	Bereitstellung
Design to design	Entwurf ggf. kontextspezifisch abweichend z.B. Architekturdesign entwerfen
designer	Designer
detailed design	Feinentwurf
disaster recovery procedures	Disaster-Recovery-Verfahren
domain knowledge	Fachkenntnisse
evaluate	evaluieren oder bewerten/beurteilen (kontextabhängig)
Hot Sites	Hot Sites
increment	Inkrement
item	Element
Key performance indicator	Key-Performance-Indicator (KPI)
microservices	Microservices
organization	Organisation oder Unternehmen (kontextabhängig)
product domain	Produkteinsatzbereich

Englisch	Deutsch
refactoring	Restrukturierung (Refactoring)
refinement	Verfeinerung (Refinement)
risk level	Risikostufe
service virtualization	Service-Virtualisierung
smart image capturing technology	"Smart"-Imaging-Technologie
team’s velocity	Velocity (Maß für Produktivität im agilen Projekt) des Teams
user	Benutzer
user base	Benutzergruppe
Wideband Delphi estimation technique	Breitband-Delphi-Schätzverfahren

12. Index

Abnahmekriterien 24, 43, 45, 52, 75, 78, 80, 81, 86
Abnahmetest 16, 32, 41, 48, 49, 73
  Benutzer- 32, 42
  betrieblicher 32, 42
  regulatorischer 32, 43
  vertraglicher 32, 43
abstrakter Testfall 24, 28
Abweichungsbericht *Siehe* Fehlerbericht
Ad-hoc-Review 51
agile Softwareentwicklung 16, 22, 24, 26, 29, 34, 36, 37, 47, 52, 57, 73, 75, 78, 80, 82
Aktionsworte *Siehe* schlüsselwortgetriebenes Testverfahren
Alpha-Test 32, 42, 43
änderungsbezogenes Testen 32, 47, 48
Anomalie 26, 86
Anweisungstest 68
Anweisungsüberdeckung 62, 68
anwendungsfallbasierter Test 67
Anwendungsfallbasierter Test 62
Application-Lifecycle-Management-Werkzeuge 90
Äquivalenzklasse 64, 65
Äquivalenzklassenbildung 62, 64
Auswirkungsanalyse 29, 32, 50
Bankanwendungsbeispiele 18, 47
Barrierefreiheit
  Test der 48
Befund 17, 31, 55, 59
Benutzerabnahmetest 32, 35, 42
Beta-Test 32, 42, 43
Big Bang 40
Black-Box-Testverfahren 24, 45, 46, 62, 63, 64, 69
Buddy-Check 57
checklistenbasiertes Review 51
checklistenbasiertes Testen 62
Codeüberdeckung 16, 46, 90
compliance *Siehe* Konformität
datengetriebenes Testen 88
datengetriebenes Testverfahren 92
Debugging 13, 16, 17
Definition-of-Done 22, 27, 74, 77, 78, 80, 82, 84, 103
Definition-of-Ready 77, 78, 103
Disaster-Recovery-Verfahren 42, 44, 103
Dry Run 58, 59
dynamischer Test 15, 19, 30, 51, 52, 53
Eingangskriterien 71, 78
embedded systems 67
Endekriterien 22, 27, 54, 55, 58, 60, 71, 74, 77, 78, 80, 82, 84
Entscheidungstabellentest 62, 66
Entscheidungstest 68
Entscheidungsüberdeckung 48, 62, 68
Entwickler 17, 30, 31, 34, 37, 43, 64, 86
Entwicklungsmodell
  abnahmetestgetriebene Entwicklung 24
  agiles 35
  inkrementell 33, 34, 35
  iterativ 33, 34, 76
  sequenzielles 32, 33, 76
  testgetriebenes 37
  verhaltensgetriebene Entwicklung 24
  V-Modell 33
erfahrungsbasierte Testverfahren 24, 62, 64, 69
  checklistenbasiertes Testen *Siehe* checklistenbasiertes Testen
  exploratives Testen *Siehe* exploratives Testen

intuitive Testfallermittlung *Siehe* intuitive Testfallermittlung
erfahrungsbasierter Test 25
erwartetes Ergebnis 26, 92
exploratives Testen 25, 28, 29, 62, 69
Facilitator 56
falsch negativ 18, 41
falsch positiv 18, 41, 86
Fehlerart 23, 53, 57, 59
Fehlerbericht 27, 29, 55, 72, 86, 87
Fehlermanagement 37, 71, 72, 84, 85, 86
Fehlermanagementwerkzeug 27, 87, 90
Fehlernachtest 16, 26, 32, 45, 47, 53, 79, 81, 85, 86
Fehlerschweregrad 86
Fehlerwirkung 13, 15, 16, 17, 18, 19, 21, 26, 30, 32, 36, 37, 39, 41, 44, 53, 65, 69, 73, 77, 86
Abnahmetest 44
Integrationstest 39
Komponententest 37
Systemintegrationstest 39
Systemtest 41
Fehlerzustand 13, 15, 16, 17, 18, 19, 20, 24, 26, 27, 29, 30, 32, 36, 37, 38, 39, 40, 41, 42, 44, 45, 47, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 64, 65, 68, 69, 73, 78, 81, 82, 85, 86, 87, 89
Abnahmetest 44
Integrationstest 39
Komponententest 37
Systemintegrationstest 39
Systemtest 41
Fehlhandlung 13, 17, 26, 69
formales Review 51
funktionaler Test 32, 35, 45, 48, 63, 70
funktionales Qualitätsmerkmal 45
Gebrauchstauglichkeit 45, 46, 73, 75, 78, 83
Gebrauchstauglichkeitstest 48, 76
Grenzwert 65, 66
Grenzwertanalyse 46, 62, 65
Grundursache 13, 18
Grundursachenanalyse 17, 18, 37, 58
Gutachter 56, 58, 59, 61
informelles Review 51, 57
Inspektion 51, 58
Integrationstest 32, 37, 38, 39, 40, 50, 65, 89
Interoperabilitätstest 35
intrusiv (Testwerkzeug) 89
intuitive Testfallermittlung 62, 69
ISO-Norm
20246 54, 56
25010 46
29119-1 16
29119-2 21
29119-3 27, 76, 82, 87
29119-4 64
Istergebnis 26, 92
IT-Sicherheit 39, 44, 45, 46, 49, 54, 73, 75, 78, 79, 83
IT-Sicherheitsschwachstellen 42
IT-Sicherheitstest 48
Kanban 34
kommerzielle Standardsoftware 32, 35, 78
Kompatibilität 45, 49, 75, 83
Komponentenintegrationstest 16, 32, 38, 39, 47, 48, 49, 76
Komponententest 16, 32, 36, 37, 38, 39, 45, 46, 48, 49, 68, 76, 89
Konfigurationsmanagement 71, 74, 83
Konformität 15, 39, 43, 79
konkreter Testfall 29
Lernziele 100
Mastertestkonzept 76
Metrik 55, 59, 74, 76, 80, 81, 82, 94
modellbasiertes Testen 52, 90, 92
Moderator 56

nicht-funktionale Eigenschaft 75, 77
nicht-funktionaler Test 32, 35, 46, 48, 63, 70
nicht-funktionales Qualitätsmerkmal 45, 46, 49
Notfallwiederherstellung 42
Pairing 57
Peer-Review 57
Performanz 44, 45, 46, 49, 73, 75, 78, 83, 90, 91
Performanztest 42, 46, 76
Performanztestwerkzeug 89, 90
perspektivisches Lesen 51, 59, 60
Pestizid-Paradoxon 19
Pilotprojekt 94
Priorität 79, 82, 87
Produktrisiko 34, 71, 83
Projektrisiko 20, 34, 71, 83, 84
Pufferüberlauf 54
Qualität 13
Qualitätssteuerung 60
Qualitätsmanagement 17
Qualitätsmerkmal 78, 79, 83
Qualitätssicherung 13, 17
Qualitätssteuerung 17
Rational Unified Process 34
Regressionstest 19, 26, 32, 34, 36, 40, 45, 47, 49, 50, 53, 77, 79, 89, 90, 91
regressionsvermeidend 77
Review 8, 15, 31, 33, 41, 51, 52, 53, 54, 55, 56, 57, 59, 60, 61, 84, 86, 87
- Ad-hoc- 59, *Siehe* Ad-hoc-Review
- checklistenbasiert 59
- checklistenbasiertes *Siehe* checklistenbasiertes Review
- Erfolgsfaktoren 60
- formales *Siehe* formales Review
- informelles *Siehe* informelles Review
- paarweises 57
- perspektivisch *Siehe* perspektivisches Lesen
- rollenbasiert 59, 60
- rollenbasiertes *Siehe* rollenbasiertes Review
- szenariobasiert 59
- szenariobasiertes *Siehe* szenariobasiertes Review
- technisches *Siehe* technisches Review

Reviewart 57
Reviewleiter 56
Reviewmoderator 56
Reviewprozess 54
Reviewsitzung 55, 56, 57, 58, 61
Risiko 71, 83, 85, 91
risikobasiertes Testen 71, 77, 85
Risikostufe 63, 71, 77, 104
rollenbasiertes Review 51
schlüsselwortgetriebenes Testverfahren 92
Scrum 34
session-based testing 69
Shift left 19
Sicherungs- und Wiederherstellungsverfahren 43, 103
Simulation 89
sitzungsbasiertes Testen 69
spezifikationsbasierte Testverfahren *Siehe* Black-Box-Testverfahren
Spiralmodell 34
Standardsoftware
- kommerziell 43, 44, 50

statische Analyse 51, 52
statischer Test 19, 30, 51, 52, 53, 57, 90
strukturbasierte Testverfahren *Siehe* White-Box-Testverfahren
strukturelle Testverfahren *Siehe* White-Box-Testverfahren
Systemintegrationstest 32, 35, 38, 48, 49
Systemtest 32, 38, 39, 40, 41, 48, 49
Szenarien 59

szenariobasiertes Review 51

technisches Review 51, 58

Test 19

- änderungsbezogener 47
- dynamischer *Siehe* dynamischer Test
- funktionaler *Siehe* funktionaler Test
- nicht-funktionaler *Siehe* nicht-funktionaler Test
- risikobasierter *Siehe* risikobasiertes Testen
- schlüsselwortgetriebener *Siehe* schlüsselwortgetriebener Test
- statischer *Siehe* statischer Test

test driven development 37

test first 37

Testablauf 13, 25, 28, 29, 75, 79

Testabschluss 13, 26, 29

Testabschlussbericht 27, 29, 30, 71, 74, 81, 82

Testanalyse 13, 22, 23, 24, 25, 28, 29, 33, 63, 74, 75, 76

Testarbeitsergebnis 27, 29

Testart 20, 32, 35, 39, 45, 47, 49, 70, 73, 75, 76, 77, 85

Testaufwandschätzung 79

Testausführungsplan 13, 25, 28, 75, 79

Testausführungswerkzeug 88, 92

Testautomatisierung 34, 36, 37, 38, 40, 47, 49, 75, 77, 88, 89, 91, 92

Testbasis 13, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 32, 36, 38, 40, 43, 50, 54, 63, 64, 75, 77, 79

Testbedingung 13, 23, 24, 25, 26, 28, 29, 45, 46, 63, 64, 70, 75, 77, 81, 82

Testbericht 81

Test-Charta 24, 25, 28

Testdaten 13, 18, 19, 24, 25, 27, 28, 29, 63, 64, 75, 77, 78, 84, 89, 90, 91, 92

Testdurchführung 13, 25, 29, 74, 76

Testelement 23, 26, 29, 81, 83, 86

Testen 13, 15, 16, 17, 19, 30, 31

- datengetriebenes- *Siehe* datengetriebenes Testen
- unabhängiges Testen 73
- Ziele des Testens 15

Testentwurf 13, 15, 17, 18, 22, 24, 25, 28, 31, 33, 36, 37, 40, 45, 46, 47, 53, 60, 63, 74, 75, 76, 77, 84, 90, 91, 103

Tester 16, 30, 31, 33, 71, 73, 74, 75

Testfall 13, 24, 25, 26, 28, 29, 32, 36, 37, 43, 45, 46, 47, 50, 52, 62, 63, 64, 65, 66, 68, 69, 70, 75, 77, 78, 79, 81, 82, 84, 85, 87, 89, 90

Testfortschrittsbericht 23, 30, 71, 74

testgetriebene Entwicklung 37

Testinfrastruktur 27

Testkonzept 22, 27, 52, 71, 74, 75, 76, 82

Testmanagementwerkzeug 30, 88, 90, 92

Testmanager 30, 71, 74, 75, 86

Testmetriken *Siehe* Metrik

Testmittel 13, 18, 24, 25, 26, 27, 29, 52, 74, 77, 83, 90

Testobjekt 13, 15, 26, 32, 36, 38, 41, 44, 63, 68, 81, 82, 92, 93

Testorakel 13

Testorganisationen 73

Testplanung 13, 22, 27, 71, 76

Testprozess 13, 20

Testprozessreife 27

Testrahmen 25, 90

Testrealisierung 13, 22, 25, 28, 29, 63, 74, 76, 90

Testrichtlinie 74

Testschätzung 71, 79

Testschätzverfahren 80

- Breitband-Delphi 80
- expertenbasiert 80
- metrikbasiert 80
- Planungspoker 80

Teststeuerung 13, 22, 23, 27, 28, 29, 71, 76, 80, 81

Teststrategie 71, 74, 77

Teststufe 32, 35, 47

Testsuite 13, 25, 28, 77, 79, 89, 94

Testüberwachung 13, 22, 23, 27, 28, 29, 71, 76, 80, 81, 85

Testumgebung 18, 25, 27, 28, 32, 36, 38, 40, 75, 78, 81, 84, 86

Testunabhängigkeit 73

Testverfahren 62

Testvorgehensweise 71, 77

Testwerkzeug 26, 78, 88, 89, 90, 92

Testziel 13, 32

Überdeckung 13, 24, 45, 46, 47, 54, 59, 62, 64, 65, 66, 67, 68, 69, 70, 78, 80, 91

Überdeckungskriterien 21, 22, 23, 28, 29

Überdeckungswerkzeug 89

Übertragbarkeit 49, 73, 75, 83

Usability Test *Siehe* Gebrauchstauglichkeitstest

Use Case 67

Validierung 13, 15, 17, 31, 40, 42

Verfolgbarkeit 13, 20, 24, 25, 26, 27, 28, 29, 45, 46, 50, 54, 75, 83, 90, 93

Verifizierung 13, 15, 17, 25, 26, 28, 29, 31, 36, 37, 40, 42, 44

Vorleser 58

Walkthrough 51, 57

Wartbarkeit 50, 83

Wartungstest 32

White-Box-Test 32, 46, 48, 63

White-Box-Testverfahren 24, 62, 68, 69

Zustandsdiagramm 41

Zustandsübergangsdiagramm 66, 67

Zustandsübergangstabelle 67

Zustandsübergangstest 62, 66, 67

Zuverlässigkeit 45, 49, 77, 78, 79, 83

Zuverlässigkeitstest 48