Certified Tester

Foundation Level Syllabus

Version 2011 1.0.1

International Software Testing

Qualifications Board

Deutschsprachige Ausgabe. Herausgegeben durch

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

Certified Tester

Foundation Level Syllabus
(Deutschsprachige Ausgabe)

Übersetzung des englischsprachigen Lehrplans des International Software Testing Qualifications Board (ISTQB®), Originaltitel: Certified Tester, Foundation Level Syllabus, Version 2011.

Änderungsübersicht deutschsprachige Ausgabe

Version	Gültig ab:	Bemerkung
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

Dank

International Software Testing Qualifications Board Working Group Foundation Level (Ausgabe 2011): Thomas Müller (Leiter) und 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 nationalen Boards für die Verbesserungsvorschläge zur aktuellen Version des Lehrplans.

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 TestingQualifications Board Working Group Foundation Level (Ausgabe 2010): Thomas Müller (Leiter), RahulVerma, 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 und Erik van Veenendaal) und allen nationalen Boards für die Verbesserungsvorschläge zur aktuellen Version des Lehrplans.

Das German Testing Board (GTB), das Swiss Testing Board (STB) und das Austrian Testing Board (ATB) danken ebenso dem Review-Team 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 Working Group Foundation Level (Ausgabe 2007): Thomas Müller (Leiter), 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 Verbesserungsvorschläge zur aktuellen Version des Lehrplans durch die nationalen Boards.

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).

Einführung

Zweck des Dokuments

Dieser Lehrplan definiert die **Basisstufe** (Foundation Level) des Softwaretest-Ausbildungsprogramms des International Software Testing Qualifications Board (im Folgenden kurz ISTQB® genannt). Das ISTQB® stellt den Lehrplan den **nationalen Testing Boards** zur Verfügung, damit diese in ihrem Zuständigkeitsbereich Ausbildungsanbieter akkreditieren, Prüfungsfragen in der jeweiligen Landessprache erarbeiten und Prüfungsinstitutionen zur Verfügung stellen können. An Hand des Lehrplans erstellen **Ausbildungsanbieter** ihre Kursunterlagen und legen eine angemessene Unterrichtsmethodik für die Akkreditierung fest. Die Lernenden bereiten sich anhand des Lehrplans auf die Prüfung vor.
Weitere Informationen über Geschichte und Hintergrund dieses Lehrplans sind im Anhang A dieses Lehrplans aufgeführt.

Der ISTQB®Certified Tester, Foundation Level

Die Basisstufe des Certified Tester Ausbildungsprogramms soll alle in das Thema Softwaretesten involvierten Personen ansprechen. Das schließt Personen in Rollen wie Tester, Testanalysten, Testingenieure, Testberater, Testmanager, Abnahmetester und Softwareentwickler mit ein. Die Basisstufe richtet sich ebenso an Personen in der Rolle Projektleiter, Qualitätsmanager, Softwareentwicklungsmanager, Systemanalytiker (Business Analysten), IT-Leiter oder Managementberater, welche sich ein Basiswissen und Grundlagenverständnis über das Thema Softwaretesten erwerben möchten. Das Foundation Level Zertifikat ist Voraussetzung, um die Prüfung zum Certified Tester Advanced Level (Aufbaustufe) zu absolvieren.

Lernziele/Kognitive Stufen des Wissens

Jeder Abschnitt dieses Lehrplans ist einer kognitiven Stufe zugeordnet:

- K1: sich erinnern
- K2: verstehen
- K3: anwenden
- K4: analysieren

Weitere Details und Beispiele von Lernzielen finden Sie in Anhang B.

Alle Begriffe, die im Absatz direkt unter der Überschrift unter „Begriffe“ genannt werden, sollen wiedergegeben werden können (K1), auch wenn das in den Lernzielen nicht explizit angegeben ist. Es gelten die Definitionen des ISTQB Glossars bzw. der nationalen Übersetzung in der jeweils freigegebenen Fassung.

Die Prüfung

Auf diesem Lehrplan basiert die Prüfung für das Foundation Level Zertifikat. **Eine Prüfungsfrage kann Stoff aus mehreren Kapiteln des Lehrplans abfragen. Alle Abschnitte (Kapitel** 1 **bis** 6**) dieses Lehrplans können geprüft werden.**

Das Format der Prüfung ist Multiple Choice.

Prüfungen können unmittelbar im Anschluss an einen akkreditierten Ausbildungslehrgang oder Kurs, aber auch unabhängig davon (z.B. in einem Prüfzentrum oder einer öffentlich zugänglichen Prüfung) abgelegt werden. Die Teilnahme an einem Kurs stellt keine Voraussetzung für das Ablegen der Prüfung dar. Die von den nationalen Testing Boards zugelassenen Prüfungsanbieter sind auf der jeweiligen Homepage im Internet aufgelistet (z.B. German Testing Board e.V.: www.german-testing-board.info).

Akkreditierung

Nationale ISTQB Boards dürfen Ausbildungsanbieter akkreditieren, deren Ausbildungsunterlagen entsprechend diesem Lehrplan aufgebaut sind. Die Akkreditierungsrichtlinien können bei diesem nati-

onalen 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 als Bestandteil eine ISTQB® Prüfung enthalten.

Weitere Hinweise für Ausbildungsanbieter sind in Anhang D enthalten.

Detaillierungsgrad

Der Detaillierungsgrad dieses Lehrplans erlaubt international konsistentes Lehren und Prüfen. Um dieses Ziel zu erreichen, enthält dieser Lehrplan Folgendes:

- allgemeine Lernziele, welche die Intention der Basisstufe beschreiben
- Inhalte, die zu lehren sind, mit einer Beschreibung und, wo notwendig, Referenzen zu weiterführender Literatur
- Lernziele für jeden Wissensbereich, welche das beobachtbare kognitive Ergebnis der Schulung und die zu erzielende Einstellung des Teilnehmers beschreiben
- eine Liste von Begriffen, welche der Teilnehmer wiedergeben und verstehen soll
- eine Beschreibung der wichtigen zu lehrenden Konzepte, einschließlich Quellen wie anerkannte Fachliteratur, Normen und Standards

Der Lehrplan ist keine vollständige Beschreibung des Wissensgebiets „Softwaretesten“. Er reflektiert lediglich den nötigen Umfang und Detaillierungsgrad, welcher für die Lernziele des Foundation Level relevant ist.

Lehrplanaufbau

Der Lehrplan besteht aus 6 Hauptkapiteln. Jeder Haupttitel eines Kapitels zeigt die anspruchsvollste Lernzielkategorie/höchste kognitive Stufe, welche mit dem jeweiligen Kapitel abgedeckt werden soll und legt die Unterrichtszeit fest, welche in einem akkreditierten Kurs mindestens für dieses Kapitel aufgewendet werden muss.

Beispiel:

2 Testen im Softwarelebenszyklus (K2)	115 Minuten

Das Beispiel zeigt, dass in Kapitel 2 Lernziele K1 (ein Lernziel einer höheren Taxonomiestufe impliziert die Lernziele der tieferen Stufen) und K2 (aber nicht K3) erwartet werden und 115 Minuten für das Lehren des Materials in diesem Kapitel vorgesehen sind.

Jedes Kapitel enthält eine Anzahl von Unterkapiteln. Jedes Unterkapitel kann wiederum Lernziele und einen Zeitrahmen vorgeben. Wird bei einem Unterkapitel keine Zeit angegeben, so ist diese im Oberkapitel bereits enthalten.

1 Grundlagen des Softwaretestens (K2) — 155 Minuten

Lernziele für Grundlagen des Softwaretestens

Die Lernziele legen fest, was Sie nach Beenden des jeweiligen Moduls gelernt haben sollten.

1.1Warum sind Softwaretests notwendig? (K2)

LO-1.1.1 Anhand von Beispielen beschreiben können, auf welche Art ein Softwarefehler Menschen, der Umwelt oder einem Unternehmen Schaden zufügen kann (K2)

LO-1.1.2 Zwischen der Ursache eines Fehlerzustands und seinen Auswirkungen unterscheiden können (K2)

LO-1.1.3 Anhand von Beispielen herleiten können, warum Testen notwendig ist (K2)

LO-1.1.4 Beschreiben können, warum Testen Teil der Qualitätssicherung ist und Beispiele angeben, wie Testen zu einer höheren Qualität beiträgt (K2)

LO-1.1.5 Die Begriffe Fehlhandlung, Fehler/Fehlerzustand, Fehlerwirkung und die zugehörigen Definitionen anhand von Beispielen erklären und vergleichen können (K2)

1.2 Was ist Softwaretesten? (K2)

LO-1.2.1 Die allgemeinen Zielsetzungen des Testens kennen (K1)

LO-1.2.2 Beispiele für Testziele der verschiedenen Phasen des Softwarelebenszyklus nennen können (K2)

LO-1.2.3 Zwischen Testen und Debugging unterscheiden können (K2)

1.3Die sieben Grundsätze des Softwaretestens (K2)

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

1.4 Fundamentaler Testprozess (K1)

LO-1.4.1 Die fünf Hauptaktivitäten des fundamentalen Testprozesses und die jeweiligen Aufgaben von der Testplanung bis zum Abschluss der Testaktivitäten kennen (K1)

1.5 Die Psychologie des Testens (K2)

LO-1.5.1 Die psychologischen Faktoren kennen, die Einfluss auf den Testerfolg haben (K1)

LO-1.5.2 Zwischen der unterschiedlichen Denkweise eines Testers und eines Entwicklers differenzieren können (K2)

1.1 Warum sind Softwaretests notwendig? (K2) — 20 Minuten

Begriffe
Fehler, Fehlerwirkung, Fehlerzustand/ Defekt, Fehlhandlung, Qualität, Risiko

1.1.1 Softwaresystemzusammenhang (K1)

Softwaresysteme sind aus dem täglichen Leben nicht wegzudenken, angefangen von Business-Software (z.B. Bankanwendungen) bis hin zu Gebrauchsgegenständen (z.B. Autos). Die meisten Endanwender haben bereits schlechte Erfahrungen mit Softwaresystemen gemacht, die nicht so funktioniert haben wie erwartet. Software, die nicht korrekt funktioniert, kann zu vielerlei Problemen führen, wie Geld-, Zeit- oder Imageverlust oder sogar zu Personenschäden, wie Verletzungen oder Tod.

1.1.2 Ursachen von Softwarefehlern (K2)

Ein Mensch kann eine Fehlhandlung begehen, die einen Fehlerzustand im Programmcode oder in einem Dokument verursacht. Wenn der fehlerhafte Code ausgeführt wird, wird das System möglicherweise nicht das tun, was es tun sollte (oder etwas tun, was es nicht tun sollte) und dabei eine Fehlerwirkung hervorrufen. Fehler in Software, Systemen oder Dokumenten können, müssen aber nicht zu einer Fehlerwirkung führen.

Fehlerzustände treten auf, weil Menschen Fehlhandlungen begehen, z.B. unter Zeitdruck, bei komplexem Code, durch Komplexität der Infrastruktur, bei sich ändernden Technologien, und/oder vielen Systemwechselbeziehungen.

Fehlerwirkungen können aber auch durch Umgebungsbedingungen hervorgerufen werden. Zum Beispiel können Strahlung, elektromagnetische Felder oder Schmutz Fehlerzustände in der Firmware verursachen; ebenso kann die Ausführung der Software durch das Ändern von Hardwarezuständen beeinflusst werden.

1.1.3 Die Rolle des Testens bei Entwicklung, Wartung und Betrieb von Software (K2)

Intensives Testen von Systemen und Dokumentation kann helfen das Risiko zu reduzieren, dass Probleme im operativen Betrieb auftreten. Weiterhin kann es dazu beitragen, die Qualität des Softwaresystems zu erhöhen, indem Fehlerzustände vor der betrieblichen Freigabe gefunden und behoben werden.

Softwaretesten kann auch notwendig sein, um vertragliche oder gesetzliche Vorgaben oder spezielle Industrienormen zu erfüllen.

1.1.4 Testen und Qualität (K2)

Testen ermöglicht es, die Qualität von Software zu messen. Qualität wird hier ausgedrückt durch die Anzahl gefundener Fehlerzustände. Das gilt sowohl für funktionale als auch für nicht-funktionale Anforderungen und Qualitätsmerkmale (z.B. Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit); für weitere Informationen zum Thema nicht-funktionales Testen siehe Kapitel 2; für weitere Information über Softwarequalitätsmerkmale siehe die Norm ‘Software engineering – Product-quality’ (ISO 9126-1, 2001).

Wenn wenige oder keine Fehlerzustände gefunden werden, kann Testen Vertrauen in die Qualität eines Systems schaffen. Ein angemessen spezifizierter Test, der keine Fehler zeigt, reduziert das allgemeine Risikoniveau in einem System. Falls Testen Fehlerzustände findet und diese Fehlerzustände behoben werden, steigt die Qualität des Softwaresystems.

Aus den Fehlern vorangegangener Projekte sollte gelernt werden. Wenn man die Fehlerursachen verstanden hat, die beim Test in anderen Projekten gefunden wurden, kann man Entwicklungsprozesse zielgerichtet verbessern. Das wiederum beugt dem erneuten Auftreten der Fehlerzustände vor und sollte als Konsequenz die Qualität zukünftiger Systeme verbessern. Das ist ein Aspekt der Qualitätssicherung.

Testen sollte als eine Qualitätssicherungsmaßnahme in den Entwicklungsprozess integriert sein (neben beispielsweise Entwicklungsstandards, Schulung und Fehlerursachenanalyse).

1.1.5 Wie viel Testaufwand ist notwendig? (K2)

Um zu entscheiden, wie viel Testen notwendig ist, sollte das Risikoniveau berücksichtigt werden. Das schließt sowohl technische, Betriebssicherheits- und wirtschaftliche Risiken, als auch Projektrandbedingungen, wie Zeit und Budget ein. (Risiko wird im Kapitel 5 weiter diskutiert.)

Testen sollte den Beteiligten genügend Informationen liefern, um fundierte Entscheidungen über die Freigabe der getesteten Software oder des Systems treffen zu können. Die Freigabe kann die Übergabe des Systems an den nächsten Entwicklungsschritt bedeuten oder die Übergabe des Systems an die Kunden.

1.2 Was ist Softwaretesten? (K2)	30 Minuten

Begriffe
Anforderung, Debugging, Review, Testen, Testfall, Testziel

Hintergrund

Verbreitet ist die Auffassung, dass Testen nur aus dem Ausführen von Tests, d.h. dem Ausführen der Software, besteht. Dabei handelt es sich jedoch nur um einen Teilbereich des Testens.

Weitere Testaktivitäten sind vor und nach der Testdurchführung angesiedelt. Dazu gehören: Planung und Steuerung der Tests, Auswahl der Testbedingungen, Testfallspezifikation, Ausführung der Testfälle, Überprüfung der Ergebnisse, Auswertung der Endekriterien, Berichten über den Testprozess und das zu testende System sowie nach Abschluss einer Testphase Abschlussarbeiten zu Ende zu bringen. Zum Testen zählt ebenfalls das Prüfen von Dokumenten (Quellcode inbegriffen) und die Durchführung von statischen Analysen.

Sowohl das dynamische Testen als auch das statische Testen können als Mittel zur Erreichung ähnlicher Zielsetzungen eingesetzt werden. Dabei werden Informationen zur Verbesserung des zu testenden Systems, des Entwicklungs- und des Testprozesses geliefert.

Testen kann die folgenden Ziele haben:
- Aufdecken von Fehlerzuständen
- Erzeugen von Vertrauen bezüglich des Qualitätsniveaus des Systems
- Liefern von Informationen zur Entscheidungsfindung
- Vorbeugen von Fehlerzuständen

Ein konsequenter Prozess und ein Beginn der Tätigkeiten zur Erstellung von Tests schon früh im Lebenszyklus (das Prüfen der Testbasis durch den Testentwurf) kann Fehler im Programmcode verhindern. Reviews von Dokumenten (z.B. Anforderungsspezifikation) sowie Identifizierung und Lösung von Problemen kann ebenfalls Fehler im Programmcode verhindern.

Aus den verschiedenen Zielsetzungen beim Testen ergeben sich verschiedene Gesichtspunkte. Zum Beispiel könnte bei herstellerinternen Tests im Testentwurf (z.B. Komponententest, Integrationstest oder Systemtest) das Hauptziel sein, so viele Fehlerwirkungen wie möglich zu verursachen, so dass Fehlerzustände in der Software identifiziert und behoben werden können. Demgegenüber könnte im Abnahmetest das Hauptziel sein, zu bestätigen, dass das System wie erwartet funktioniert, um Vertrauen zu schaffen, dass es den Anforderungen entspricht. In manchen Fällen könnte das Hauptziel des Testens sein, die Softwarequalität zu bewerten (ohne die Absicht Fehlerzustände zu beheben), um die Beteiligten über das Risiko einer Systemfreigabe zu einem bestimmten Zeitpunkt zu informieren. Wartungstests enthalten oft Tests, die sicherstellen sollen, dass durch die Änderung der Software keine neuen Fehler eingebaut wurden. Beim Betriebstest (orientiert an Nutzungsprofilen) kann das Hauptziel sein, ein System hinsichtlich Ausprägungen wie Zuverlässigkeit oder Verfügbarkeit zu bewerten.

Debugging und Testen sind verschiedene Dinge. Dynamische Tests können Fehlerwirkungen zeigen, die durch Fehlerzustände verursacht werden. Debugging ist eine Entwicklungsaktivität, die die Ursache einer Fehlerwirkung identifiziert, analysiert und entfernt. Anschließende Fehlernachtests durch einen Tester stellen sicher, dass die Lösung wirklich die Fehlerwirkung behoben hat. Die Verantwortung für Testen liegt üblicherweise beim Tester, die für Debugging beim Entwickler.

Der Testprozess und seine Aktivitäten werden in Abschnitt 1.4 näher behandelt.

1.3 Die sieben Grundsätze des Softwaretestens (K2)	**35 Minuten**

Begriffe
erschöpfender Test

Grundsätze

In den letzten 40 Jahren haben sich folgende Grundsätze des Testens herauskristallisiert, die als generelle Leitlinien beim Testen angesehen werden.

Grundsatz 1: Testen zeigt die Anwesenheit von Fehlerzuständen
Mit Testen wird das Vorhandensein von Fehlerzuständen nachgewiesen.
Mit Testen lässt sich nicht beweisen, dass keine Fehlerzustände im Testobjekt vorhanden sind. Ausreichendes Testen verringert die Wahrscheinlichkeit, dass noch unentdeckte Fehlerzustände im Testobjekt vorhanden sind. Selbst wenn keine Fehlerzustände im Test aufgezeigt wurden, ist das kein Nachweis für Fehlerfreiheit.

Grundsatz 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. Tests sind immer nur Stichproben, und der Testaufwand ist entsprechend Risiko und Priorität festzulegen.

Grundsatz 3: Mit dem Testen frühzeitig beginnen
Um Fehlerzustände frühzeitig zu finden, sollen Testaktivitäten im System- oder Softwarelebenszyklus so früh wie möglich beginnen und definierte Ziele verfolgen.

Grundsatz 4: Häufung von Fehlern
Der Testaufwand soll sich proportional zu der erwarteten und später beobachteten Fehlerdichte auf die Module fokussieren. Ein kleiner Teil der Module enthält gewöhnlich die meisten Fehlerzustände, die während der Testphase entdeckt werden oder ist für die meisten Fehlerwirkungen im Betrieb verantwortlich.

Grundsatz 5: Wiederholungen haben keine Wirksamkeit
Wiederholungen der immer gleichen Testfälle führen nicht zu neuen Erkenntnissen. Damit die Effektivität der Tests nicht abnimmt, sind die Testfälle regelmäßig zu prüfen und neue oder modifizierte Testfälle zu erstellen. Bisher nicht geprüfte Teile der Software oder unberücksichtigte Konstellationen bei der Eingabe werden dann ausgeführt und somit mögliche weitere Fehlerzustände nachgewiesen.

Grundsatz 6: Testen ist abhängig vom Umfeld
Je nach Einsatzgebiet und Umfeld des zu prüfenden Systems ist das Testen anzupassen. Sicherheitskritische Systeme werden beispielsweise anders getestet als E-Commerce-Systeme.

Grundsatz 7: Trugschluss: „Keine Fehler“ bedeutet ein brauchbares System
Fehlerzustände zu finden und zu beseitigen, hilft nicht, wenn das gebaute System nicht nutzbar ist und nicht den Vorstellungen und Erwartungen der Nutzer entspricht.

1.4 Fundamentaler Testprozess (K1) — 35 Minuten

Begriffe
Abweichung, Endekriterien, Fehlernachtest, Mastertestkonzept, Regressionstest, Testablauf, Testabschlussbericht, Testbasis, Testbedingung, Testdaten, Testdurchführung, Testkonzept, Testmittel, Testprotokoll, Testrichtlinie, Testsuite, Testüberdeckung

Hintergrund

Die Testdurchführung ist der sichtbarste Teil des Testens. Um effektiv und effizient zu sein, müssen Testkonzepte darüber hinaus aber Zeit vorsehen, um die Tests zu planen, Testfälle zu spezifizieren und die Testdurchführung vorzubereiten, sowie die Testergebnisse zu bewerten.

Der fundamentale Testprozess besteht aus den folgenden Hauptaktivitäten:

- Testplanung und Steuerung
- Testanalyse und Testentwurf
- Testrealisierung und Testdurchführung
- Bewertung von Endekriterien und Bericht
- Abschluss der Testaktivitäten

Auch wenn sie hier logisch sequentiell aufgelistet sind, können all diese Testprozessaktivitäten in der Praxis zeitlich überlappend oder parallel stattfinden. Gewöhnlich ist es nötig, Ausprägungen und Reihenfolge dieser Hauptaktivitäten jeweils dem zu testenden System oder dem Projekt anzupassen.

1.4.1 Testplanung und Steuerung (K1)

Zur Testplanung gehören folgende Aktivitäten: die Definition der Testziele und die Festlegung der Testaktivitäten, die notwendig sind, um Aufgabenumfang und Testziele erreichen zu können.

Teststeuerung ist die fortlaufende Aktivität, den aktuellen Testfortschritt mit dem Plan zu vergleichen und den Status, einschließlich eventueller Abweichungen, zu berichten. Dazu gehört gegebenenfalls auch das Einleiten von Korrekturmaßnahmen. Um Tests steuern zu können, ist es notwendig, projektbegleitend geeignete Fortschrittsdaten zu ermitteln. Die Testplanung muss Feedback aus solchen Überwachungs- und Steuerungsaktivitäten berücksichtigen und die Pläne entsprechend fortschreiben.

Aufgaben der Testplanung und –steuerung werden im Kapitel 5 des Lehrplans im Detail behandelt.

1.4.2 Testanalyse und Testentwurf (K1)

Testanalyse und -entwurf ist die Aktivität, in der die allgemeinen Testziele zu konkreten Testbedingungen und Testfällen verfeinert werden.

Dies umfasst die folgenden Hauptaufgaben:

- Review der Testbasis (z.B. Anforderungen, Software Integrity Level[1] (Risikoausmaß), Risikoanalysebericht, Architektur, Design, Schnittstellenspezifikation)
- Bewertung der Testbarkeit von Testbasis und Testobjekten
- Identifizierung und Priorisierung der Testbedingungen auf Grundlage der Testobjektanalyse, der Spezifikation, des Verhaltens und der Struktur der Software
- Entwurf (Design) und Priorisierung von abstrakten Testfällen
- Identifizierung benötigter Testdaten, um Definition von Testbedingungen und Testfällen zu unterstützen

[1] Der Erfüllungsgrad einer Menge vom Stakeholder ausgewählter Software- und/oder Software-basierter Merkmale (z.B. Softwarekomplexität, Risikoeinstufung, Sicherheitsstufe (Zugriffsschutz) und funktionalen Sicherheit, gewünschte Performanz, Zuverlässigkeit, oder Kosten), die definiert wurden, um die Bedeutung der Software für den Stakeholder zum Ausdruck zu bringen.

- Entwurf des Testumgebungsaufbaus und Identifikation der benötigten Infrastruktur und Werkzeuge
- Erzeugen (bzw. Sicherstellung) der Rückverfolgbarkeit zwischen Testbasis und Testfällen in beiden Richtungen

1.4.3 Testrealisierung und Testdurchführung (K1)

Testrealisierung und -durchführung ist die Aktivität, bei der unter Berücksichtigung aller anderen Informationen, die zur Testdurchführung nötig sind, Testabläufe und Testskripte spezifiziert werden, indem Testfälle in einer besonderen Reihenfolge kombiniert werden. Des Weiteren wird die Testumgebung in dieser Phase entsprechend konfiguriert und genutzt.

Testrealisierung und -durchführung umfassen die folgenden Hauptaufgaben:

- Endgültige Festlegung, Realisierung und Priorisierung von Testfällen (einschließlich Festlegung der Testdaten)
- Erstellung und Priorisierung des Testablaufs, Erstellung der Testdaten, der Testszenarien und optional Vorbereitung der Testrahmen und Entwicklung von Skripten zur Testautomatisierung
- Erstellung von Testsuiten basierend auf dem Testablauf, um die Testdurchführung möglichst effizient zu gestalten
- Kontrolle, ob die Testumgebung korrekt aufgesetzt wurde und Sicherstellung der richtigen Konfigurationen
- Überprüfung und Aktualisierung der Rückverfolgbarkeit zwischen Testbasis und Testfällen in beide Richtungen
- Ausführung von Testabläufen (manuell oder automatisiert) unter Einhaltung des Testplans (Reihenfolge, Testsuiten etc.)
- Protokollierung der Testergebnisse und Dokumentation der genauen Version des jeweiligen Testobjekts und der eingesetzten Testwerkzeugen und Testmittel
- Vergleich der Ist-Ergebnisse mit den vorausgesagten Ergebnissen
- gefundene Fehlerwirkungen oder Abweichungen festhalten und analysieren, um den Grund eines Problems festzustellen (z.B. Fehler im Code, in spezifizierten Testdaten, im Testdokument oder Fehler bei der Durchführung passiert)
- Alle Testfälle, die eine Fehlerwirkung aufgedeckt haben, müssen nach der Behebung der jeweiligen Ursachen nochmals getestet werden (Fehlernachtest). Ein Fehlernachtest wird durchgeführt, um sicherzustellen, dass eine Fehlerbehebung in der Software den gewünschten Erfolg gebracht hat. Darüber hinaus sind weitere Testwiederholungen (Regressionstest) nötig, um sicherzustellen, dass die Fehlerbehebung bzw. Softwareänderung keinen negativen Einfluss auf bereits bestehende Funktionalität hatte, oder dass nicht weitere (bisher maskierte) Fehlerzustände freigelegt wurden.

1.4.4 Bewertung von Endekriterien und Bericht (K1)

Mit der Bewertung der Endekriterien/Testauswertung werden die Testaktivitäten auf ihre Ziele hin untersucht. Diese Phase sollte in jeder Teststufe abgehandelt werden (siehe Abschnitt 2.2Teststufen (K2)).

Zu Testauswertung und -bericht gehören folgende Hauptaufgaben:

- Auswertung der Testprotokolle in Hinblick auf die im Testkonzept festgelegten Endekriterien
- Entscheidung, ob mehr Tests durchgeführt oder die festgelegten Endekriterien angepasst werden müssen
- Erstellung des Testabschlussberichts für die Stakeholder

1.4.5 Abschluss der Testaktivitäten (K1)

Während des Abschlusses der Testaktivitäten werden Daten von abgeschlossenen Aktivitäten vorangegangener Testphasen gesammelt und konsolidiert (Erfahrungen, Testmittel, Fakten, Zahlen). Testabschlussaktivitäten finden im Rahmen von Projektmeilensteinen statt; beispielsweise wenn eine Software in Betrieb genommen wird, ein Testprojekt abgeschlossen (oder abgebrochen) wird, ein Meilenstein erreicht wird oder ein Wartungs-Release (maintenance release) abgeschlossen ist.

Der Abschluss der Testaktivitäten umfasst folgende Hauptaufgaben:

- Kontrolle, welche der geplanten Arbeitsergebnisse geliefert wurden,
- Schließung der Fehler-/Abweichungsberichte oder Erstellung von Änderungsanforderungen für weiter bestehende Fehler/Abweichungen
- Dokumentation der Abnahme des Systems
- Dokumentation und Archivierung der Testmittel, Testumgebung und der Infrastruktur für spätere Wiederverwendung
- Übergabe der Testmittel an die Wartungsorganisation
- Analyse und Dokumentation von „lessons learned“, um nötige Änderungen für spätere Projekte abzuleiten
- Nutzung der gesammelten Informationen, um die Testreife zu verbessern

1.5 Die Psychologie des Testens (K2) — 25 Minuten

Begriffe
intuitive Testfallermittlung, Unabhängigkeit

Hintergrund

Die Einstellung zu Testdurchführung und Reviewphase unterscheidet sich von derjenigen bei der Softwareentwicklung.

Mit der richtigen Einstellung sind auch Entwickler fähig, ihren eigenen Code zu testen. Die Verlagerung dieser Verantwortung auf einen Tester hat neben der Verteilung des Aufwands jedoch noch weitere Vorteile: eine unabhängige Sichtweise von geschulten, professionellen Testexperten. Unabhängiges Testen kann in jeder Teststufe angewendet werden.

Ein gewisser Grad an Unabhängigkeit steigert die Effektivität des Testers bei der Fehlersuche (Betriebsblindheit). Unabhängigkeit ist allerdings auf keinen Fall ein Ersatz für Erfahrung (Vertrautheit) mit der Software, auch Entwickler können effizient viele Fehler in ihrem eigenen Code finden.

Folgende Stufen der Unabhängigkeit (von niedrig nach hoch) können definiert werden:

- Der Test wird vom Entwickler für den eigenen Code durchgeführt (keine Unabhängigkeit).
- Der Test wird von einem anderen Entwickler durchgeführt (z.B. innerhalb des Entwicklungsteams).
- Der Test wird von ein oder mehreren Personen aus einer anderen organisatorischen Einheit (z.B. unabhängiges Testteam) oder einem Testspezialisten (z.B. Spezialist für Benutzungsfreundlichkeit oder Performance) durchgeführt.
- Der Test wird von ein oder mehreren Personen außerhalb der (entwickelnden) Organisation (z.B. internes Testlabor) oder der Firma durchgeführt (d.h. Outsourcing oder Zertifizierung durch externe Institutionen).

Mitarbeiter und Projekte werden durch Zielsetzungen angetrieben. Mitarbeiter neigen dazu, ihre Pläne an die Ziele anzupassen, die ihnen vom Management oder anderen Stakeholdern vorgegeben werden. So versucht ein Tester, Fehlerzustände in der Software zu finden oder zu bestätigen, dass die Software die Ziele erfüllt. Daher ist es sehr wichtig, die Ziele des Testens klar aufzuzeigen.

Das Aufdecken von Fehlerwirkungen während der Testphase könnte als Kritik gegen den Autor oder das Produkt aufgefasst werden. Testen wird daher oft als destruktive Tätigkeit angesehen, obwohl es sehr konstruktiv für das Management von Produktrisiken ist. Ein guter Tester benötigt für seine Fehlersuche viel Neugier, professionellen Pessimismus, eine kritische Einstellung, ein Auge fürs Detail, ein gutes Kommunikationsverhalten gegenüber den Entwicklern und Erfahrung, auf die er der intuitiven Testfallermittlung (Error Guessing) zurückgreifen kann.

Wenn Fehler, Fehlerzustände oder Fehlerwirkungen positiv kommuniziert werden, können Schwierigkeiten zwischen Testern und Entwicklern, Analysten und Designern vermieden werden. Das gilt sowohl für Fehlerzustände, die während Reviews gefunden werden, als auch für die, die beim Test gefunden werden.

Tester und Testmanager müssen eine ausgeprägte Kontaktfähigkeit und gute kommunikative Fähigkeiten besitzen, um sachbezogene Informationen über gefundene Fehlerzustände, Fortschritte und Risiken austauschen zu können. Einem Autor eines Dokuments oder einem Entwickler kann die Information über den gefundenen Fehlerzustand helfen, seine Qualifikation zu verbessern. Werden Fehlerzustände während der Testphase gefunden und behoben, so spart das Zeit und Geld zu einem späteren Zeitpunkt (in der Produktion) und verringert das Risiko.

Kommunikationsprobleme können speziell dann auftreten, wenn Tester nur als Übermittler von schlechten Nachrichten angesehen werden. Es gibt aber einige Möglichkeiten, die Beziehung und die Kommunikation zwischen den Testern und ihrem Umfeld zu verbessern:

- Beginnen Sie mit der Zusammenarbeit anstatt zu streiten – erinnern Sie jeden an das zentrale Ziel: eine bessere Qualität der Software!
- Kommunizieren Sie gefundene Fehler eines Produkts neutral, sachbezogen und vermeiden Sie Kritik an der verantwortlichen Person. Schreiben Sie Fehler- und Abweichungsberichte beispielsweise objektiv und sachlich.
- Versuchen Sie, das Verhalten und die Gefühle der anderen Person zu verstehen.
- Stellen Sie sicher, dass Sie die andere Person richtig verstanden haben und umgekehrt.

1.6 Ethische Leitlinien — 10 Minuten

Durch ihre Tätigkeit im Bereich Softwaretesten erhalten Personen oft Zugang zu vertraulichen und rechtlich privilegierten Informationen. Damit diese Informationen nicht missbräuchlich verwendet werden, sind ethische Leitlinien nötig. In Anlehnung an den Ethik-Kodex von ACM und IEEE stellt das ISTQB die folgenden ethischen Leitlinien auf.

- ÖFFENTLICHKEIT
  Das Verhalten zertifizierter Softwaretester soll nicht im Widerspruch zum öffentlichen Interesse stehen.

- KUNDE UND ARBEITGEBER
  Das Verhalten zertifizierter Softwaretester soll den Interessen ihrer Kunden und Arbeitgeber entsprechen und dabei nicht im Widerspruch zum öffentlichen Interesse stehen.

- PRODUKT
  Zertifizierte Softwaretester sollen sicherstellen, dass die Arbeitsergebnisse, die sie liefern (für die von ihnen getesteten Produkte und Systeme), höchste fachliche Anforderungen erfüllen.

- URTEILSVERMÖGEN
  Zertifizierte Softwaretester sollen bei ihrer professionellen Beurteilung aufrichtig und unabhängig sein.

- MANAGEMENT
  Zertifizierte Softwaretestmanager und -testleiter sollen eine ethische Haltung beim Management des Softwaretestens haben und fördern.

- BERUFSBILD
  Zertifizierte Softwaretester sollen Integrität und Ansehen ihres Berufs fördern und dabei nicht im Widerspruch zum öffentlichen Interesse stehen.

- KOLLEGEN
  Zertifizierte Softwaretester sollen sich ihren Kolleginnen und Kollegen gegenüber fair und hilfsbereit verhalten und die Kooperation mit Softwareentwicklern fördern.

- PERSÖNLICH
  Zertifizierte Softwaretester sollen sich in ihrem Beruf lebenslang fort- und weiterbilden und eine ethische Haltung in ihrer Berufsausübung vertreten.

Referenzen

1 Linz, 2012
1.1.5 Black, 2001,Kaner, 2002
1.2 Beizer, 1990, Black, 2001, Myers 2001
1.3 Beizer, 1990, Hetzel, 1988, Myers 2001, Linz, 2012
1.4 Hetzel, 1988
1.4.5 Black, 2001, Craig, 2002
1.5 Black, 2001, Hetzel, 1988

2 Testen im Softwarelebenszyklus (K2) — 115 Minuten

Lernziele für den Abschnitt Testen im Softwarelebenszyklus

Die Lernziele legen fest, was Sie nach Beenden des jeweiligen Moduls gelernt haben sollten.

2.1 Softwareentwicklungsmodelle (K2)

LO-2.1.1 Beziehungen zwischen Entwicklungs- und Testaktivitäten und Arbeitsergebnissen im Entwicklungslebenszyklus anhand von Beispielen unter Verwendung von Projekt- und Produkteigenschaften erklären können (K2)

LO-2.1.2 Die Tatsache erkennen, dass Softwareentwicklungsmodelle an das Projekt und die Produkteigenschaften angepasst werden müssen (K1)

LO-2.1.3 Eigenschaften „guter" Tests nennen können, die in beliebigen Entwicklungslebenszyklen anwendbar sind (K1)

2.2 Teststufen (K2)

LO-2.2.1 Verschiedene Teststufen vergleichen können: Hauptziele, typische Testobjekte, typische Testziele (z.B. funktionale oder strukturelle) und entsprechende Arbeitsergebnisse, Rollen beim Testen, Arten von zu identifizierenden Fehlerzuständen und –wirkungen (K2)

2.3 Testarten (K2)

LO-2.3.1 Vier Arten des Softwaretestens (funktional, nicht-funktional, strukturell und änderungsbezogen) an Hand von Beispielen vergleichen können (K2)

LO-2.3.2 Erkennen, dass funktionale und strukturelle Tests auf jeder Teststufe angewendet werden können (K1)

LO-2.3.3 Nicht-funktionale Testarten auf Grundlage von nicht-funktionalen Anforderungen identifizieren und beschreiben können (K2)

LO-2.3.4 Testarten basierend auf der Analyse der Struktur oder Architektur des Softwaresystems identifizieren und beschreiben können (K2)

LO-2.3.5 Zweck eines Fehlernachtests und eines Regressionstests beschreiben können (K2)

2.4 Wartungstests (K2)

LO-2.4.1 Wartungstests (also Testen eines existierenden Systems) vergleichen können mit dem Testen einer neuen Anwendung bzgl. der Testarten, Auslöser des Testens und des Testumfangs (K2)

LO-2.4.2 Indikatoren für Wartungstests (also Modifikation, Migration und Außerbetriebnahme der Software) erkennen können (K1)

LO-2.4.3 Die Rolle von Regressionstests und Auswirkungsanalyse in der Softwarewartung beschreiben können (K2)

2.1 Softwareentwicklungsmodelle (K2) — 20 Minuten

Begriffe
iterativ-inkrementelles Entwicklungsmodell, kommerzielle Standardsoftware (COTS – Commercial Off The Shelf), V-Modell (allgemeines), Validierung, Verifizierung

Hintergrund
Testen findet nicht isoliert statt; Testaktivitäten sind immer bezogen auf Softwareentwicklungsaktivitäten. Verschiedene Entwicklungslebenszyklusmodelle erfordern verschiedene Testansätze.

2.1.1 V-Modell (sequentielles Entwicklungsmodell) (K2)

Es existieren Varianten des V-Modells. Das Gängigste, das allgemeine V-Modell, besteht aus fünf Entwicklungsstufen und vier Teststufen. Die vier Teststufen korrespondieren mit den Entwicklungsstufen.

Die vier in diesem Lehrplan verwendeten Teststufen nach dem allgemeinen V-Modell sind:

- Komponententest (unit test)
- Integrationstest
- Systemtest
- Abnahmetest

In der Praxis kann ein V-Modell, entsprechend dem Projekt(-vorgehen) oder Produkt, das entwickelt und getestet werden soll, weniger, mehr oder andere Stufen aufweisen. Beispielsweise kann es Komponentenintegrationstests nach Komponententests oder Systemintegrationstests nach Systemtests geben.

Entwicklungsdokumente (wie Geschäftsvorfälle oder Anwendungsfälle, Anforderungsspezifikationen, Entwurfsdokumente und Code), die während der Entwicklung entstehen, sind oft die Basis für Tests in einer oder mehreren Stufen.
Referenzen für generische Arbeitspapiere können unter anderem in folgenden Standards gefunden werden:

- CMMI-2004 (Capability Maturity Model Integration)
- IEEE/IEC 12207-2008 (‘Software life cycle processes’)

Verifizierung und Validierung (und früher Testentwurf) können während der Erstellung der Entwicklungsdokumente durchgeführt werden.

2.1.2 Iterativ-inkrementelle Entwicklungsmodelle (K2)

Bei iterativ-inkrementeller Entwicklung werden Anforderungen, Entwurf, Entwicklung und Test in einer Reihe kurzer Entwicklungszyklen durchlaufen. Beispiele hierfür sind Prototyping, Rapid Application Development (RAD), der Rational Unified Process (RUP) und agile Entwicklungsmodelle. Ein System, welches unter Nutzung dieser Entwicklungsmodelle erstellt wird, kann in jeder Iteration die verschiedenen Teststufen durchlaufen. Jedes Inkrement bzw. Erweiterung, die der bisherigen Entwicklung hinzugefügt wird, ergibt ein wachsendes System, das ebenso getestet werden muss. Regressionstests haben daher bei allen Iterationen nach dem ersten Zyklus eine zunehmende Bedeutung. Verifizierung und Validierung können für jede Erweiterung durchgeführt werden.

2.1.3 Testen innerhalb eines Entwicklungslebenszyklus (K2)

In jedem Entwicklungslebenszyklus findet man einige Charakteristika für gutes Testen:

- Zu jeder Entwicklungsaktivität gibt es eine zugehörige Aktivität im Testen.
- Jede Teststufe hat Testziele, die spezifisch für diese Stufe sind.

- Die Analyse und der Entwurf der Tests für eine Teststufe sollten während der zugehörigen Entwicklungsaktivität beginnen.
- Die Tester sollten im Reviewprozess der Entwicklungsdokumente (Anforderungen, Analyse und Design) eingebunden werden, sobald eine Vorabversion eines der Dokumente verfügbar ist.

Teststufen können in Abhängigkeit des Projekts oder der Systemarchitektur kombiniert oder neu festgelegt (geordnet) werden. Zum Beispiel kann ein Käufer für die Integration kommerzieller Standardsoftware den Integrationstest auf Systemebene (z.B. zur Integration in die Infrastruktur und mit anderen Systemen oder zur Nutzung in der Produktivumgebung) und den Abnahmetest (beispielsweise funktionale und/oder nicht-funktionale Tests und Nutzer- und/oder Betriebstests) durchführen.

2.2 Teststufen (K2) — 40 Minuten

Begriffe
Alpha-Test, Benutzer-Abnahmetest, Beta-Test, Feldtest, funktionale Anforderung, Integration, Integrationstest, Komponententest, nicht-funktionale Anforderung, Platzhalter (stubs), Robustheitstest, Systemtest, testgetriebene Entwicklung, Teststufe, Treiber, Testumgebung

Hintergrund

Alle Teststufen können durch die folgenden Aspekte charakterisiert werden: allgemeine Ziele; die Arbeitsergebnisse, von denen die Testfälle abgeleitet werden (also die Testbasis); das Testobjekt (also was getestet wird); typische Fehlerwirkungen und –zustände, die gefunden werden sollten; Anforderungen an den Testrahmen und Werkzeugunterstützung; spezifische Ansätze und Verantwortlichkeiten.

Das Testen der Konfigurationsdaten eines Systems soll in der Testplanung berücksichtigt werden.

2.2.1 Komponententest (K2)

Testbasis:

- Anforderungen an die Komponente
- detaillierter Entwurf
- Code

Typische Testobjekte:

- Komponenten
- Programme
- Datenumwandlung/Migrationsprogramme
- Datenbankmodule

Der Komponententest (auch bekannt als Unit-, Modul- oder Programmtest) hat zum Ziel, Software, die separat getestet werden kann (z.B. Module, Programme, Objekte, Klassen, etc.), zu prüfen und darin vorhandene Fehler zu finden. In Abhängigkeit von Lebenszyklus und System kann das durch Isolierung vom Rest des Systems erreicht werden. Dabei werden meist Platzhalter, Treiber und Simulatoren eingesetzt.

Der Komponententest kann das Testen funktionaler wie auch nicht-funktionaler Aspekte umfassen, so etwa das Testen der Ressourcenverwendung (z.B. Suche nach Speicherengpässe), Robustheitstest oder auch struktureller Test (z.B. Entscheidungsüberdeckung). Testfälle werden von Entwicklungsdokumenten wie einer Komponentenspezifikation, dem Softwareentwurf oder dem Datenmodell abgeleitet.

Meistens stehen den Testern beim Komponententest der Quellcode wie auch Unterstützung aus der Entwicklungsumgebung zur Verfügung, beispielsweise eine spezielle Komponententestumgebung (Unittest-Framework) oder Debugging-Werkzeuge. In der Praxis sind oft die für den Code verantwortlichen Entwickler an den Komponententests beteiligt. Dabei werden die gefundenen Fehlerzustände häufig sofort korrigiert und gar nicht erst formell behandelt.

Ein Ansatz beim Komponententest ist es, die Testfälle vor der Implementierung der Funktionalität vorzubereiten und zu automatisieren. Dies wird Test-First-Ansatz oder testgetriebene Entwicklung (test-driven) genannt. Dieser Ansatz ist sehr iterativ und basiert auf Zyklen aus Entwicklung von Testfällen, der Entwicklung und Integration von kleinen Code-Stücken und der Ausführung von Komponententests im Wechsel mit der Behebung der Probleme, bis die Tests erfolgreich durchlaufen sind.

2.2.2 Integrationstest (K2)

Testbasis

- Software- und Systementwurf
- Architektur
- Nutzungsabläufe/Workflows
- Anwendungsfälle (use cases)

Typische Testobjekte

- Subsysteme
- Datenbankimplementierungen
- Infrastruktur
- Schnittstellen
- Systemkonfiguration und Konfigurationsdaten

Der Integrationstest prüft die Schnittstellen zwischen Komponenten und die Interaktionen zwischen verschiedenen Teilen eines Systems, beispielsweise zum Betriebssystem, Dateisystem, zur Hardware, und er prüft die Schnittstellen zwischen Systemen.

Es können mehrere Integrationsstufen zum Einsatz gelangen und diese können Testobjekte unterschiedlichster Größe betreffen. Zum Beispiel:

1. Ein Komponentenintegrationstest prüft das Zusammenspiel der Softwarekomponenten und wird nach dem Komponententest durchgeführt.
2. Ein Systemintegrationstest prüft das Zusammenspiel verschiedener Softwaresysteme oder zwischen Hardware und Software und kann nach dem Systemtest durchgeführt werden. In einem solchen Fall hat das Entwicklungsteam oft nur die eine Seite der zu prüfenden Schnittstelle unter seiner Kontrolle. Dies kann als Risiko betrachtet werden. Geschäftsprozesse, die als Workflows implementiert sind, können eine Reihe von Systemen nutzen. Plattformübergreifende Fragestellungen können signifikant sein.

Je größer der Umfang einer Integration ist, desto schwieriger ist die Isolation von Fehlerzuständen in einer spezifischen Komponente oder einem System, was zur Erhöhung des Risikos und zusätzlichem Zeitbedarf zur Fehlerbehebung führen kann.

Systematische Integrationsstrategien können auf der Systemarchitektur (z.B. Top-Down und Bottom-Up), funktionalen Aufgaben, Transaktionsverarbeitungssequenzen oder anderen Aspekten des Systems oder seiner Komponenten basieren. Um die Fehlerisolation zu erleichtern und Fehlerzustände früh aufzudecken, sind inkrementelle Integrationsstrategien normalerweise der Big-Bang-Strategie vorzuziehen.

Das Testen spezieller nicht-funktionaler Eigenschaften (z.B. Performanz) kann im Integrationstest ebenso enthalten sein, wie funktionale Tests.

Bei jeder Integrationsstufe sollen Tester sich ausschließlich auf die eigentliche Integration konzentrieren. Wenn zum Beispiel das Modul A mit dem Modul B integriert wird, so soll der Fokus auf der Kommunikation zwischen den Modulen und nicht etwa auf der Funktionalität der einzelnen Module liegen, welche Inhalt des Komponententests war. Sowohl funktionale als auch strukturelle Ansätze können genutzt werden.

Idealerweise sollten Tester die Architektur verstehen und entsprechend auf die Integrationsplanung Einfluss nehmen. Werden Integrationstests bereits vor der Erstellung der einzelnen Komponenten oder Systeme geplant, können diese dann in der für die Integration effizientesten Reihenfolge entwickelt werden.

2.2.3 Systemtest (K2)

Testbasis

- System- und Anforderungsspezifikation
- Anwendungsfälle (use cases)
- funktionale Spezifikation
- Risikoanalyseberichte

Typische Testobjekte

- System-, Anwender- und Betriebshandbücher
- Systemkonfiguration und Konfigurationsdaten

Der Systemtest beschäftigt sich mit dem Verhalten eines Gesamtsystems/-produkts. Das Testziel soll klar im Master- und/oder Stufentestkonzept dieser Teststufe festgelegt sein.

Beim Systemtest sollte die Testumgebung mit der finalen Ziel- oder Produktivumgebung so weit wie möglich übereinstimmen, um das Risiko umgebungsspezifischer Fehler, die nicht während des Testens gefunden werden, zu minimieren.

Systemtests können Tests einschließen, die auf einer Risikoanalyse basieren und/oder auf Anforderungsspezifikationen, Geschäftsprozessen, Anwendungsfällen oder anderen abstrakten textuellen Beschreibungen oder Modellen des Systemverhaltens, auf Interaktionen mit dem Betriebssystem und den Systemressourcen.

Systemtests sollen funktionale und nicht-funktionale Anforderungen an das System sowie Datenqualitätscharakteristiken untersuchen. Dabei müssen sich Tester auch oft mit unvollständigen oder undokumentierten Anforderungen befassen. Funktionale Anforderungen werden im Systemtest zunächst mit spezifikationsorientierten Testentwurfsverfahren (Black-Box-Testentwurfsverfahren) getestet. Beispielsweise kann eine Entscheidungstabelle für die Kombination der Wirkungen in Geschäftsregeln erstellt werden. Strukturbasierte Verfahren (White-Box-Testentwurfsverfahren) können eingesetzt werden, um die Überdeckung der Tests zu bewerten, bezogen auf ein strukturelles Element wie die Menüstruktur oder die Navigationsstruktur einer Website (siehe Kapitel 4).

Systemtests werden oft durch unabhängige Testteams durchgeführt.

2.2.4 Abnahmetest (K2)

Testbasis:

- Benutzeranforderungen
- Systemanforderungen
- Anwendungsfälle (use cases)
- Geschäftsprozesse
- Risikoanalyseberichte

Typische Testobjekte:

- Geschäftsprozesse des voll integrierten Systems
- Betriebs- und Wartungsprozesse
- Anwenderverfahren
- Formulare
- Berichte
- Konfigurationsdaten

Der Abnahmetest liegt meist im Verantwortungsbereich der Kunden oder Benutzer des Systems. Andere Stakeholder können jedoch auch daran beteiligt sein.

Das Ziel des Abnahmetests besteht darin, Vertrauen in das System, Teilsystem oder in spezifische nicht-funktionale Eigenschaften eines Systems zu gewinnen. Das Finden von Fehlerzuständen ist nicht das Hauptziel beim Abnahmetest. Abnahmetests können die Bereitschaft eines Systems für den Einsatz und die Nutzung bewerten, obwohl sie nicht notwendigerweise die letzte Teststufe darstellen. So könnte beispielsweise ein umfangreicher Systemintegrationstest dem Abnahmetest eines der Systeme folgen.

Die Durchführung von Abnahmetests kann zu verschiedenen Zeiten im Lebenszyklus erfolgen:

- Eine Standardsoftware kann einem Abnahmetest unterzogen werden, wenn sie installiert oder integriert ist.
- Der Abnahmetest bezüglich der Benutzbarkeit einer Komponente kann während des Komponententests durchgeführt werden.
- Der Abnahmetest einer neuen funktionalen Erweiterung kann vor dem Systemtest erfolgen.

Unter anderem gibt es folgende typische Ausprägungen des Abnahmetests:

Anwender-Abnahmetest

Er prüft die Tauglichkeit eines Systems zum Gebrauch durch Anwender bzw. Kunden.

Betrieblicher Abnahmetest

Die Abnahme des Systems durch den Systemadministrator enthält:

- Test des Erstellens und Wiedereinspielens von Sicherungskopien (backup/ restore)
- Wiederherstellbarkeit nach Ausfällen
- Benutzermanagement
- Wartungsaufgaben
- Datenlade- u. Migrationsaufgaben und
- periodische Überprüfungen von Sicherheitslücken

Regulatorischer und vertraglicher Abnahmetest

Beim vertraglichen Abnahmetest wird kundenindividuelle Software explizit gegen die vertraglichen Abnahmekriterien geprüft. Abnahmekriterien sollten beim Vertragsabschluss zwischen den beteiligten Parteien definiert werden.
Regulatorische Abnahmetests werden gegen alle Gesetze und Standards durchgeführt, denen das System entsprechen muss – beispielsweise staatliche, gesetzliche oder Sicherheitsbestimmungen.

Alpha- und Beta-Test (oder Feldtest)

Hersteller kommerzieller oder Standardsoftware wollen oft Feedback von potenziellen oder existierenden Kunden erhalten, bevor sie ein Produkt kommerziell zum Kauf anbieten. Der Alpha-Test wird am Herstellerstandort durchgeführt, nicht jedoch vom Entwicklungsteam. Der Beta- (oder Feldtest) wird von Kunden oder potenziellen Kunden an den Kundenstandorten durchgeführt.

Organisationen können ebenso andere Begriffe nutzen z.B. Fabrikabnahmetests oder Kundenakzeptanztests für Systeme, die getestet werden, bevor oder nachdem sie zum Einsatzort eines Kunden gebracht wurden.

2.3 Testarten (K2)	40 Minuten

Begriffe
Benutzbarkeitstest, Black-Box-Test, Codeüberdeckung, funktionaler Test, Interoperabilitätstest, Lasttest, Performanztest, Portabilitätstest, Sicherheitstest, Stresstest, struktureller Test, Wartbarkeitstest, White-Box-Test, Zuverlässigkeitstest

Hintergrund

Eine Menge von Testaktivitäten kann darauf ausgerichtet sein, das Softwaresystem (oder ein Teilsystem) entweder aus einem bestimmten Grund bzw. Anlass oder aber mit einem bestimmten Testziel zu prüfen.

Eine Testart ist auf ein spezielles Testziel ausgerichtet:

- Testen einer zu erfüllenden Funktion
- Testen einer nicht-funktionalen Anforderung, wie Zuverlässigkeit oder Benutzbarkeit
- Testen der Struktur oder Architektur der Software beziehungsweise des Systems
- Prüfen auf erfolgreiche Beseitigung eines Fehlers (Nachtest) oder Prüfen auf unbeabsichtigte beziehungsweise ungewollte Änderungen oder Seiteneffekte (Regressionstest)

Ein Modell der zu testenden Software kann sowohl für strukturelle Tests, für nicht-funktionale Tests als auch für funktionale Tests entwickelt und/oder eingesetzt werden, zum Beispiel ein Kontrollflussgraph oder Menüstrukturmodell für strukturelle Tests, ein Prozessablaufmodell, ein Zustandsübergangsmodell, oder eine Klartextspezifikation für das funktionale Testen oder ein Performanzmodell, ein Benutzbarkeitsmodell oder ein Modell zum Thema Sicherheitsbedrohung (security threat modeling) für nicht-funktionale Tests.

2.3.1 Testen der Funktionalität (funktionaler Test) (K2)

Die Funktionalität, die ein System, Teilsystem oder eine Komponente zu erbringen hat, kann entweder in Arbeitsprodukten wie Anforderungsspezifikationen, Anwendungsfällen oder in einer funktionalen Spezifikation beschrieben sein, oder sie kann undokumentiert sein. Die Funktionalität besagt, „was“ das System leistet.

Funktionale Tests basieren auf Funktionen, Eigenschaften (beschrieben in Entwicklungsdokumenten oder gemäß dem Verständnis der Tester) und ihrer Interoperabilität zu bestimmten Systemen. Sie kommen in allen Teststufen zur Anwendung (z.B. Komponententest basierend auf der Komponentenspezifikation).

Spezifikationsorientierte Testentwurfsverfahren werden verwendet, um Testbedingungen und Testfälle aus der Funktionalität der Software oder des Systems herzuleiten (siehe Kapitel 4). Ein funktionaler Test betrachtet das von außen sichtbare Verhalten der Software (Black-Box-Test).

Ein Typ des funktionalen Tests, der Sicherheitstest, prüft, ob Funktionen, die Software und Daten schützen sollen (z.B. Firewalls), wirksam gegenüber externen Bedrohungen, wie Viren etc., sind. Ein anderer Typ des funktionalen Tests ist der Interoperabilitätstest. Er bewertet die Fähigkeit des Softwareprodukts mit ein oder mehr spezifizierten Komponenten oder Systemen zu interagieren.

2.3.2 Testen der nicht-funktionalen Softwaremerkmale (nicht-funktionaler Test) (K2)

Nicht-funktionales Testen umfasst unter anderem: Performanztest, Lasttest, Stresstest, Benutzbarkeitstest, Wartbarkeitstest, Zuverlässigkeitstest und Portabilitätstest. Es geht darum, „wie“ das System arbeitet.

Nicht-funktionales Testen kann in allen Teststufen zur Anwendung kommen. Der Begriff nicht-funktionaler Test bezeichnet Testarten, die zur Prüfung von Software- und Systemmerkmalen ver-

wendet werden. Zur Quantifizierung dieser Merkmale werden unterschiedliche Maßstäbe eingesetzt, so zum Beispiel Antwortzeiten beim Performanztest. Diese Testarten können sich auf ein Qualitätsmodell stützen, wie zum Beispiel auf die Softwarequalitätsmerkmale, die in der Norm 'Software engineering – Productquality' (ISO 9126-1, 2001) definiert sind. Nicht-funktionale Tests betrachten das nach außen sichtbare Verhalten. Sie verwenden meistens Black-Box-Testentwurfsverfahren, um diese Prüfung durchzuführen**.**

2.3.3 Testen der Softwarestruktur/Softwarearchitektur (strukturbasierter Test) (K2)

Strukturelles Testen (White-Box-Test) kann in allen Teststufen angewandt werden. Strukturelle Testentwurfsverfahren werden am besten nach den spezifikationsorientierten Testentwurfsverfahren eingesetzt, um die Testintensität anhand der gemessenen Abdeckungen zu beurteilen.

Testüberdeckung ist ein Maß dafür, inwiefern eine Struktur durch eine Testsuite geprüft bzw. ausgeführt (überdeckt) wurde. Dabei wird jeweils der prozentuale Anteil der überdeckten Strukturelemente angegeben. Ist die erreichte Testüberdeckung kleiner als 100%, kann diese verbessert werden, indem für die noch nicht überdeckten Elemente zusätzliche Testfälle spezifiziert werden. Entsprechende Testentwurfsverfahren sind im Kapitel 4 beschrieben.

Werkzeuge zur Messung der Codeüberdeckung, wie Anweisungs- oder Entscheidungsüberdeckung, können in allen Teststufen eingesetzt werden, im Speziellen aber im Komponenten- und Komponentenintegrationstest. Strukturelles Testen kann auch auf der Systemarchitektur aufbauen, so zum Beispiel auf der Aufrufhierarchie.

Der Ansatz des strukturbasierten Testens kann sinngemäß ebenso in den Teststufen System-, Systemintegration- oder Abnahmetest eingesetzt werden (beispielsweise für Geschäftsmodelle oder Menüstrukturen).

2.3.4 Testen im Zusammenhang mit Änderungen (Fehlernachtest und Regressionstest) (K2)

Nachdem ein Fehlerzustand entdeckt und korrigiert wurde, sollte die Software anschließend erneut getestet werden, um zu bestätigen, dass der Fehlerzustand erfolgreich entfernt wurde. In diesem Fall spricht man von einem Fehlernachtest. Debugging (Lokalisieren, und Entfernen eines Fehlerzustands) ist eine Entwicklungs- und keine Testaufgabe.

Unter Regressionstest verstehen wir das wiederholte Testen eines bereits getesteten Programms nach dessen Modifikation. Er hat das Ziel nachzuweisen, dass durch die vorgenommenen Modifikationen keine Fehlerzustände eingebaut wurden oder bisher unentdeckte Fehlerzustände wirksam werden. Diese Fehlerzustände können entweder in der zu testenden Software selbst oder aber in einer anderen Softwarekomponente liegen. Ein Regressionstest wird ausgeführt, wenn sich die Software selbst oder ihre Umgebung ändert. Der Umfang des Regressionstests ist durch das Risiko von neuen Fehlerzuständen in der vorher funktionierenden Software bestimmt.

Tests, die für Fehlernachtests oder Regressionstests vorgesehen sind, müssen wiederholbar sein.

Regressionstests können in allen Teststufen durchgeführt werden und enthalten funktionale, nicht-funktionale wie auch strukturelle Tests. Regressionstests werden oft wiederholt und ändern sich im Normalfall eher selten, was sie zu bevorzugten Kandidaten für eine Automatisierung macht.

2.4 Wartungstest (K2) — 15 Minuten

Begriffe
Auswirkungsanalyse, Wartungstest

Hintergrund

Einmal in Betrieb, bleibt ein Softwaresystem oft über Jahre oder Jahrzehnte im Einsatz. Während dieser Zeit werden das System, seine Konfigurationsdaten und seine Umgebung mehrmals korrigiert, gewechselt oder erweitert. Eine vorausschauende Release-Planung ist entscheidend für einen erfolgreichen Wartungstest, hierbei muss eine Unterscheidung zwischen geplanten Releases und „Hot Fixes“ gemacht werden. Der Wartungstest wird an einem betreibbaren System ausgeführt, wobei die Tests jeweils durch Modifikationen, Migrationen oder Einzug der Software (Außerbetriebnahme, Ablösung) oder des Systems bedingt werden.

Modifikationen enthalten geplante Erweiterungen (z.B. basierend auf dem geplanten Release), Korrekturen und Notfallkorrekturen, Umgebungsänderungen wie eine geplante Aktualisierung des Betriebs- oder Datenbanksystems, geplante Upgrades von kommerzieller Standardsoftware oder Patches zur Korrektur erst kürzlich entdeckter Schwachstellen des Betriebssystems.

Ein Wartungstest bei einer Migration (z.B. von einer Plattform zu einer anderen) soll sowohl Tests im Betrieb der neuen Umgebung als auch der geänderten Software umfassen. Migrationstests (Konvertierungstests) werden ebenso benötigt, wenn Daten aus einer anderen Anwendung in das zu wartende System migriert werden.

Ein Wartungstest bei Außerbetriebnahme/ Einzug oder der Ablösung eines Systems kann das Testen der Datenmigration oder der Archivierung umfassen (falls eine lange Aufbewahrungszeit notwendig ist).

Zusätzlich zum Testen der eingebrachten Modifikationen gehört zum Wartungstesten das Regressionstesten der Systemteile, die nicht geändert wurden. Der Umfang der Wartungstests ist abhängig vom dem mit der Änderung verbundenen Risiko, der Größe des existierenden Systems und dem Umfang der Änderung. Wartungstests können in Abhängigkeit von Modifikationen in allen Teststufen und für alle Testarten durchgeführt werden.

Die Ermittlung, inwiefern ein bestehendes System durch Modifikationen beeinflusst wird, nennt man Auswirkungsanalyse. Sie wird verwendet, um zu entscheiden, wie viele Regressionstests durchzuführen sind. Die Auswirkungsanalyse kann benutzt werden, um eine Regressionstestsuite festzulegen.

Bei veralteten oder gar fehlenden Spezifikationen oder wenn keine Tester mit entsprechendem Fachwissen verfügbar sind, kann sich der Wartungstest sehr schwierig gestalten.

Referenzen
2 Linz, 2012
2.1.3 CMMI, 2004, Craig, 2002, Hetzel, 1988, IEEE 12207, 2008
2.2 Hetzel, 1988
2.2.4 Copeland, 2004, Myers 2001
2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004
2.3.2 Black, 2001, ISO 9126-1, 2001
2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1988
2.3.4 Hetzel, 1988, IEEE STD 829 – 1998
2.4 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE STD 829 – 1998

3 Statischer Test (K2) — 60 Minuten

Lernziele für den Abschnitt statischer Test

Die Lernziele legen fest, was Sie nach Beenden des jeweiligen Moduls gelernt haben sollten.

3.1 Statische Prüftechniken und der Testprozess (K2)

LO-3.1.1 Arbeitsergebnisse der Softwareentwicklung, die mit den verschiedenen statischen Prüftechniken geprüft werden, erkennen. (K1)

LO-3.1.2 Bedeutung und Nutzen statischer Methoden für die Bewertung von Arbeitsergebnissen der Softwareentwicklung beschreiben können. (K2)

LO-3.1.3 Unterschiede zwischen statischen und dynamischen Techniken erklären können, wobei Ziele und zu identifizierende Fehlerarten sowie die Rolle dieser Techniken im Softwarelebenszyklus zu beachten sind. (K2)

3.2Reviewprozess (K2)

LO-3.2.1 Aktivitäten, Rollen und Verantwortlichkeiten eines typischen formalen Reviews wiedergeben können. (K1)

LO-3.2.2 Unterschiede zwischen den verschiedenen Reviewarten (informelles Review, technisches Review, Walkthrough und Inspektion) erklären können. (K2)

LO-3.2.3 Faktoren für die erfolgreiche Durchführung eines Reviews erklären können. (K2)

3.3 Werkzeuggestützte statische Analyse (K2)

LO-3.3.1 Typische Fehlerzustände und Fehler wiedergeben können, die durch eine statische Analyse identifiziert werden können, und sie mit Reviews und dynamischen Tests vergleichen. (K1)

LO-3.3.2 Den typischen Nutzen der statischen Analyse anhand von Beispielen beschreiben können. (K2)

LO-3.3.3 Typische Fehlerzustände im Quellcode und Entwurf, die durch eine werkzeuggestützte statische Analyse identifiziert werden können, auflisten. (K1)

3.1 Statische Prüftechniken und der Testprozess (K2)	**15 Minuten**

Begriffe
Dynamisches Testen, statischer Test

Hintergrund

Anders als das dynamische Testen, welches die Ausführung der Software voraussetzt, beruhen statische Prüftechniken auf der „manuellen“ Überprüfung (Reviews) oder automatisierten Analysen (statische Analyse) des Codes oder anderer Projektdokumentation, ohne den Programmcode auszuführen.

Reviews sind eine Möglichkeit, Arbeitsergebnisse der Softwareentwicklung (einschließlich Code) zu prüfen und können problemlos bereits lange vor der dynamischen Testdurchführung durchgeführt werden. Fehlerzustände, die durch Reviews in den frühen Phasen des Softwarelebenszyklus entdeckt werden (z.B. Fehlerzustände in den Anforderungen), sind häufig bedeutend kostengünstiger zu beheben, als solche, die erst während der Testdurchführung bei Ausführung des Programmcodes gefunden werden.

Ein Review kann komplett als manuelle Aktivität durchgeführt, aber ebenso durch Werkzeuge unterstützt werden. Die wichtigste manuelle Tätigkeit ist die Prüfung und Kommentierung des Arbeitsergebnisses. Jedes Arbeitsergebnis der Softwareentwicklung kann einem Review unterzogen werden, einschließlich Anforderungsspezifikationen, Designspezifikationen, Quellcode, Testkonzepte, Testspezifikationen, Testfälle, Testskripte, Anwenderhandbücher oder Web-Seiten.

Vorteile von Reviews sind frühe Aufdeckung und Korrektur von Fehlerzuständen, Verbesserung der Softwareentwicklungsproduktivität, reduzierte Entwicklungsdauer, reduzierte Testkosten und -dauer, Reduzierung der Kosten während der Lebensdauer, weniger Fehlerzustände und verbesserte Kommunikation. Reviews können beispielsweise Auslassungen in Anforderungen aufdecken (z.B. fehlende Funktionen), die durch einen dynamischen Test vermutlich nicht gefunden würden.

Reviews, statische Analyse und dynamischer Test haben das gleiche Ziel, nämlich Fehlerzustände zu identifizieren. Sie ergänzen sich: Die verschiedenen Methoden können verschiedene Arten von Fehlern wirksam und effizient aufdecken. Verglichen mit dem dynamischen Test finden statische Prüftechniken eher Ursachen der Fehlerwirkungen (Fehlerzustände) als Fehlerwirkungen selbst.

Zu den typische Fehlerzuständen, die effektiver und effizienter durch Reviews als durch dynamische Tests zu finden sind, gehören: Abweichungen von Standards, Fehlerzustände in Anforderungen, Fehlerzustände im Design, unzureichende Wartbarkeit und fehlerhafte Schnittstellenspezifikationen.

3.2 Reviewprozess (K2)	**25 Minuten**

Begriffe
Eingangskriterien, formales Review, Gutachter, informelles Review, Inspektion, Metrik, Moderator, Peer Review, Protokollant, technisches Review, Walkthrough

Hintergrund

Die verschiedenen Arten von Reviews variieren zwischen informell, charakterisiert durch fehlende schriftlichen Vorgaben für die Gutachter, und systematisch, charakterisiert durch die Einbindung eines Reviewteams, dokumentierte Reviewergebnisse und dokumentierte Abläufe zur Durchführung von Reviews. Der Formalismus eines Reviewprozesses ist abhängig von Faktoren wie Reife des Entwicklungsprozesses, gesetzlichen oder regulatorischen Anforderungen oder der Notwendigkeit eines Prüfnachweises.

Die Art und Weise, wie ein Review durchgeführt wird, ist abhängig von den festgelegten Zielen des Reviews (z.B. Finden von Fehlerzuständen, Erwerben von Verständnis, Ausbildung von Testern und neuen Teammitgliedern oder einer Diskussion und Entscheidungsfindung im Konsens).

3.2.1 Aktivitäten eines formalen Reviews (K1)

Ein typisches formales Review besteht aus folgenden Hauptaktivitäten:

1. **Planen**
   - Festlegen von Review-/Prüfkriterien
   - Auswahl der beteiligten Personen
   - Besetzen der Rollen
   - Festlegen der Eingangs- und Endekriterien bei mehr formalen Reviewarten (z.B. Inspektion)
   - Auswahl der zu prüfenden Dokumententeile
   - Prüfen der Eingangskriterien (bei formaleren Reviewarten)
2. **Kick-off**
   - Verteilen der Dokumente
   - Erläutern der Ziele, des Prozesses und der Dokumente den Teilnehmern gegenüber
3. **Individuelle Vorbereitung**
   - Vorbereiten der Reviewsitzung durch Prüfung des/der Dokuments/e
   - Notieren von potenziellen Fehlerzuständen, Fragen und Kommentaren
4. **Prüfen/Bewerten/Festhalten der Ergebnisse (Reviewsitzung)**
   - Diskussion oder Protokollierung, mit dokumentierten Ergebnissen oder Protokollen (bei formaleren Reviewarten)
   - Festhalten von Fehlerzuständen, Empfehlungen zum Umgang mit ihnen oder Entscheidungen über die Fehlerzustände
   - Prüfen/Bewerten und Protokollieren von Problemen während eines physischen Treffens oder Nachverfolgen von elektronischer Gruppenkommunikation

5. **Überarbeiten**
   - Beheben der gefundenen Fehlerzustände typischerweise durch den Autor
   - Protokollieren des aktualisierten Fehlerstatus (in formalen Reviews)
6. **Nachbereiten**
   - Prüfen, ob Fehlerzustände zugewiesen wurden
   - Sammeln von Metriken
   - Prüfen von Endekriterien (bei formaleren Reviewarten)

3.2.2 Rollen und Verantwortlichkeiten (K1)

Bei einem typischen formalen Review findet man folgende Rollen:

- **Manager**: Die Person, die über die Durchführung von Reviews entscheidet, Zeit im Projektplan zur Verfügung stellt und überprüft, ob die Reviewziele erfüllt sind.
- **Moderator**: Die Person, die das Review eines Dokuments bzw. von einigen zusammengehörenden Dokumenten leitet, einschließlich der Reviewplanung, der Leitung der Sitzung und der Nachbereitung nach der Sitzung. Falls nötig, kann der Moderator zwischen den verschiedenen Standpunkten vermitteln. Er ist häufig die Person, von der der Erfolg des Reviews abhängt.
- **Autor**: Der Verfasser oder die Person, die für das/die zu prüfende/n Dokument/e hauptverantwortlich ist.
- **Gutachter**: Personen mit einem spezifischen technischen oder fachlichen Hintergrund (auch Prüfer oder Inspektoren genannt), die nach der nötigen Vorbereitung im Prüfobjekt Befunde identifizieren und beschreiben (z.B. Fehlerzustände). Gutachter sollten so gewählt werden, dass verschiedene Sichten und Rollen im Reviewprozess vertreten sind. Sie sollten an allen Reviewsitzungen teilnehmen können.
- **Protokollant**: Die Person, die alle Ergebnisse, Probleme und offenen Punkte dokumentiert, die im Verlauf der Sitzung identifiziert werden.

Softwareprodukte oder darauf bezogene Arbeitsergebnisse aus verschiedenen Perspektiven zu betrachten und Checklisten zu nutzen, kann Reviews wirksamer und effizienter machen. So kann eine Checkliste helfen, bisher unentdeckte Probleme aufzudecken, wenn sie typische Anforderungsprobleme enthält und unterschiedliche Perspektiven einnimmt, beispielsweise vom Benutzer, Wartungspersonal, Tester oder Operator.

3.2.3 Reviewarten (K2)

Ein einzelnes Softwareprodukt oder ein darauf bezogenes Arbeitsergebnis kann Gegenstand von mehr als einem Review sein. Falls mehr als nur eine Reviewart eingesetzt wird, kann die Reihenfolge variieren. Ein informelles Review beispielsweise könnte vor einem technischen Review durchgeführt werden oder eine Inspektion einer Anforderungsspezifikation kann vor einem Walkthrough mit Kunden durchgeführt werden. Die Hauptcharakteristika, optionale Bestandteile und Zwecke allgemeiner Reviewarten sind:

Informelles Review

- kein formaler Prozess
- kann in Form des Programmierens in Paaren (pair programming) durchgeführt werden oder ein technischer Experte unterzieht Entwurf und Quellcode einem Review
- Ergebnisse können dokumentiert werden
- Nutzen variiert abhängig von den Gutachtern
- Hauptzweck: Günstiger Weg eine Verbesserung zu erreichen

Walkthrough

- Sitzung geleitet durch den Autor
- kann in Form von Szenarien, Probeläufen oder im Kreis gleichgestellter Mitarbeiter (Peer Review) stattfinden
- Open-End-Sitzungen
  - wahlweise der Sitzung vorausgehende Vorbereitung der Gutachter
  - wahlweise Vorbereitung eines Reviewberichts, der eine Liste der Befunde enthält
- wahlweise Protokollant (der aber nicht der Autor ist)
- kann in der Praxis von informell bis sehr formal variieren
- Hauptzweck: Lernen, Verständnis erzielen, Fehlerzustände finden

Technisches Review

- dokumentierter und definierter Fehlerfindungsprozess, der gleichgestellte Mitarbeiter und technische Experten sowie optional Personen aus dem Management einschließt
- kann als Peer Review ohne Teilnahme des Managements ausgeführt werden
- idealerweise durch einen geschulten Moderator geleitet (nicht der Autor)
- Vorbereitung vor der Sitzung durch Gutachter
- wahlweise Nutzung von Checklisten
- Vorbereitung eines Reviewberichts, der folgende Punkte enthält: die Liste der Befunde, eine Gesamtbewertung, inwieweit das Softwareprodukt die Anforderungen erfüllt, und Empfehlungen in Bezug auf die Befunde, wo angebracht
- kann in der Praxis von informell bis sehr formal variieren
- Hauptzweck: Diskussion, Entscheidungen treffen, Alternativen bewerten, Fehlerzustände finden, technische Probleme lösen und prüfen, ob Übereinstimmung mit Spezifikationen, Plänen, Bestimmungen und Standards existiert

Inspektion

- geleitet durch einen geschulten Moderator (nicht der Autor)
- gewöhnlich durchgeführt als Prüfung durch gleichgestellte Mitarbeiter
- definierte Rollen
- schließt das Sammeln von Metriken ein
- formaler Prozess basierend auf Regeln und Checklisten
- spezifizierte Eingangs- und Endekriterien für die Abnahme des Softwareprodukts
- Vorbereitung vor der Sitzung
- Inspektionsbericht mit Liste der Befunde
- formaler Prozess für Folgeaktivitäten (optional mit Komponenten zur Prozessverbesserung)
- wahlweise Vorleser
- Hauptzweck: Fehlerzustände finden

Walkthroughs, technische Reviews und Inspektionen können in einer Gruppe von gleichgestellten Kollegen (Peers), d.h. Kollegen aus der gleichen organisatorischen Ebene, durchgeführt werden. Diese Art von Review wird auch „Peer Review“ genannt.

3.2.4 Erfolgsfaktoren für Reviews (K2)

Erfolgsfaktoren für Reviews beinhalten:

- Jedes Review hat klar definierte Ziele.
- Abhängig von den Reviewzielen werden geeignete Personen ausgewählt.
- Tester sind geschätzte Gutachter, die einen Beitrag zum Review leisten. Weiterhin lernen sie auch das Produkt kennen, was sie befähigt Tests früher vorzubereiten.
- Gefundene Fehlerzustände werden positiv aufgenommen und werden objektiv zur Sprache gebracht.
- Menschliche und psychologische Aspekte werden beachtet, es beispielsweise als eine positive Erfahrung für den Autor zu gestalten.

- Das Review wird in einer Atmosphäre des Vertrauens durchgeführt, das Ergebnis dient nicht zur Beurteilung der Teilnehmer.
- Es werden die Reviewtechniken angewendet, die zur Erreichung der Reviewziele, für Art und Stufe von Arbeitsergebnissen der Softwareentwicklung und für die Gutachter geeignet sind.
- Wenn sie geeignet sind, die Effektivität der Fehleridentifikation zu steigern, werden Checklisten oder Rollen verwendet.
- Es finden Schulungen in Reviewtechniken statt, besonders für die formaleren Methoden wie Inspektionen.
- Das Management unterstützt einen guten Reviewprozess, indem es beispielsweise angemessene Zeit für Reviewaktivitäten im Projektplan einräumt.
- Es liegt eine Betonung auf Lernen und Prozessverbesserung.

3.3 Werkzeuggestützte statische Analyse(K2) — 20 Minuten

Begriffe
Compiler, Datenfluss, Komplexität, Kontrollfluss, statische Analyse

Hintergrund

Das Ziel der statischen Analyse ist es, Fehlerzustände in Softwarequellcode und in den Softwaremodellen zu finden. Statische Analyse wird durchgeführt, ohne dass die untersuchte Software tatsächlich durch das Werkzeug ausgeführt wird; dynamischer Test führt Softwarecode aus. Statische Analyse kann Fehlerzustände lokalisieren, die durch dynamisches Testen schwer zu finden sind. Ebenso wie Reviews findet die statische Analyse Fehlerzustände eher als Fehlerwirkungen. Statische Analysewerkzeuge analysieren Programmcode (z.B. Kontrollfluss und Datenfluss), ebenso wie generierte HTML- und XML-Ausgaben.

Vorteile der statischen Analyse:

- frühes Erkennen von Fehlerzuständen vor der Testdurchführung
- frühe Warnung vor verdächtigen Aspekten in Code oder Design wie hohes Komplexitätsmaß durch Berechnen von Metriken
- Identifizieren von Fehlerzuständen, die durch dynamischen Test nicht effektiv und effizient aufzudecken sind
- Aufdecken von Abhängigkeiten und Inkonsistenzen in Softwaremodellen, beispielsweise tote Links
- verbesserte Wartbarkeit von Code und Design
- Vorbeugen von Fehlerzuständen, wenn sich das aus Erfahrung Gelernte in der Entwicklung niederschlägt

Typische Fehlerzustände, die durch eine werkzeuggestützte statische Analyse gefunden werden können:

- Referenzierung einer Variablen mit nicht definiertem Wert
- inkonsistente Schnittstellen zwischen Modulen und Komponenten
- Variablen, die nicht verwendet oder nicht korrekt deklariert werden
- unerreichbarer (toter) Code
- fehlende oder falsche Logik (mögliche Endlosschleifen)
- übermäßig komplizierte Konstrukte
- Verletzung von Programmierkonventionen
- Sicherheitsschwachstellen
- Syntax-Verletzungen von Code und Softwaremodellen

Werkzeuge für statische Analysen werden typischerweise entwicklungsbegleitend und vor Komponenten- und Integrationstests oder beim Einchecken von Code in Konfigurationsmanagementwerkzeuge genutzt (Prüfen gegen vordefinierte Regeln oder Programmierstandards), und durch Designer während der Softwaremodellierung. Werkzeuge für statische Analysen können große Mengen von Warnungen und Hinweisen erzeugen, die gut verwaltet werden müssen, um eine effektive Nutzung des Werkzeugs zu erlauben.

Compiler können auch eine gute Unterstützung für eine statische Analyse bieten, u.a. durch Berechnen von Metriken.

Referenzen
3 Linz, 2012
3.2 IEEE 1028-2008
3.2.2 Gilb, 1993, van Veenendaal, 2004
3.2.4 Gilb, 1993, IEEE 1028
3.3 van Veenendaal, 2004

4 Testentwurfsverfahren (K4) — 285 Minuten

Lernziele für den Abschnitt Testentwurfsverfahren

Die Lernziele legen fest, was Sie nach Beenden des jeweiligen Moduls gelernt haben sollten.

4.1 Der Testentwicklungsprozess (K3)

LO-4.1.1 Unterscheiden können, zwischen Testentwurfsspezifikation, Testfallspezifikation und Testablaufspezifikation. (K2)

LO-4.1.2 Begriffe Testbedingung, Testfall und Testablauf gegenüberstellen können. (K2)

LO-4.1.3 Qualität von Testfällen bewerten können, bezüglich:
- o eindeutige Rückverfolgbarkeit zu den Anforderungen
- o Sollverhalten (K2)

LO-4.1.4 Testfälle in eine wohlstrukturierte Testablaufspezifikation übersetzen können – mit einem Detaillierungsgrad, der den Vorkenntnissen der Tester angepasst ist. (K3)

4.2 Kategorien von Testentwurfsverfahren (K2)

LO-4.2.1 Gründe wiedergeben können, dass sowohl spezifikationsorientierte (Black-Box) als auch strukturbasierte (White-Box) Testentwurfsverfahren von Nutzen sind, und für beides gängige Verfahren aufzählen können. (K1)

LO-4.2.2 Eigenschaften, Gemeinsamkeiten und Unterschiede zwischen spezifikationsorientiertem Testen, strukturbasiertem Testen und erfahrungsbasiertem Testen erklären können. (K2)

4.3 Spezifikationsorientierte oder Black-Box-Verfahren (K3)

LO-4.3.1 Testfälle anhand unterschiedlicher Softwaremodelle schreiben können, unter Verwendung von Äquivalenzklassenbildung, Grenzwertanalyse, Entscheidungstabellen und Zustandsübergangsdiagrammen und –tabellen. (K3)

LO-4.3.2 Hauptziele der fünf Testverfahren erklären können, auf welchen Ebenen und in welchem Testumfeld das Verfahren eingesetzt und wie jeweils der Überdeckungsgrad gemessen werden kann. (K2)

LO-4.3.3 Das Konzept der anwendungsfallbasierten (use case) Tests und der damit verbundenen Vorteile erklären können. (K2)

4.4 Strukturbasierte oder White-Box-Verfahren (K4)

LO-4.4.1 Das Konzept und den Wert von Codeüberdeckung beschreiben können. (K2)

LO-4.4.2 Die Konzepte von Anweisungs- und Entscheidungsüberdeckung erklären können und Gründe nennen können, warum diese Überdeckungsmaße auch auf anderen Teststufen als im Komponententest eingesetzt werden können (z.B. bei Geschäftsprozessen auf Systemebene). (K2)

LO-4.4.3 Testfälle zu vorgegebenen Kontrollflüssen schreiben können unter Verwendung von Anweisungs- und Entscheidungsüberdeckungsverfahren. (K3)

LO-4.4.4 Vollständigkeit von Anweisungs- und Entscheidungsüberdeckung in Bezug auf definierte Endekriterien überprüfen können. (K4)

4.5 Erfahrungsbasierte Verfahren (K2)

LO-4.5.1 Gründe wiedergeben können, warum Testfälle auf der Grundlage von Intuition, Erfahrung und Wissen über typische Fehlerzustände geschrieben werden sollten. (K1)

LO-4.5.2 Erfahrungsbasierte Verfahren mit spezifikationsorientierten Testverfahren vergleichen können. (K2)

4.6 Auswahl von Testverfahren (K2)

LO-4.6.1 Testentwurfsverfahren in einem vorgegebenen Zusammenhang gemäß ihrer Tauglichkeit, für die Testbasis und bezüglich Modellen und Softwaremerkmalen klassifizieren können. (K2)

4.1 Der Testentwicklungsprozess (K3) — 15 Minuten

Begriffe
Rückverfolgbarkeit, Testablaufspezifikation, Testausführungsplan, Testentwurf, Testfallspezifikation, Testskript

Hintergrund
Der Testentwicklungsprozess, der in diesem Kapitel beschrieben wird, kann auf verschiedene Weise durchgeführt werden, von sehr informell – mit wenig oder keiner Dokumentation – bis sehr formal (wie in diesem Abschnitt weiter unten beschrieben). Der Grad der Formalisierung hängt vom Kontext des Testens ab, einschließlich der Reife des Testprozesses und des Entwicklungsprozesses, zeitlicher Beschränkungen, Sicherheits- oder regulatorischer Anforderungen und der einbezogenen Personen.

Während der Testanalyse werden die dem Test zugrunde liegenden Dokumente analysiert, um festzustellen, was getestet werden muss, d.h. die Testbedingungen festzulegen. Eine Testbedingung ist definiert als eine Einheit oder ein Ereignis, z.B. eine Funktion, eine Transaktion, ein Qualitätsmerkmal oder ein strukturelles Element, das durch einen oder mehrere Testfälle verifiziert werden kann.

Indem Testbedingungen auf Spezifikationen und Anforderungen zurückgeführt werden, erlauben sie sowohl eine effektive Auswirkungsanalyse (impact analysis) bei geänderten Anforderungen als auch die Bestimmung einer Anforderungsüberdeckung, bezogen auf eine bestimmte Menge von Tests. Während der Testanalyse wird eine detaillierte Testvorgehensweise – neben anderen Gesichtspunkten – auf Grundlage von den festgestellten Risiken angewendet, um benötigte Testentwurfsverfahren auszuwählen (siehe Kapitel 5 zum Thema Risikoanalyse).

Während des Testentwurfs werden die Testfälle und Testdaten definiert und dokumentiert. Ein Testfall besteht aus einer Menge von Eingabewerten, den für die Ausführung notwendigen Vorbedingungen, der Menge der vorausgesagten Ergebnisse und den erwarteten Nachbedingungen, definiert mit dem Ziel, ein oder mehrere Testziele oder Testbedingungen abzudecken. Der IEEE Standard 829-1998 (‘Standard for Software Test Documentation’) beschreibt die Inhalte von Testentwurfsspezifikationen (inkl. Testbedingungen) und Testfallspezifikationen.

Erwartete Ergebnisse sollten im Rahmen der Spezifikation eines Testfalls ermittelt werden, und Ausgaben, Änderungen von Daten und Zuständen sowie alle anderen Folgen des Tests enthalten. Falls die erwarteten Ergebnisse nicht definiert wurden, könnte ein plausibles, aber fehlerhaftes Ergebnis fälschlicherweise als richtig angesehen werden. Erwartete Ergebnisse sollten idealerweise vor der Testdurchführung festgelegt werden.

Während der Testimplementierung werden die Testfälle entwickelt, implementiert, priorisiert und in einer Testablaufspezifikation (Drehbuch oder Testskript) zusammengefasst (IEEE STD 829-1998). Das Testdrehbuch legt die Reihenfolge der Aktionen für die Ausführung eines Tests fest. Wenn Tests unter Verwendung eines Testausführungswerkzeugs durchgeführt werden, ist die Reihenfolge der Aktionen in einem Testskript festgelegt (in einem automatisierten Testszenario).

Die verschiedenen manuellen und automatisierten Testskripte werden anschließend in einem Testausführungsplan zusammengestellt, der die Reihenfolge festlegt, in der die verschiedenen Testszenarios (und gegebenenfalls die automatisierten Testskripte) ausgeführt werden. Der Testausführungsplan berücksichtigt Faktoren wie Regressionstests, Priorisierung und logische Abhängigkeiten.

4.2 Kategorien von Testentwurfsverfahren (K2) — 15 Minuten

Begriffe
Black-Box-Testentwurfsverfahren, erfahrungsbasierte Testentwurfsverfahren, Testentwurfsverfahren, White-Box-Testentwurfsverfahren

Hintergrund

Das Ziel von Testentwurfsverfahren ist es, Testbedingungen, Testfälle und Testdaten zu ermitteln.

Ein klassischer Ansatz unterscheidet Testentwurfsverfahren in Black-Box- oder White-Box-Verfahren.

Black-Box-Verfahren (die auch spezifikationsorientierte Verfahren genannt werden) stellen einen Weg dar, Testbedingungen, Testfälle oder Testdaten für eine Komponente oder ein System aufgrund der Analyse der zugrunde liegenden Dokumentation der Testbasis abzuleiten und auszuwählen. Dies umfasst funktionales und nicht-funktionales Testen. Black-Box-Testverfahren verwenden per Definition keine Informationen über die interne Struktur der Komponente oder des Systems, welches zu testen ist. White-Box-Verfahren (auch strukturelle oder strukturbasierte Verfahren) stützen sich auf eine Analyse der Struktur einer Komponente oder eines Systems. Black-Box- und White-Box-Tests können auch mit erfahrungsbasierten Testentwurfsverfahren kombiniert werden, um die Erfahrung der Entwickler, Tester und Anwender bei der Festlegung, was getestet werden soll, wirksam einzusetzen.

Einige Verfahren lassen sich klar in eine dieser Kategorien einordnen; andere tragen Züge von mehr als einer Kategorie.

Dieser Lehrplan bezeichnet spezifikationsorientierte Testentwurfsverfahren als Black-Box-Verfahren, und strukturbasierte Testentwurfsverfahren als White-Box-Verfahren. Zusätzlich werden die erfahrungsbasierten Testentwurfsverfahren abgedeckt.

Gemeinsame Merkmale der spezifikationsorientierten Testentwurfsverfahren:

- Modelle, ob formal oder nicht formal, werden zur Spezifikation des zu lösenden Problems, der Software oder ihrer Komponente herangezogen.
- Testfälle können systematisch von diesen Modellen abgeleitet werden.

Gemeinsame Merkmale der strukturbasierten Testentwurfsverfahren:

- Informationen über den Aufbau der Software werden für die Ableitung von Testfällen verwendet, beispielsweise der Code und Informationen des Detailentwurfs (detailed design).
- Der Überdeckungsgrad der Software kann für vorhandene Testfälle gemessen werden. Weitere Testfälle können zur Erhöhung des Überdeckungsgrads systematisch abgeleitet werden.

Gemeinsame Merkmale der erfahrungsbasierten Testentwurfsverfahren:

- Das Wissen und die Erfahrung von Menschen wird zur Ableitung der Testfälle genutzt.
- Das Wissen von Testern, Entwicklern, Anwendern und Betroffenen über die Software, ihre Verwendung und ihre Umgebung ist eine Informationsquelle.
- Das Wissen über wahrscheinliche Fehlerzustände und ihre Verteilung ist eine weitere Informationsquelle.

4.3 Spezifikationsorientierte oder Black-Box-Verfahren (K3)	**150 Minuten**

Begriffe
Äquivalenzklassenbildung, anwendungsfallbasierter Test, Entscheidungstabellentest, Grenzwertanalyse, zustandsbasierter Test

4.3.1 Äquivalenzklassenbildung (K3)

Bei der Äquivalenzklassenbildung werden Eingabewerte für die Software oder das System in Gruppen eingeteilt, bei denen man von einem ähnlichen Verhalten ausgeht, so dass es wahrscheinlich ist, dass sie auf dieselbe Weise verarbeitet werden. Äquivalenzklassen können gleichermaßen für gültige Daten – also Werte, die angenommen werden sollten – gebildet werden wie für ungültige Daten – also Werte, die zurückgewiesen werden sollten. Die Äquivalenzklassen können außerdem für Ausgabewerte, interne Werte, zeitbezogene Werte (d.h. vor oder nach einem Ereignis) und für Schnittstellenparameter (d.h. integrierte Komponenten, die beim Integrationstest getestet werden) gebildet werden. Tests können so entworfen werden, dass alle gültigen und ungültigen Klassen abgedeckt werden. Äquivalenzklassenbildung kann in allen Teststufen angewandt werden.

Das Ziel ist, durch Bildung von Äquivalenzklassen eine hohe Fehlerentdeckungswahrscheinlichkeit bei minimaler Anzahl von Testfällen zu erreichen.

Äquivalenzklassenbildung kann eingesetzt werden, um Überdeckungsziele in Bezug auf Eingabe- oder Ausgabewerte zu erreichen. Es kann auf Eingaben eines menschlichen Benutzers, auf Eingaben an ein System über Schnittstellen oder im Integrationstest auf Schnittstellenparameter angewandt werden.

4.3.2 Grenzwertanalyse (K3)

Da das Verhalten an der Grenze jeder Äquivalenzklasse mit einer höheren Wahrscheinlichkeit fehlerhaft ist als das Verhalten innerhalb der Klasse, sind solche Grenzen ein Bereich, in dem Testen wahrscheinlich Fehlerzustände aufdecken wird. Der größte und der kleinste Wert einer Klasse sind deren Grenzwerte. Ein Grenzwert für eine gültige Klasse ist ein gültiger Grenzwert, die Grenze einer ungültigen Klasse ist ein ungültiger Grenzwert. Tests können entworfen werden, um beides, sowohl gültige als auch ungültige Grenzwerte abzudecken. Beim Entwurf von Testfällen wird ein Test für jeden Grenzwert gewählt.

Grenzwertanalyse kann in allen Teststufen angewandt werden. Sie ist vergleichsweise einfach anzuwenden und das Potenzial, Fehlerzustände aufzudecken, ist hoch. Detaillierte Spezifikationen sind bei der Bestimmung der interessanten Grenzen hilfreich.

Dieses Verfahren wird häufig als Erweiterung der Äquivalenzklassenbildung oder anderer Black-Box-Testverfahren betrachtet. Es kann bei der Bildung von Äquivalenzklassen angewendet werden, gleichermaßen für Benutzereingaben am Bildschirm, beispielsweise bei Zeitspannen (z.B. Timeout, Transaktionsgeschwindigkeitsanforderungen) oder Grenzen von Tabellenbereichen (z.B. Tabellengröße 256*256).

4.3.3 Entscheidungstabellentest (K3)

Entscheidungstabellen sind eine gute Möglichkeit, um Systemanforderungen zu erfassen, die logische Bedingungen enthalten und um den internen Systementwurf zu dokumentieren. Sie können zur Erfassung komplexer, von einem System umzusetzenden Regeln in Geschäftsprozessen verwendet werden. Beim Erstellen einer Entscheidungstabelle wird die Spezifikation analysiert, und die Bedingungen und Aktionen des Systems werden ermittelt.

Die Eingabebedingungen und Aktionen werden meist so festgesetzt, dass sie entweder „wahr“ oder „falsch“ sein müssen (Boolesche Werte). Die Entscheidungstabelle enthält die auslösenden Bedingungen, oft Kombinationen von „wahr“ und „falsch“ für alle Eingabebedingungen und die daraus resul-

tierenden Aktionen für jede Kombination der Bedingungen. Jede Spalte der Tabelle entspricht einer Regel im Geschäftsprozess, die eine eindeutige Kombination der Bedingungen definiert, die wiederum die Ausführung der mit dieser Regel verbundenen Aktionen nach sich zieht. Der üblicherweise bei Entscheidungstabellentest verwendete Standardüberdeckungsgrad besagt, dass wenigstens ein Testfall pro Spalte in der Tabelle benötigt wird, was in der Regel die Abdeckung aller Kombinationen der auslösenden Bedingungen umfasst.

Die Stärke des Entscheidungstabellentests ist, dass er Kombinationen von Bedingungen ableitet, die andernfalls beim Test möglicherweise nicht ausgeführt worden wären. Er kann in allen Situationen angewandt werden, in denen die Abläufe der Software von mehreren logischen Entscheidungen abhängen.

4.3.4 Zustandsbasierter Test (K3)

Ein System kann in Abhängigkeit von aktuellen Gegebenheiten oder von seiner Vorgeschichte (seinem Zustand) unterschiedliche Reaktionen zeigen. In diesem Fall kann dieser Aspekt des Systems mit einem Zustandsdiagramm dargestellt werden. Es ermöglicht dem Tester, die Software darzustellen in Bezug auf ihre Zustände, Übergänge zwischen den Zuständen, Eingaben oder Ereignisse, die die Zustandsübergänge (transitions) auslösen, und Aktionen, die aus den Übergängen folgen können. Die Zustände des Systems oder Testobjekts sind einzeln unterscheidbar, eindeutig identifizierbar und endlich in ihrer Anzahl.

Eine Zustandsübergangstabelle stellt den Zusammenhang zwischen Zuständen und Eingaben dar, und kann mögliche ungültige Übergänge aufzeigen.

Tests können so gestaltet sein, dass sie jeden Zustand oder eine typische Sequenz von Zuständen abdecken, jeden Übergang oder bestimmte Sequenzen von Übergängen ausführen, oder dass sie ungültige Übergänge überprüfen.

Zustandsbasierte Tests werden häufig in Branchen der eingebetteten Software (embedded software) und generell in der Automatisierungstechnik eingesetzt. Davon abgesehen ist dieses Verfahren genauso gut einsetzbar für die Modellierung von Geschäftsobjekten, die verschiedene Zustände besitzen, oder zum Test von dialogbasierten Abläufen (z.B. für Internet-Anwendungen oder Geschäftsszenarios).

4.3.5 Anwendungsfallbasierter Test (K2)

Tests können aus Anwendungsfällen (use cases) abgeleitet werden. Ein Anwendungsfall beschreibt die Interaktionen zwischen den Akteuren (Anwender oder Systeme), die ein aus Sicht des Anwenders oder Kunden gewünschtes und wahrnehmbares Ergebnis zur Folge haben. Anwendungsfälle können auf einer abstrakten Ebene (fachlicher Anwendungsvorfall, technologiefrei, Geschäftsprozessebene) oder auf einer Systemebene (Systemanwendungsfall auf Ebene der Systemfunktionalität) beschrieben werden. Jeder Anwendungsfall hat Vorbedingungen, die erfüllt sein müssen, damit der Anwendungsfall erfolgreich durchgeführt werden kann. Jeder Anwendungsfall endet mit Nachbedingungen, den beobachtbaren Ergebnissen und dem Endzustand des Systems, wenn der Anwendungsfall vollständig abgewickelt wurde. Ein Anwendungsfall hat üblicherweise ein Hauptszenario (das wahrscheinlichste Szenario) und alternative Szenarien.

Anwendungsfälle beschreiben die „Prozessabläufe" durch das System auf Grundlage seiner voraussichtlich tatsächlichen Verwendung. Daher sind von Anwendungsfällen abgeleitete Testfälle bestens geeignet, während des Praxiseinsatzes des Systems Fehlerzustände in den Prozessabläufen aufzudecken. Anwendungsfälle sind für den Entwurf von Abnahmetests mit Kunden-/Anwenderbeteiligung sehr hilfreich. Indem das Zusammenwirken und die gegenseitige Beeinflussung unterschiedlicher Komponenten betrachtet werden, können sie auch Fehlerzustände im Umfeld der Integration aufdecken, die durch den Test der einzelnen Komponenten nicht gefunden werden könnten. Das Entwerfen von Testfällen auf Basis von Anwendungsfällen kann mit anderen spezifikationsorientierten Testentwurfsverfahren kombiniert werden.

4.4 Strukturbasierter Test oder White-Box-Verfahren (K4)	**60 Minuten**

Begriffe
Anweisungsüberdeckung, Codeüberdeckung, Entscheidungsüberdeckung, strukturbasierter Test .

Hintergrund

Der strukturbasierte oder White-Box-Test baut auf der vorgefundenen Struktur der Software oder des Systems auf, wie aus folgenden Beispielen ersichtlich ist:

- Komponentenebene: Die Struktur der Softwarekomponente, d.h. Anweisungen, Entscheidungen, Zweige oder sogar einzelne Pfade
- Integrationsebene: Die Struktur kann ein Aufrufgraph sein (ein Diagramm, das zeigt, welche Module andere Module aufrufen)
- Systemebene: Die Struktur kann die Menüstruktur sein, Geschäftsprozesse oder die Struktur einer Webseite

In diesem Abschnitt werden drei codebezogene, strukturbasierte Testentwurfsverfahren für Codeüberdeckung vorgestellt, bezogen auf Anweisungen, Zweige und Entscheidungen. Für Entscheidungstest können Kontrollflussgraphen zur Darstellung der Alternativen für jede Entscheidung herangezogen werden.

4.4.1 Anweisungstest und -überdeckung (K4)

Im Komponententest steht Anweisungsüberdeckung für die Messung des prozentualen Anteils von allen Anweisungen einer Komponente, welche durch eine Testsuite ausgeführt wurden. Das Anweisungstestverfahren leitet Testfälle so ab, dass bestimmte Anweisungen ausgeführt werden, in der Regel mit dem Ziel die Anweisungsüberdeckung zu erhöhen.

Anweisungsüberdeckung ist bestimmt durch die Anzahl ausführbarer Anweisungen, die durch entworfene oder ausgeführte Testfälle überdeckt sind, dividiert durch die Anzahl aller ausführbaren Anweisungen des Programmcodes im Test.

4.4.2 Entscheidungstest und -überdeckung (K4)

Die Entscheidungsüberdeckung, die mit dem Zweigtest verwandt ist, ist die Messung des prozentualen Anteils eines Entscheidungsergebnisses (z.B. „wahr“ und „falsch“ bei einer IF-Anweisung), welche durch eine Testsuite ausgeführt wurden. Beim Entscheidungstestverfahren werden Testfälle abgeleitet, um spezifische Entscheidungen zu durchlaufen. Zweige nehmen ihren Anfang in Entscheidungspunkten des Programmcodes und zeigen die Übertragung der Steuerung zu verschiedenen Stellen im Code.

Die Entscheidungsüberdeckung ist bestimmt durch die Anzahl aller Entscheidungsausgänge, die durch entworfene oder ausgeführte Testfälle überdeckt sind, dividiert durch die Anzahl aller Entscheidungsausgänge des Programmcodes im Test.

Der Entscheidungstest ist eine Form des kontrollflussbasierten Tests, da er einem speziellen Kontrollfluss durch die Entscheidungspunkte folgt. Entscheidungsüberdeckung ist stärker als Anweisungsüberdeckung: 100% Entscheidungsüberdeckung schließt 100% Anweisungsüberdeckung ein, aber nicht umgekehrt.

4.4.3 Andere strukturbasierte Verfahren (K1)

Über Entscheidungsüberdeckung hinaus gibt es stärkere strukturelle Überdeckungsgrade, beispielsweise Bedingungsüberdeckung und Mehrfachbedingungsüberdeckung.

Das Konzept der Überdeckungsgrade kann auch auf andere Teststufen übertragen werden. Beispielsweise wird auf der Integrationsebene der prozentuale Anteil von Modulen, Komponenten oder

Klassen, die durch eine Testsuite ausgeführt wurden, als Modul-, Komponenten- oder Klassen-Überdeckung bezeichnet.

Werkzeugunterstützung ist beim strukturbasierten Test von Code sehr hilfreich.

4.5 Erfahrungsbasierte Verfahren (K2) — 30 Minuten

Begriffe
Exploratives Testen, (Fehler-)Angriff

Hintergrund

Erfahrungsbasiertes Testen bezeichnet Tests, die durch das Können und die Intuition des Testers und aus seiner Erfahrung mit ähnlichen Applikationen und Technologien abgeleitet werden. Wenn es zur Unterstützung systematischer Verfahren eingesetzt wird, kann dieses Verfahren zur Ermittlung spezieller Tests nützlich sein, die von formalen Verfahren nicht leicht erfasst werden, insbesondere wenn erfahrungsbasiertes Testen nach den formaleren Ansätzen eingesetzt wird. Abhängig von der Erfahrung des Testers kann die Wirksamkeit dieses Verfahrens sehr stark variieren.

Ein weit verbreitetes erfahrungsbasiertes Testverfahren ist die intuitive Testfallermittlung (error guessing). Gewöhnlich sehen Tester Fehler auf Grund ihrer Erfahrung voraus. Eine strukturierte Herangehensweise an das Verfahren der intuitiven Testfallermittlung ist es, eine Liste möglicher Fehlerzustände zu erstellen, und dann Testfälle zu entwerfen, die auf diese Fehlerzustände abzielen. Dieser systematische Ansatz wird Fehlerangriff (fault attack) genannt. Die Liste der Fehlerzustände und Fehlerwirkungen kann erstellt werden auf der Basis von Erfahrungen, verfügbaren Daten über Fehlerzustände und Fehlerwirkungen und von Allgemeinwissen darüber, warum Software sich falsch verhalten kann.

Exploratives Testen ist gleichzeitiger Testentwurf, Testdurchführung, Testprotokollierung und Lernen, auf Grundlage einer Test-Charta, der die Testziele zu entnehmen sind. Es wird innerhalb festgelegter Zeitfenster durchgeführt. Es ist ein Ansatz, der sich besonders gut eignet, wenn es nur wenige oder ungeeignete Spezifikationen gibt, unter hohem Zeitdruck, oder um andere, formalere Testverfahren zu unterstützen oder zu ergänzen. Es kann auch zur Überprüfung des Testprozesses dienen und helfen sicherzustellen, dass die schwerwiegendsten Fehlerzustände gefunden werden.

4.6 Auswahl von Testverfahren (K2) — 15 Minuten

Begriffe
keine besonderen Begriffe

Hintergrund

Die Wahl, welche Testverfahren verwendet werden sollen, hängt von einer Vielzahl von Faktoren ab, einschließlich der Art des Systems, regulatorischen Anforderungen, Kunden- oder Vertragsanforderungen, Risikostufe, Risikotyp, Testziel, verfügbarer Dokumentation, Wissen der Tester, Zeit und Geld, Softwareentwicklungsmodell, Anwendungsfallmodelle sowie frühere Erfahrungen mit gefundenen Fehlerzustandsarten.

Einige Verfahren sind für bestimmte Situationen und Teststufen besser geeignet; andere sind in allen Teststufen gleichermaßen einsetzbar.

Beim Erstellen von Testfällen benutzen Tester gewöhnlich eine Kombination von Testverfahren einschließlich prozessgetriebener, regelbasierter und datengetriebener (data-driven) Verfahren, um eine angemessene Überdeckung des Objekts im Test zu gewährleisten.

Referenzen
4 Linz, 2012
4.1 Craig, 2002, Hetzel, 1988, IEEE STD 829-1998
4.2 Beizer, 1990, Copeland, 2004
4.3.1 Copeland, 2004, Myers, 2001
4.3.2 Copeland, 2004, Myers, 2001, Linz, 2012
4.3.3 Beizer, 1990, Copeland, 2004
4.3.4 Beizer, 1990, Copeland, 2004
4.3.5 Copeland, 2004
4.4.3 Beizer, 1990, Copeland, 2004
4.5 Kaner, 2002
4.6 Beizer, 1990, Copeland, 2004

5 Testmanagement (K3) — 170 Minuten

Lernziele für den Abschnitt Testmanagement

Die Lernziele legen fest, was Sie nach Beenden des jeweiligen Moduls gelernt haben sollten.

5.1 Testorganisation (K2)

LO-5.1.1 Die Bedeutung unabhängigen Testens erkennen können. (K1)

LO-5.1.2 Vor- und Nachteile unabhängigen Testens innerhalb einer Organisation erklären können. (K2)

LO-5.1.3 Erkennen können, welche Rollen bei der Zusammenstellung eines Testteams durch Teammitglieder abgedeckt werden müssen. (K1)

LO-5.1.4 Aufgaben eines typischen Testmanagers und Testers wiedergeben können. (K1)

5.2 Testplanung und -aufwandsschätzung (K3)

LO-5.2.1 Die unterschiedlichen Stufen und Ziele der Testplanung erkennen können. (K1)

LO-5.2.2 Ziel und Inhalt des Testkonzepts, der Testentwurfsspezifikation und der Testablaufsdokumente auf der Basis des ‘Standard for Software Test Documentation’ (IEEE Std 829 - 1998) zusammenfassen können. (K2)

LO-5.2.3 Unterscheiden können zwischen konzeptionell verschiedenen Testvorgehensweisen wie analytisch, modellbasiert, methodisch, prozess-/standardkonform, dynamisch/ heuristisch, beratend oder wiederverwendungsorientiert. (K2)

LO-5.2.4 Unterscheiden können zwischen dem Gegenstand der Testplanung für ein System und der Planung der Testdurchführung. (K2)

LO-5.2.5 Einen Testausführungsplan für einen vorgegebenen Satz an Testfällen schreiben und dabei Priorisierung sowie technische und fachliche Abhängigkeiten berücksichtigen können. (K3)

LO-5.2.6 Testvorbereitungs- und Testdurchführungsaktivitäten auflisten können, die bei der Testplanung berücksichtigt werden müssen. (K1)

LO-5.2.7 Typische Faktoren benennen können, die den Testaufwand beeinflussen. (K1)

LO-5.2.8 Unterscheiden können zwischen zwei konzeptionell verschiedenartigen Methoden für die Aufwandsschätzung: dem metrikbasierten und dem expertenbasierten Ansatz. (K2)

LO-5.2.9 Erkennen/begründen können von angemessenen Eingangskriterien u. Endekriterien für spezifische Teststufen und Gruppen von Testfällen (z.B. für Integrationstests, Abnahmetests oder Testfälle für Benutzbarkeitstests). (K2)

5.3 Testfortschrittsüberwachung und -steuerung(K2)

LO-5.3.1 Die allgemeinen Metriken wiedergeben können, die für die Überwachung von Testvorbereitung und Testdurchführung angewendet werden. (K1)

LO-5.3.2 Testmetriken für Testberichte und Teststeuerung (z.B. aufgedeckte und behobene Fehlerzustände und bestandene und nicht bestandene Tests) in Bezug auf Zweck und Nutzung erklären und vergleichen können. (K2)

LO-5.3.3 Zweck und Inhalt des Testabschlussberichts auf der Basis des ‘Standard for Software Test Documentation’ (IEEE Std 829-1998) zusammenfassen können. (K2)

5.4 Konfigurationsmanagement (K2)

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

5.5 Risiko und Testen (K2)

LO-5.5.1 Ein Risiko als ein mögliches Problem beschreiben können, das das Erreichen der Projektziele von einem oder mehreren Stakeholdern gefährdet. (K2)

LO-5.5.2 Wiedergeben können, dass die Höhe des Risikos durch die Wahrscheinlichkeit (des Eintritts) und die Auswirkung (Schaden im Eintrittsfall) bestimmt wird. (K1)

LO-5.5.3 Zwischen den Projekt- und den Produktrisiken unterscheiden können. (K2)

LO-5.5.4 Typische Produkt- und Projektrisiken erkennen können. (K1)

LO-5.5.5 Durch Beispiele beschreiben können, wie Risikoanalyse und Risikomanagement für die Testplanung eingesetzt werden können. (K2)

5.6 Abweichungsmanagement/Fehlermanagement (K3)

LO-5.6.1 Den Inhalt eines Abweichungsberichts auf Basis des ‘Standard for Software Test Documentation’ (IEEE Std 829-1998) kennen. (K1)

LO-5.6.2 Einen Abweichungsbericht über die Beobachtung einer Fehlerwirkung während des Testens schreiben können. (K3)

5.1 Testorganisation (K2)	**30 Minuten**

Begriffe
Tester, Testmanager

5.1.1 Testorganisation und Unabhängigkeit (K2)

Die Effektivität der Fehlerfindung durch Testen und Prüfungen kann durch den Einsatz unabhängiger Tester verbessert werden. Die folgenden Organisationsformen unterscheiden sich im Grad der Unabhängigkeit:

- kein unabhängiger Tester. Entwickler testen ihren eigenen Code
- unabhängige Tester innerhalb des Entwicklungsteams
- unabhängiges Testteam oder -gruppe innerhalb der Organisation, das/die dem Projektmanagement oder dem Management der Linienorganisation berichtet
- unabhängige Tester aus der Fachabteilung oder Anwendergruppe
- unabhängige Testspezialisten für spezifische Testarten, wie Tester für Benutzerfreundlichkeit, Sicherheits- oder Konformitätstester (die ein Softwareprodukt gegen Standards und gesetzliche Vorschriften prüfen)
- unabhängige Tester, ausgegliedert oder aus externen Organisationen

Für große, komplexe Projekte oder Projekte sicherheitskritische Systeme betreffend ist es im Allgemeinen am besten, verschiedene Teststufen durchzuführen und dabei die Tests einiger oder aller Stufen von unabhängigen Testern ausführen zu lassen. Entwicklungspersonal kann speziell in niedrigen Teststufen am Testen beteiligt sein, wobei ihr Mangel an Objektivität oft ihre Effektivität beschränkt. Die unabhängigen Tester können die Befugnis besitzen, Testprozesse und Regeln zu fordern und zu definieren, sollten diese prozessverwandten Rollen aber nur bei Vorliegen eines klaren Managementauftrags einnehmen.

Vorteile von Unabhängigkeit:

- Unabhängige Tester sehen andere und unterschiedliche Fehler und sind unvoreingenommen.
- Ein unabhängiger Tester kann Annahmen verifizieren, die während der Spezifikation und Implementierung des Systems gemacht wurden.

Nachteile von Unabhängigkeit:

- Tester sind vom Entwicklungsteam isoliert (wenn als vollkommen unabhängig behandelt).
- Die Entwickler können das Verantwortungsgefühl für Qualität verlieren.
- Unabhängige Tester können als Engpass gesehen werden oder die Schuld für Verzögerungen zugewiesen bekommen.

Testaufgaben können von Personen in einer spezifischen Testrolle oder von jemandem in einer anderen Rolle durchgeführt werden, beispielsweise Projektmanager, Qualitätsmanager, Entwickler, Fach- und Bereichsexperte, Mitarbeiter in Infrastruktur oder IT-Betrieb.

5.1.2 Aufgaben von Testmanager und Tester (K1)

In diesem Lehrplan werden die zwei Rollen Testmanager und Tester behandelt. Die Aktivitäten und Aufgaben, die von Personen mit diesen zwei Rollen durchgeführt werden, hängen von Projekt- und Produktkontext, den Personen in den Rollen und der Organisation ab.

Manchmal wird der Testmanager als Testleiter oder Testkoordinator bezeichnet. Die Rolle des Testleiters kann von einem Projektleiter, einem Entwicklungsleiter, einem Qualitätsmanager oder dem Manager einer Testgruppe ausgeübt werden. In größeren Projekten können zwei Rollen existieren: Testmanager und Testleiter/ Testkoordinator. Typischerweise plant, überwacht und steuert der Testmanager die Testaktivitäten und die Aufgaben wie in Kapitel1.4 definiert.

Typische Aufgaben eines Mitarbeiters in der Rolle Testmanager können sein:

- Koordination der Teststrategie und Planung mit Projektleitern und anderen Beteiligten
- Erstellen oder Prüfen der Teststrategie für das Projekt und einer Testrichtlinie für die Organisation
- Einbringen der Testperspektive in andere Projektaktivitäten, beispielsweise die Integrationsplanung
- Planen der Tests – unter Beachtung des Kontexts und mit Verständnis von Testzielen und Risiken – einschließlich Auswahl der Testvorgehensweise, Schätzen der Zeit, des Aufwands und der Kosten des Testens, Ressourcenbeschaffung, Definition der Teststufen, Testzyklen und Planen des Abweichungsmanagements
- Initiieren der Spezifikation, Vorbereiten, Implementieren und Durchführen von Tests, Überwachen der Testergebnisse und Prüfen der Endekriterien
- Anpassen der Planung an Testergebnisse und Testfortschritt (manchmal in den Statusberichten dokumentiert) und Einleiten aller erforderlichen Maßnahmen bei Problemen
- Aufbau eines angemessenen Konfigurationsmanagements der Testmittel zur Rückverfolgbarkeit
- Einführen passender Metriken zum Messen des Testfortschritts und zur Bewertung der Qualität des Testens und des Produkts
- Entscheidung, was zu welchem Grad und wie automatisiert werden sollte
- Auswahl der Werkzeuge zur Testunterstützung und Organisation sämtlicher Werkzeugschulungen für Tester
- Entscheiden über die Implementierung der Testumgebung
- Schreiben von Testabschlussberichten auf der Grundlage der Informationen, die während des Testens gesammelt werden

Typische Aufgaben eines Testers können sein:

- Mitarbeit an und Prüfung von Testkonzepten
- Analyse, Prüfung und Bewertung von Benutzeranforderungen, Spezifikationen und Modellen im Hinblick auf Testbarkeit
- Erstellen von Testspezifikationen
- Aufbau der Testumgebung (oft in Abstimmung mit System- und Netzwerkadministration).
- Vorbereiten oder Anfordern von Testdaten
- Implementieren von Tests auf allen Stufen, Durchführen der Tests und ihre Protokollierung, Auswerten der Testergebnisse und Dokumentation der Abweichungen von erwarteten Ergebnissen
- Einsetzen von Testadministrations- oder Testmanagement- und Testüberwachungswerkzeugen wie gefordert
- Automatisieren von Tests (kann durch einen Entwickler oder Testautomatisierungsexperten unterstützt werden)
- Messen der Leistungsfähigkeit/Performanz von Komponenten und Systemen (wenn zutreffend)
- Prüfen der Tests, die von anderen entwickelt wurden

Personen, die in der Testanalyse, Testentwurf, spezifischen Testarten oder in der Testautomatisierung arbeiten, können Spezialisten in diesen Rollen sein. Abhängig von der Teststufe und den Risiken, die mit dem Produkt und dem Projekt verbunden sind, können verschiedene Personen die Rolle von Testern übernehmen und dabei einen gewissen Grad von Unabhängigkeit bewahren. Typische Tester auf der Komponenten- und Integrationsstufe sind Entwickler, auf der Abnahmeteststufe Fachexperten und Anwender, und für den Abnahmetest auf operativer Ebene der Betreiber.

5.2 Testplanung und -schätzung (K3)	**40 Minuten**

Begriffe
Teststrategie, Testvorgehensweise

5.2.1 Testplanung (K2)

Dieser Abschnitt behandelt den Zweck von Testplanung im Rahmen von Entwicklungs- und Implementierungsprojekten sowie für Wartungsaktivitäten. Die Planung kann in einem Mastertestkonzept und in separaten Testkonzepten für Teststufen, wie dem Systemtest und dem Abnahmetest, dokumentiert werden. Die Gliederung für ein Testplanungsdokument kann dem Standard 'Standard for Software Test Documentation' (IEEE Std 829-1998) entnommen werden.

Die Planung wird durch die Testrichtlinie der Organisation, den Testumfang, die Ziele, Risiken, Einschränkungen, Kritikalität, Testbarkeit und Verfügbarkeit von Ressourcen beeinflusst. Je weiter sich die Projekt- und Testplanung entwickelt, desto mehr Informationen werden verfügbar und desto mehr Details können im Plan berücksichtigt werden.

Testplanung ist eine kontinuierliche Aktivität und wird in allen Lebenszyklusprozessen und -aktivitäten durchgeführt. Feedback aus den Testaktivitäten wird genutzt, um sich ändernde Risiken zu erkennen, so dass die Planung angepasst werden kann.

5.2.2 Testplanungsaktivitäten (K3)

Testplanungsaktivitäten für ein ganzes System oder Systemteile können sein:

- Festlegen des Umfangs und der Risiken sowie Identifizierung der Testziele
- Definieren des allgemeinen Testansatzes, einschließlich Definition der Teststufen und der Eingangs- und Endekriterien
- Koordinieren und Integrieren der Testaktivitäten in die Aktivitäten des Softwarelebenszyklus (Beschaffung, Bereitstellung, Entwicklung, Betrieb und Wartung)
- Entscheiden, was zu testen ist, welche Rollen welche Testaktivitäten ausführen werden, wann und wie die Testaktivitäten auszuführen sind und wie die Testergebnisse bewertet werden
- Testanalyse und Entwurfsaktivitäten planen
- Testimplementierung, -ausführung und -bewertung planen
- Ressourcen den verschiedenen definierten Aufgaben zuordnen
- Definieren des Umfangs, des Detaillierungsgrads, der Struktur und der Vorlagen für die Testdokumentation
- Selektieren der Metriken zur Überwachung und Steuerung der Testvorbereitung und -durchführung, Fehlerzustandsbehebung und Risikofaktoren
- Bestimmen des Detaillierungsgrads für Testablaufspezifikationen, um genügend Informationen in Hinblick auf eine reproduzierbare Testvorbereitung und -durchführung zu liefern

5.2.3 Testeingangskriterien (K2)

Eingangskriterien bestimmen, wann der Test begonnen wird, beispielsweise zu Beginn einer Teststufe oder wenn eine Reihe Tests zur Ausführung bereit steht.

Typische Eingangskriterien:

- Verfügbarkeit und Einsatzbereitschaft der Testumgebung
- Bereitschaft der Testwerkzeuge in der Testumgebung
- Verfügbarkeit des testbaren Codes
- Verfügbarkeit der Testdaten

5.2.4 Endekriterien (K2)

Endekriterien definieren, wann mit dem Testen aufgehört wird, z.B. am Ende einer Teststufe oder wenn eine Reihe von Tests ein spezifisches Ergebnis erzielt hat.

Typische Endekriterien:

- Intensitätsmaße, beispielsweise Codeüberdeckung, Funktionalität oder Risiko
- Schätzungen über Fehlerdichte oder Zuverlässigkeitsmaße
- Kosten
- Verbleibende Risiken, beispielsweise nicht behobene Fehlerzustände oder fehlende Testüberdeckung in bestimmten Bereichen
- Zeitpläne, beispielsweise basierend auf dem Termin der Markteinführung

5.2.5 Testaufwandsschätzung (K2)

Zwei Ansätze für die Schätzung des Testaufwands sind:

- Der metrikenbasierte Ansatz: Schätzung des Testaufwands auf der Basis von Metriken früherer oder ähnlicher Projekte oder auf der Basis von typischen Werten
- Der expertenbasierte Ansatz: Schätzung des Aufwands für die einzelnen Aufgaben durch die Verantwortlichen für diese Aufgaben oder durch Experten

Sobald der Testaufwand geschätzt ist, können Ressourcen identifiziert und ein Zeitplan erstellt werden.

Der Testaufwand kann von einer Anzahl von Faktoren abhängen, u.a.:

- Charakteristiken des Produkts, Qualität der Spezifikation und anderer Informationen, die für das Testmodell herangezogen werden (d.h. die Testbasis), Größe des Produkts, Komplexität der Problembereiche, Anforderungen an Zuverlässigkeit und Sicherheit und Anforderungen an die Dokumentation
- Charakteristiken des Entwicklungsprozesses: Stabilität der Organisation, benutzte Werkzeuge, Testprozess, Kenntnisse der involvierten Personen und Zeitdruck
- Testergebnisse: Anzahl der Fehlerzustände und Menge der erforderlichen Nacharbeiten

5.2.6 Teststrategie, Testvorgehensweise (K2)

Die Testvorgehensweise ist die Umsetzung der Teststrategie in einem spezifischen Projekt. Die Testvorgehensweise wird in den Testkonzepten und im Testentwurf definiert und verfeinert. Sie enthält typischerweise die Entscheidungen, basierend auf den (Test-) Projektzielen und der Risikoanalyse. Sie ist der Ausgangspunkt für die Planung des Testprozesses, für die Auswahl der Testentwurfsverfahren und der durchzuführenden Testarten, sowie für die Spezifikation der Eingangskriterien und Endekriterien.

Die gewählte Testvorgehensweise ist abhängig vom Kontext. Die Auswahl kann beeinflusst sein von Faktoren wie Risiken, Gefahren und Sicherheit, verfügbare Ressourcen und Fähigkeiten, die Technologie, Art des Systems (z.B. maßgeschneidert oder kommerzielle COTS-Software), Testzielen und Vorschriften.

Typische Vorgehensweisen:

- Analytische Vorgehensweisen, wie das risikoorientierte Testen, in dem das Testen auf die Bereiche der größten Risiken ausgerichtet ist
- Modellbasierte Vorgehensweisen wie das stochastische Testen, das statistische Informationen über Ausfallraten (beispielsweise Zuverlässigkeitswachstumsmodelle) oder Systembenutzung (beispielsweise Benutzungsprofile) nutzt
- Methodische Vorgehensweisen wie das ausfallbasierte (einschließlich intuitiver Testfallermittlung und Fehlerangriff), erfahrungsbasierte, checklistenbasierte und qualitätsmerkmalbasierte Testen
- Prozess- oder standardkonforme Vorgehensweisen, spezifiziert durch Industriestandards, oder die verschiedenen agilen Methoden

- Dynamische und heuristische Vorgehensweisen, wie das explorative Testen, bei dem das Testen weniger vorgeplant ist und stärker auf Ereignisse reagiert und Durchführung und Auswertung parallel laufen
- Beratende Vorgehensweisen, in denen die Testüberdeckung primär durch Hinweise und Beratung von Technologie- und/oder Geschäftsbereichsexperten außerhalb des Testteams getrieben wird
- Wiederverwendungsorientierte Vorgehensweisen, bei denen man vorhandene Tests und Testumgebungen (aus früheren Projekten), umfangreiche Automatisierung von funktionalen Regressionstests und Standardtestsuiten als Ausgangsbasis übernimmt. Ziel ist, die Tests schnell und pragmatisch aufzusetzen.

Unterschiedliche Ansätze können kombiniert werden, beispielsweise zu einer risikoorientierten dynamischen Vorgehensweise.

5.3 Testfortschrittsüberwachung und -steuerung (K2)	**20 Minuten**

Begriffe
Ausfallrate, Fehlerdichte, Testabschlussbericht, Teststeuerung, Testüberwachung

5.3.1 Testfortschrittsüberwachung (K1)

Das Ziel der Testfortschrittsüberwachung ist es, Feedback und Übersicht über Testaktivitäten zu liefern. Zu überwachende Informationen können manuell oder automatisiert gesammelt werden. Sie können herangezogen werden, um Endekriterien wie Testüberdeckung zu messen sowie den Fortschritt gegen den Zeitplan und gegen das Budget zu beurteilen.

Gebräuchliche Testmetriken sind u.a.:

- Prozentsatz der durchgeführten Arbeiten in der Testvorbereitung (oder Prozentsatz der vorbereiteten geplanten Testfälle)
- Prozentsatz der durchgeführten Arbeiten in der Vorbereitung der Testumgebung
- Testfalldurchführung (z.B. die Anzahl der durchgeführten/nicht durchgeführten Testfälle und der bestandenen/nicht bestandenen Testfälle)
- Fehlerzustandsinformationen (z.B. Fehlerdichte, gefundene und behobene Fehlerzustände, Ausfallrate und Fehlernachtestergebnisse)
- Testüberdeckung der Anforderungen, Risiken oder des Codes
- subjektives Vertrauen der Tester in das Produkt
- Daten der Testmeilensteine
- Testkosten, inklusive Kosten im Vergleich zum Nutzen durch das Auffinden des nächsten Fehlerzustands oder für den nächsten Testdurchlauf

5.3.2 Testberichterstattung (K2)

Testberichterstattung beschäftigt sich mit der Zusammenfassung der Informationen über die Testaktivitäten, einschließlich:

- was während des Testzeitraums passierte, z.B. zu Zeitpunkten, an denen Endekriterien erfüllt wurden
- analysierte Informationen und Metriken zur Unterstützung von Empfehlungen und Entscheidungen über zukünftige Aktivitäten, wie eine Beurteilung der verbleibenden Fehlerzustände, die ökonomischen Vorteile fortgesetzten Testens, offen stehende Risiken und der Grad des Vertrauens in die getestete Software

Die Gliederung eines Testabschlussberichts ist im ‘Standard for Software Test Documentation’ (IEEE Std 829-1998) enthalten.

Metriken sollten während des Testens und am Ende einer Teststufe zur Beurteilung folgender Aspekte gesammelt werden:

- Angemessenheit der Testziele für die Teststufe
- Angemessenheit des gewählten Testvorgehens
- Effektivität des Testens in Relation zu den Zielen

5.3.3 Teststeuerung (K2)

Teststeuerung beschreibt sämtliche Führungs- oder Korrekturmaßnahmen, die auf Grund gesammelter oder berichteter Informationen und Metriken ergriffen werden. Maßnahmen können jede Testaktivität betreffen und können jede andere Softwarelebenszyklusaktivität oder -aufgabe beeinflussen.

Beispiele von Maßnahmen zur Teststeuerung:

- Entscheidungsfindung basierend auf Informationen aus der Testüberwachung
- Repriorisierung von Tests, wenn identifizierte Risiken auftreten (z.B. verspätete Lieferung der Software)
- Änderung des Testzeitplans aufgrund der Verfügbarkeit oder Nichtverfügbarkeit der Testumgebung
- Setzen eines Eingangskriteriums mit der Maßgabe, dass Fehlerbehebungen durch einen Entwickler nach zu testen sind, bevor die Software an die nächste Teststufe übergeben wird

5.4 Konfigurationsmanagement(K2)	10 Minuten

Begriffe
Konfigurationsmanagement, Versionskontrolle

Hintergrund

Ziel des Konfigurationsmanagements ist es, die Integrität der Produkte herzustellen und zu erhalten. Das betrifft Komponenten, Daten und Dokumentation der Software oder des Systems während des Projekt- und Produktlebenszyklus.

Für das Testen kann das Konfigurationsmanagement sicherstellen, dass:

- alle Teile der Testmittel identifiziert und einer Versionskontrolle unterworfen sind sowie Änderungen verfolgt und zueinander und zu den Entwicklungseinheiten (Testobjekten) in Beziehung gesetzt werden, so dass die Rückverfolgbarkeit während des gesamten Testprozesses oder auch des gesamten Produktlebenszyklus erhalten werden kann
- alle identifizierten Dokumente und Entwicklungsgegenstände eindeutig in der Testdokumentation referenziert werden.

Das Konfigurationsmanagement unterstützt den Tester, die Testobjekte, Testdokumente, Tests und den/die Testrahmen eindeutig zu identifizieren (und zu reproduzieren).

Während der Testplanung sollten die Konfigurationsmanagementverfahren und -infrastruktur (Werkzeuge) ausgewählt, dokumentiert und implementiert werden.

5.5 Risiko und Testen (K2)	30 Minuten

Begriffe
Produktrisiko, Projektrisiko, Risiko, risikoorientiertes Testen

Hintergrund

Risiko kann definiert werden als die Eintrittswahrscheinlichkeit eines Ereignisses, einer Gefahr, Bedrohung oder Situation, die zu unerwünschten Konsequenzen oder einem potenziellen Problem führen. Die Höhe des Risikos wird bestimmt durch die Wahrscheinlichkeit des Eintritts und die Auswirkung eines unerwünschten Ereignisses (Schaden, der aus dem Ereignis resultiert).

5.5.1 Projektrisiken (K2)

Projektrisiken sind die Risiken, die mit der Fähigkeit des Projekts zusammenhängen, seine Ziele zu erreichen:

- Organisatorische Faktoren:
  - Qualifikation, Schulung und Mitarbeiterengpässe
  - Personalaspekte
  - politische Aspekte wie
    - Probleme mit Testern, die ihre Anforderungen und Ergebnisse kommunizieren
    - Versagen des Teams bei der Verfolgung von Informationen, die durch Testen und in Reviews gefunden werden (z.B. fehlende Verbesserung der Entwicklungs- und Testpraktiken)
  - unangemessene Einstellung gegenüber oder Erwartung an das Testen (wenn beispielsweise das Finden von Fehlerzuständen beim Testen nicht als wertvoll betrachtet wird)
- Technische Aspekte:
  - Probleme bei der Definition der richtigen Anforderungen
  - Ausmaß, in dem Anforderungen unter den gegebenen Randbedingungen nicht erfüllt werden können
  - nicht rechtzeitig verfügbare Testumgebung
  - verspätete Datenkonvertierung, Migrationsplanung und -entwicklung sowie verspätete Tests der Datenkonvertierung/Migrationswerkzeuge
  - geringe Qualität des Designs, des Codes, der Konfigurationsdaten, Testdaten und der Tests
- Lieferantenaspekte:
  - Versagen einer dritten Partei
  - Vertragsaspekte

Zur Analyse, Management und Reduzierung dieser Risiken folgt der Testmanager etablierten Projektmanagementprinzipien. Eine Vorlage für Testkonzepte (test plan) im ‘Standard for Software Test Documentation’ (IEEE Std 829-1998) fordert die Benennung von Risiken und (Gegen-) Maßnahmen.

5.5.2 Produktrisiken (K2)

Mögliche Ausfallbereiche (unerwünschte zukünftige Ereignisse oder Gefahren) in der Software oder dem System werden als Produktrisiken bezeichnet, da sie ein Risiko für die Qualität des Produkts darstellen. Dazu gehören:

- gelieferte fehleranfällige Software
- Potenzial, dass die Software/Hardware einem Individuum oder einer Firma Schaden zufügen könnte
- schlechte Softwareeigenschaften (z.B. fehlende/mangelhafte Funktionalität, Zuverlässigkeit, Benutzbarkeit und Performanz)

- schlechte Datenintegrität und -qualität (z.B. Datenmigrationsprobleme, Datenkonvertierungsprobleme, Datenüberführungsprobleme, Verletzung von Datenstandards)
- Software, die ihre beabsichtigten Funktionen nicht erfüllt

Risiken werden herangezogen, um zu entscheiden, in welchem Bereich mit dem Testen begonnen wird und welche Bereiche intensiver getestet werden; Testen wird eingesetzt, um das Risiko oder den Schaden eines unerwünschten Effekts zu reduzieren.

Produktrisiken sind ein spezieller Risikotyp für den Erfolg eines Projekts. Als Aktivität der Risikoüberwachung liefert das Testen Rückmeldungen über das verbleibende Risiko, indem es die Effektivität der Behebung kritischer Fehlerzustände und von Notfallplänen misst.

Ein risikoorientiertes Testvorgehen erlaubt pro-aktive Möglichkeiten zur Reduzierung des Produktrisikos, beginnend mit den ersten Projektphasen. Es enthält die Identifikation von Produktrisiken und ihre Verwendung zur Führung der Testplanung und -steuerung sowie der Spezifikation, Vorbereitung und Durchführung von Tests. Bei einem risikoorientierten Testvorgehen können die identifizierten Risiken genutzt werden, um:

- einzusetzende Testverfahren zu bestimmen
- auszuführenden Testumfang zu bestimmen
- Testen zu priorisieren mit dem Ziel, die kritischen Fehlerzustände so früh wie möglich zu finden
- zu bestimmen, ob zusätzlich zum Testen weitere Tätigkeiten zur Risikoreduktion notwendig sind (z.B. die Bereitstellung von Schulungsmaßnahmen für unerfahrene Designer)

Risikoorientiertes Testen nutzt das Wissen und die Kenntnisse der Stakeholder, um Risiken zu identifizieren, sowie entsprechende Teststufen zur Risikobehandlung zu bestimmen.

Um sicherzustellen, dass die Wahrscheinlichkeit von Produktfehlern minimiert wird, bietet das Risikomanagement einen systematischen Ansatz für:

- Bewertung (und regelmäßige Neubewertung) dessen, was an Fehlern auftreten kann (Risiken)
- Festlegung, welche Risiken reduziert werden müssen
- Implementierung von Maßnahmen zur Behandlung dieser Risiken

Zusätzlich kann Testen die Identifikation neuer Risiken unterstützen. Es kann helfen festzulegen, welche Risiken reduziert werden sollten und es kann die Unsicherheit bezüglich der Risiken verringern.

5.6 Fehler- und Abweichungsmanagement (K3) — 40 Minuten

Begriffe
Abweichungsprotokollierung, Fehler- und Abweichungsbericht, Fehler- und Abweichungsmanagement

Hintergrund

Da eines der Ziele des Testens das Finden von Fehlerzuständen ist, müssen Unterschiede zwischen aktuellen und erwarteten Ergebnissen als Abweichung aufgezeichnet werden. Eine Abweichung muss untersucht werden. Sie kann sich als Fehlerzustand erweisen. Angemessene Maßnahmen sollten definiert sein, um Abweichungen und Fehlerzustände zu beseitigen. Abweichungen und Fehlerzustände sollten von der Entdeckung und Klassifizierung bis hin zur Korrektur und Überprüfung der Lösung verfolgt werden. Um alle Abweichungen bis zum Abschluss zu verwalten, sollte die Organisation einen Fehler- und Abweichungsmanagementprozess sowie Regeln für die Klassifizierung etablieren.

Abweichungen können während der Entwicklung, in Reviews, Tests sowie beim Einsatz von Software festgestellt werden. Sie können im Code oder dem betriebenen System oder in jeder Art von Dokumentation festgestellt werden (einschließlich Anforderungen, Entwicklungsdokumente, Testdokumente und Benutzerinformation wie „Hilfe" oder Installationsanleitungen).

Abweichungsberichte haben folgende Ziele:

- Für die Entwickler und andere Parteien liefern sie Hinweise, um nach Bedarf die Identifikation, Isolation und Korrektur zu ermöglichen.
- Für den Testmanager sind sie ein Hilfsmittel zur Verfolgung der Systemqualität im Test und des Testfortschritts.
- Sie liefern Hinweise zur Testprozessverbesserung.

Ein Abweichungsbericht kann die folgenden Informationen enthalten:

- Meldungsdatum, meldende Organisation und Autor
- IST- und erwartete Ergebnisse
- Identifikation des Testobjekts (Konfigurationsobjekt) und der Testumgebung
- Software- oder Systemlebenszyklusprozess, in dem die Abweichung beobachtet wurde
- Beschreibung der Abweichung, um Reproduzierbarkeit und Behebung zu ermöglichen (einschließlich Protokoll, Datenbank-Dumps oder Screenshots)
- Umfang und Grad der Auswirkung auf Stakeholder-Interessen
- Klassifizierung der Schwere der Auswirkung auf das System
- Dringlichkeit/Priorität für die Behebung
- Status der Abweichung (z.B. offen, zurückgestellt, dupliziert, wartet auf Behebung, Behebung wartet auf Fehlernachtest oder geschlossen)
- Schlussfolgerungen, Empfehlungen und Freigaben
- globale Punkte, beispielsweise andere Bereiche, die durch eine Änderung aufgrund der Abweichung beeinflusst sein könnten
- Änderungshistorie, was von Projektteammitgliedern im Zusammenhang mit der Abweichung zur Isolation, Behebung und Bestätigung der Behebung durchgeführt wurde
- Referenzen einschließlich der Testfallspezifikations-ID, welche das Problem aufdeckte

Der Aufbau eines Abweichungsberichts wird auch im 'Standard for Software Test Documentation' (IEEE Std 829-1998) behandelt.

Referenzen
5 Linz, 2012
5.1.1 Black, 2001, Hetzel, 1988
5.1.2 Black, 2001, Hetzel, 1988
5.2.6 Black, 2001, Craig, 2002, IEEE Std 829-1998, Kaner 2002
5.3.3 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE Std 829-1998
5.4 Craig, 2002
5.5.2 Black, 2001, IEEE Std 829-1998
5.6 Black, 2001, IEEE Std 829-1998

6 Testwerkzeuge (K2) — 80 Minuten

Lernziele für den Abschnitt Testwerkzeuge

Die Lernziele legen fest, was Sie nach Beenden des jeweiligen Moduls gelernt haben sollten.

6.1 Typen von Testwerkzeugen (K2)

LO-6.1.1 Verschiedene Testwerkzeuge nach ihrem Zweck, den Aktivitäten des fundamentalen Testprozesses und dem Softwarelebenszyklus klassifizieren können. (K2)

LO-6.1.3 Den Begriff Testwerkzeug und den Zweck der Werkzeugunterstützung für den Test erklären können. (K2) [2]

6.2 Effektive Anwendung von Werkzeugen: Potenzieller Nutzen und Risiken(K2)

LO-6.2.1 Den möglichen Nutzen und die möglichen Risiken der Werkzeugunterstützung und Testautomatisierung benennen und erklären können. (K2)

LO-6.2.2 Sich an besondere Gesichtspunkte von Testausführungswerkzeugen, statischer Analyse - und Testmanagementwerkzeugen erinnern können. (K1)

6.3 Einführung von Testwerkzeugen in eine Organisation (K1)

LO-6.3.1 Die wichtigsten Schritte bei einer Werkzeugeinführung in einer Organisation nennen können. (K1)

LO-6.3.2 Die Ziele einer Vorstudie zur Werkzeugevaluierung und einer Pilotphase zur Werkzeugimplementierung benennen können. (K1)

LO-6.3.3 Die wichtigsten Schritte einer erfolgreichen Werkzeugeinführung kennen und wissen, dass es nicht ausreicht, lediglich ein Werkzeug zu kaufen. (K1)

[2] LO-6.1.2 wurde absichtlich übersprungen

6.1 Typen von Testwerkzeugen (K2) — 45 Minuten

Begriffe
Analysewerkzeug, Anforderungsmanagementwerkzeug, Debugger, dynamisches Analysewerkzeug, Fehler- und Abweichungsmanagementwerkzeug, Komparator, Unittest-Framework, Konfigurationsmanagementwerkzeug, Lasttestwerkzeug, Modellierungswerkzeug, Performanztestwerkzeug, Sicherheitsprüfwerkzeug, Stresstestwerkzeug, Testausführungswerkzeug, Testdatengenerator, Testentwurfswerkzeug, Testmanagementwerkzeug, Testmonitor, Testrahmen, Überdeckungsanalysator, Werkzeugunterstützung für den Reviewprozess

6.1.1 Werkzeugunterstützung für das Testen (K2)

Testwerkzeuge können eine oder mehrere Aktivitäten des Testens unterstützen. Hierzu gehören:

1. Werkzeuge, die direkt beim Testen eingesetzt werden, wie Testdurchführungswerkzeuge, Testdatengeneratoren und Ergebniskomparatoren
2. Werkzeuge, die das Management des Testprozesses unterstützen, wie Werkzeuge für das Management von Tests, Testergebnissen, Daten, Anforderungen, Abweichungen, Fehlerzuständen usw., sowie Werkzeuge für das Berichtswesen und zur Überwachung der Testdurchführung
3. Werkzeuge, die zu Untersuchungszwecken eingesetzt werden (z.B. Werkzeuge, die Dateiaktivitäten einer Anwendung überwachen).
4. alle Werkzeuge, die das Testen unterstützen (auch ein Tabellenkalkulationsprogramm gilt in dieser Hinsicht als Testwerkzeug)

Die Werkzeugunterstützung für den Test/das Testen kann je nach Kontext einem oder mehreren Zwecken dienen:

- Steigerung der Effizienz der Testaktivitäten durch eine Automatisierung sich wiederholender Testaufgaben oder durch die Unterstützung manueller Testaktivitäten, wie Testplanung, Testentwurf, Testberichts- und Überwachungsaufgaben
- Automatisierung von Testaktivitäten, für deren manuelle Durchführung erhebliche Ressourcen notwendig wären (z.B. statisches Testen)
- Automatisierung von Testaktivitäten, die manuell nicht durchgeführt werden können (z.B. Performanztests von Client-Server-Anwendungen in größerem Umfang)
- Steigerung der Zuverlässigkeit des Testens (z.B. durch automatisierten Vergleich großer Datenmengen oder durch simuliertes Verhalten)

Auch der Begriff „Testframework" wird häufig in der Industrie verwendet, und zwar mit mindestens drei Bedeutungen:

- wiederverwendbare und erweiterbare Testbibliotheken, die zum Erstellen von Testwerkzeugen dienen können (auch als Testrahmen bezeichnet)
- Art des Entwurfs der Testautomatisierung (z.B. datengetrieben, schlüsselwortgetrieben)
- der gesamte Prozess der Testdurchführung

In diesem Lehrplan wird der Begriff „Testrahmen" in den beiden erstgenannten Bedeutungen verwendet, wie in Abschnitt 6.1.6 beschrieben.

6.1.2 Klassifizierung von Testwerkzeugen (K2)

Es existieren Testwerkzeuge, die verschiedene Aspekte des Testens unterstützen. Werkzeuge können nach verschiedenen Kriterien klassifiziert werden:

- Zweck
- kommerziell/kostenlos/Open-Source/Shareware
- verwendete Technologie etc.

In diesem Lehrplan werden Testwerkzeuge danach klassifiziert, welche Aktivitäten des Testens sie unterstützen.

Einige Testwerkzeuge unterstützen lediglich eine Testaktivität; andere können mehreren Testaktivitäten zugeordnet werden, wobei sich die Klassifizierung an der nahe liegendsten Verwendung orientiert. Werkzeuge eines einzelnen Anbieters, besonders solche, die dazu vorgesehen sind, gemeinsam eingesetzt zu werden und die untereinander Schnittstellen haben, können als eine Einheit betrachtet werden.

Einige Testwerkzeuge werden als intrusiv bezeichnet; d.h. sie können das Verhalten des Testobjekts beeinflussen. Zum Beispiel kann das tatsächliche Zeitverhalten durch zusätzliche Befehle, die durch das Werkzeug ausgeführt werden, unterschiedlich sein oder es kann ein unterschiedlicher Überdeckungsgrad gemessen werden. Ein solches intrusives Verhalten eines Testwerkzeugs wird auch als „Untersuchungseffekt“ bezeichnet.

Einige der existierenden Werkzeuge sind insbesondere für Entwickler zur Unterstützung des Komponententests und des Komponentenintegrationstests geeignet. Solche Werkzeuge werden im Folgenden mit „(E)” gekennzeichnet.

6.1.3 Werkzeugunterstützung für das Management des Testens (K1)

Managementwerkzeuge können für sämtliche Testaktivitäten über den gesamten Softwarelebenszyklus verwendet werden.

Testmanagementwerkzeuge

Diese Werkzeuge bieten Schnittstellen für die Testdurchführung, das Verfolgen von Fehlerzuständen und das Verwalten von Anforderungen zusammen mit der Unterstützung von quantitativen Analysen und dem Berichten über Testobjekte. Sie unterstützen auch das Nachverfolgen der Testobjekte zu den Anforderungsspezifikationen und können unabhängige Möglichkeiten zur Versionskontrolle oder eine Schnittstelle zu einem entsprechenden externen Werkzeug aufweisen.

Anforderungsmanagementwerkzeuge

Diese Werkzeuge verwalten Anforderungsbeschreibungen, speichern die Merkmale der Anforderungen (einschließlich Priorität), liefern eindeutige Bezeichnungen und unterstützen die Nachverfolgung der Anforderungen bis zu einzelnen Tests. Diese Werkzeuge können auch bei der Identifizierung inkonsistenter oder fehlender Anforderungen helfen.

Fehler- und Abweichungsmanagementwerkzeuge

Diese Werkzeuge speichern und verwalten Fehler- und Abweichungsberichte, d.h. Fehlerzustände, Änderungsanforderungen (change requests), Fehlerwirkungen oder wahrgenommene Probleme und Anomalien. Sie unterstützen die Verwaltung des Lebenszyklus von Abweichungen, optional auch mit statistischen Analysen.

Konfigurationsmanagementwerkzeuge

Obwohl sie im engeren Sinne keine Testwerkzeuge sind, sind sie für Ablage und Versionsmanagement von Testmitteln und damit im Zusammenhang stehender Software nötig, insbesondere wenn mehr als eine Hardware-/Softwareumgebung mit verschiedenen Betriebssystemversionen, Compilern, Browsern etc. zu konfigurieren ist.

6.1.4 Werkzeugunterstützung für den statischen Test (K1)

Werkzeuge für statische Tests liefern einen kostengünstigen Weg, um mehr Fehlerzustände in früheren Phasen des Entwicklungsprozesses zu finden.

Reviewwerkzeuge

Diese Werkzeuge helfen mit Prozessen, Checklisten und Regeln beim Review und werden zur Ablage und Kommunikation von Reviewkommentaren und Berichten zu Fehlerzuständen und Aufwand verwendet. Darüber hinaus können diese Werkzeuge bei verteilten Reviews (online reviews) unterstützen, insbesondere bei großen Teams oder wenn die Mitglieder des Reviewteams geographisch verteilt sind.

Statische Analysewerkzeuge (E)

Diese Werkzeuge unterstützen Entwickler und Tester in der Aufdeckung von Fehlerzuständen vor dem dynamischen Testen, indem sie Programmierkonventionen einschließlich sicherer Programmierung (secure coding) erzwingen und die Analyse von Strukturen und Abhängigkeiten ermöglichen. Sie können ebenso bei der Planung oder Risikoanalyse helfen, indem sie Metriken (z.B. Komplexität) aus dem Code ermitteln.

Modellierungswerkzeuge (E)

Diese Werkzeuge werden verwendet, um Softwaremodelle zu validieren (z.B. physikalische Datenmodelle, PDM, für relationale Datenbanken). Sie zeigen Inkonsistenzen auf und finden Fehlerzustände und werden oft genutzt, um das Generieren von auf dem Modell basierenden Testfällen zu unterstützen.

6.1.5 Werkzeugunterstützung für die Testspezifikation (K1)

Testentwurfswerkzeuge

Diese Werkzeuge werden verwendet, um Testeingaben, ausführbare Tests und/oder Testorakel aus den Anforderungen, der graphischen Benutzungsschnittstelle (GUI), dem Entwurfsmodell (Zustands-, Daten- oder Objektmodell) oder aus dem Code zu generieren.

Testdatengeneratoren und -editoren

Mit Hilfe von Testdatengeneratoren und -editoren können aus Datenbanken, Dateien oder Datenströmen zunächst Testdaten ermittelt und dann sämtliche für einen Test benötigte Testdaten bearbeitet werden, um den Datenschutz durch Anonymisierung sicherzustellen.

6.1.6 Werkzeugunterstützung für die Testdurchführung und die Protokollierung (K1)

Testausführungswerkzeuge

Diese Werkzeuge, ermöglichen eine automatische oder halbautomatische Ausführung von Tests unter Verwendung der gespeicherten Eingaben und der erwarteten Ausgaben mittels einer Skriptsprache. Normalerweise liefern sie für jeden Testlauf ein Protokoll. Sie können auch genutzt werden, um Tests aufzuzeichnen, und üblicherweise unterstützen sie Skriptsprachen oder eine GUI-basierte Konfiguration zur Parametrisierung der Daten oder für andere spezifische Anpassungen.

Testrahmen/Unittest-Framework(E)

Ein Unittest-Framework oder Testrahmen erleichtert den Test einer Komponente oder eines Teilsystems durch Simulation der Umgebung des Testobjekts, durch Scheinobjekte (Simulatoren) als Platzhalter und/oder Treiber.

Vergleichswerkzeuge/Komparatoren

Vergleichswerkzeuge ermitteln die Unterschiede zwischen Dateien, Datenbanken oder Testergebnissen. Testausführungswerkzeuge enthalten typischerweise dynamische Vergleichswerkzeuge. Es besteht auch die Möglichkeit, dass ein Vergleich durch ein separates Werkzeug erst nach der Testdurchführung ausgeführt wird. Ein Vergleichswerkzeug kann auch ein Testorakel verwenden, insbesondere wenn der Vergleich automatisiert erfolgt.

Werkzeuge zur Überdeckungsmessung (E)

Diese Werkzeuge messen – mit intrusiven oder nicht-intrusiven Mitteln – den prozentualen Anteil spezifischer Codestrukturtypen (z.B. Anweisungen, Zweige oder Entscheidungen und Module oder Funktionsaufrufe), die durch eine Menge von Tests ausgeführt bzw. durchlaufen wurden.

Sicherheitsprüfwerkzeuge

Diese Werkzeuge dienen der Bewertung der Sicherheitsmerkmale von Software. Dabei wird bewertet, wie die Software in der Lage ist, die Vertraulichkeit der Daten, deren Integrität, Bestätigung der Echtheit, Autorisierung, Verfügbarkeit und Nichtabstreitbarkeit zu schützen. Sicherheitsprüfwerkzeuge sind meist für eine bestimmte Technologie, Plattform und Zielsetzung ausgelegt.

6.1.7 Werkzeugunterstützung für Performanzmessungen und Testmonitore (K1)

Dynamische Analysewerkzeuge (E)

Dynamische Analysewerkzeuge decken Fehlerzustände auf, wie sie lediglich zur Laufzeit eines Programms sichtbar werden, also z.B. Zeitabhängigkeiten und Speicherengpässe. Diese werden typischerweise im Komponenten- und Komponentenintegrationstest sowie im Rahmen der Tests der Middleware verwendet.

Performanztest-/Lasttest-/Stresstestwerkzeuge

Performanztestwerkzeuge überwachen und protokollieren, wie sich ein System unter verschiedenen simulierten Benutzungsbedingungen verhält, hinsichtlich Anzahl konkurrierender Nutzer, Hochlauf-/Anlaufverhalten (ramp-up pattern) sowie Häufigkeit und relativem Anteil von Transaktionen. Die Last wird durch Erzeugen virtueller Nutzer simuliert, die einen ausgewählten Satz an Transaktionen durchführen, verteilt auf verschiedene Testmaschinen, allgemein bekannt als Lastgeneratoren.

Testmonitore

Testmonitore analysieren, verifizieren und zeichnen kontinuierlich die Verwendung von spezifischen Systemressourcen auf und geben Warnungen zu. möglichen Problemen bei der Erbringung von Diensten aus.

6.1.8 Werkzeugunterstützung für spezifische Anwendungsbereiche (K1)

Bewertung der Datenqualität

Bei manchen Projekten stehen die Daten im Mittelpunkt, beispielsweise bei Datenkonvertierungs- bzw. Datenmigrationsprojekten und bei Anwendungen wie Data Warehouses. Die Attribute der Daten können sowohl hinsichtlich ihrer Kritikalität und ihres Volumens variieren. In einem solchen Kontext sind Werkzeuge für die Bewertung der Datenqualität erforderlich, um die Datenkonvertierungs- und die Migrationsvorschriften zu prüfen und zu verifizieren. Damit soll sichergestellt werden, dass die verarbeiteten Daten korrekt und vollständig sind, und dass sie einem vorab definierten kontextspezifischen Standard entsprechen.

Auch für den Benutzbarkeitstest gibt es Testwerkzeuge.

6.2 Effektive Anwendung von Werkzeugen: Potenzieller Nutzen und Risiken (K2) — 20 Minuten

Begriffe
datengetriebener Test, schlüsselwortgetriebener Test, Skriptsprache

6.2.1 Potenzieller Nutzen und Risiken einer Werkzeugunterstützung für das Testen (für alle Werkzeuge) (K2)

Einfach ein Werkzeug zu kaufen oder zu mieten, garantiert noch keinen Erfolg mit dem Werkzeug. Jede Art von Werkzeug kann zusätzlichen Aufwand erfordern, um einen tatsächlichen und nachhaltigen Nutzen zu erreichen. Werkzeuge können Chancen und erheblichen potenziellen Nutzen bei der Unterstützung des Testens bieten. Allerdings birgt der Einsatz auch Risiken, die berücksichtigt werden müssen.

Potenzieller Nutzen der Verwendung von Testwerkzeugen:

- weniger sich wiederholende Tätigkeiten (z.B. für Regressionstestläufe, wiederholte Eingaben der gleichen Testdaten und Prüfungen gegen Programmierkonventionen)
- bessere Konsistenz und Wiederholbarkeit (die gleichen Tests werden beispielsweise in der gleichen Reihenfolge mit der gleichen Häufigkeit durch ein Werkzeug ausgeführt und aus Anforderungen hergeleitet)
- objektive Bewertung durch eine Werkzeugunterstützung (z.B. statische Messungen, Überdeckungsmessungen)
- vereinfachter Zugriff auf Informationen über durchgeführte Tests (z.B. Statistiken und graphische Darstellungen über den Testfortschritt, die Fehlerrate und die Performanz)

Risiken der Verwendung von Testwerkzeugen:

- unrealistische Erwartungen an das Werkzeug (einschließlich Funktionalität und Benutzungsfreundlichkeit)
- unterschätzen der Zeit, der Kosten und des Aufwands für die erstmalige Einführung eines Werkzeugs (einschließlich Training und externe Beratung)
- unterschätzen der Zeit und des Aufwands, um einen signifikanten und anhaltenden Nutzen aus der Anwendung eines Werkzeugs ziehen zu können (einschließlich der Notwendigkeit von Änderungen im Testprozess und der kontinuierlichen Verbesserung in der Art und Weise, wie das Werkzeug verwendet wird)
- unterschätzen des erforderlichen Aufwands für die Wartung der durch das Werkzeug erzeugten Ergebnisse
- blindes Vertrauen in das Werkzeug (Ersatz für einen Testentwurf oder Verwenden automatisierter Tests, wo manuelles Testen geeigneter wäre)
- vernachlässigen der Versionskontrolle von Testgegenständen im Testwerkzeug
- vernachlässigen der Beziehungen und der Interoperabilitätsproblematik zwischen kritischen Werkzeugen, wie Anforderungsmanagementwerkzeugen, Versionskontrollwerkzeugen, Fehler- und Abweichungsmanagementwerkzeugen, Fehlerverfolgungswerkzeugen und Werkzeugen unterschiedlicher Hersteller
- Risiko, dass der Werkzeughersteller den Betrieb einstellt, das Werkzeug vom Markt nimmt, oder das Werkzeug an einen anderen Hersteller verkauft
- mangelhafte Leistungen des Herstellers hinsichtlich Kundenunterstützung, Upgrades und Fehlerbehebungen
- Risiko, dass das Projekt zum Erstellen des kostenfreien bzw. Open-Source-Werkzeugs eingestellt wird
- unvorhergesehene Probleme, z.B. dass eine neue Plattform nicht unterstützt werden kann

6.2.2 Spezielle Betrachtungen zu einigen Werkzeugarten (K1)

Testausführungswerkzeuge

Testausführungswerkzeuge führen Testobjekte aus, indem sie automatisierte Testskripte nutzen. Testwerkzeuge dieser Art erfordern oft einen erheblichen Aufwand, um einen signifikanten Nutzen zu erzielen.

Es mag attraktiv scheinen, die Aktionen eines manuellen Testers aufzuzeichnen, aber diese Vorgehensweise ist nicht auf eine größere Anzahl von automatisierten Testskripten skalierbar. Ein aufgezeichnetes Skript ist eine lineare Repräsentation von spezifischen in das Skript integrierten Daten. Wenn während der Testdurchführung unerwartete Ereignisse auftreten, können Skripte dieser Art instabil werden.

Eine datengetriebene Testvorgehensweise (data-driven approach) trennt die Eingaben (die Testdaten) vom Testfall und legt sie in einem Tabellenblatt ab. Ein generisches Testskript liest die Eingabewerte bei Testausführung aus dem Tabellenblatt. Damit kann das gleiche Testskript mit unterschiedlichen Daten durchgeführt werden. Tester, die sich nicht mit der Skriptsprache auskennen, können dennoch Testdaten für die vordefinierten Testskripte erstellen.

Es gibt weitere Verfahren, die bei den datengetriebenen Verfahren eingesetzt werden. Bei diesen Verfahren werden, anstatt Datenkombinationen hart codiert in einem Tabellenblatt einzugeben, Daten in Echtzeit generiert und der Applikation zur Verfügung gestellt. Das geschieht mit Hilfe von Algorithmen, die auf konfigurierbaren Parametern basieren. Beispielsweise kann ein Werkzeug einen Algorithmus verwenden, der eine zufällige Benutzer-ID erstellt, und damit das Muster wiederholbar wird, ein sogenanntes „Seed“ einsetzt, das die Zufälligkeit steuert.

In einem schlüsselwortgetriebenen Testansatz (keyword-driven approach) enthält ein Tabellenblatt zusätzlich zu den Testdaten Schlüsselwörter (auch Aktionswörter genannt), welche die auszuführenden Aktionen beschreiben. Auch wenn Tester sich nicht mit einer Skriptsprache auskennen, können sie so Tests unter Verwendung von Schlüsselwörtern definieren, die sich auf die zu testende Applikation hin anpassen lassen.

Technische Kenntnisse in den Skriptsprachen werden in allen Ansätzen benötigt (entweder von einem Mitarbeiter in der Rolle des Testers oder durch einen Testautomatisierungsspezialisten).

Unabhängig vom verwendeten skriptbasierten Verfahren müssen die erwarteten Ergebnisse für jeden Test für einen späteren Vergleich bereits abgelegt sein.

Statische Analysewerkzeuge

Statische Analysewerkzeuge dienen der Analyse des Quellcodes und können die Einhaltung von Programmierkonventionen erzwingen. Jedoch ist zu berücksichtigen, dass bei Anwendung eines statischen Analysators für existierenden Quellcode eine riesige Menge Meldungen erzeugt werden können. Wird ein Compiler mit integriertem statischen Analysator oder mit entsprechenden Compiler-Optionen verwendet, so wird die Übersetzung in Objektcode durch die erzeugten Warnmeldungen nicht unterbrochen, jedoch sollten diese Warnmeldungen analysiert werden, um die Wartbarkeit des Quellcodes zu verbessern. Eine Basisimplementierung eines statischen Analysators sollte deshalb die Möglichkeit enthalten, einige dieser Analysen bzw. die entsprechenden Regeln zu deaktivieren.

Testmanagementwerkzeuge

Testmanagementwerkzeuge sollten Schnittstellen zu anderen Werkzeugen oder zu einem Standardtabellenkalkulationsprogramm (z.B. Excel) enthalten, um nützliche Informationen nach den Bedürfnissen der Organisation aufzubereiten.

6.3 Einführung von Testwerkzeugen in eine Organisation(K1)	**15 Minuten**

Begriffe
keine spezifischen Begriffe

Hintergrund

Die wichtigsten Gesichtspunkte bei der Auswahl eines Werkzeugs für eine Organisation sind:

- Bewerten der Reife einer Organisation, Analyse der Stärken und Schwächen, Identifikation von Möglichkeiten für die Verbesserung des Testprozesses, unterstützt durch Testwerkzeuge
- Evaluation gegen klar spezifizierte Anforderungen und objektive Kriterien (für Nutzen und Anwendung)
- Eignungsnachweis (proof-of-concept) durch Nutzen des Werkzeugs während der Evaluierungsphase, um zu verifizieren, ob es mit der zu testenden Software und in der aktuellen Infrastruktur effektiv funktioniert, bzw. um nötige Anpassungen der Infrastruktur zu identifizieren damit eine effektive Nutzung des Werkzeugs möglich ist
- Evaluation der Anbieter (einschließlich Trainingsunterstützung, Support und kommerzielle bzw. vertragliche Aspekte) oder des Dienstleistungsanbieters bei nicht kommerziellen Werkzeugen
- Identifikation der internen Anforderungen für Coaching und Anleitung bei der Anwendung des Werkzeugs
- Evaluation des Schulungsbedarfs, die die Testautomatisierungskenntnisse des aktuellen Testteams berücksichtigt
- Schätzen des Kosten/Nutzen-Verhältnisses, basierend auf einem konkreten Business Case

Die Einführung des ausgewählten Werkzeugs in einer Organisation beginnt mit einem Pilotprojekt, welches folgende Ziele verfolgt:

- Detailliertes Kennenlernen des Werkzeugs
- Bewertung, wie das Werkzeug mit den existierenden Werkzeugen und Prozessen zusammenpasst, festlegen, was ggf. angepasst werden muss
- Entscheidung über die Standardisierung des Werkzeugeinsatzes hinsichtlich Nutzung, Verwaltung, Speicherung und Wartung des Werkzeugs und der vom Werkzeug erzeugten/verwendeten Ergebnisse (z.B. Namenskonventionen für Dateien und Tests, Neuanlage von Bibliotheken und die Festlegung von modularen Testsuiten)
- Bewerten, ob der Nutzen mit vertretbaren Kosten erreicht werden kann

Erfolgsfaktoren der Inbetriebnahme innerhalb einer Organisation sind:

- Das Werkzeug wird schrittweise in der ganzen Organisation in Betrieb genommen.
- Adaptierung und Prozessverbesserung harmonieren mit dem Werkzeug.
- Für neue Anwender werden Trainingsmaßnahmen und Coaching bereitgestellt.
- Es sind Richtlinien für die Werkzeugbenutzung definiert.
- Es gibt Verfahren, um Nutzungsdaten über den derzeitigen Gebrauch zu sammeln.
- Werkzeugverwendung und tatsächlicher Nutzen werden beobachtet.
- Das Testteam erhält Unterstützung für das Werkzeug.
- Es wird ein Erfahrungskatalog erstellt, basierend auf den Erfahrungen aller Teams.

Referenzen
6 Linz, 2012
6.2.2 Buwalda, 2001, Fewster, 1999
6.3 Fewster, 1999

7 Referenzen

7.1 Standards

ISTQB® Standard Glossary of Terms used in Software Testing, Version 2.2

ISTQB®/GTB Standard Glossar der Testbegriffe – Englisch - Deutsch und Deutsch - Englisch Version 2.2

[CMMI] Chrissis, M.B., Konrad, M. and Shrum, S. (2004) *CMMI, Guidelines for Process Integration and Product Improvement*, Addison Wesley: Reading, MA
siehe Kapitel 2.1

[IEEE Std 829-1998] IEEE Std 829™(1998) IEEE Standard for Software Test Documentation
siehe Kapitel 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™ (2008) IEEE Standard for Software Reviews and Audits
siehe Kapitel 3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-2008, Software Life Cycle Processes
siehe Kapitel 2.1

[ISO 9126] ISO/IEC 9126-1:2001, Software engineering – Product quality
siehe Kapitel 2.3

7.2 Bücher

[Beizer, 1990] Beizer, B. (1990) Software Testing Techniques (2nd edition), Van Nostrand Reinhold: Boston
siehe Kapitel 1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6

[Black, 2001] Black, R. (2001) Managing the Testing Process (3rd edition), John Wiley & Sons Inc.: New York
siehe Kapitel 1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6

[Buwalda, 2001] Buwalda, H. et al. (2001) Integrated Test Design and Automation, Addison Wesley: Reading, MA
siehe Kapitel 6.2

[Copeland, 2004] Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood, MA
siehe Kapitel 2.2, 2.3, 4.2, 4.3, 4.4, 4.6

[Craig, 2002] Craig, Rick D. und Jaskiel, Stefan P. (2002) Systematic Software Testing, Artech House: Norwood, MA
siehe Kapitel 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4

[Fewster, 1999] Fewster, M. und Graham, D. (1999) Software Test Automation, Addison Wesley: Reading, MA
siehe Kapitel 6.2, 6.3

[Gilb, 1993]: Gilb, Tom und Graham, Dorothy (1993) Software Inspection, Addison Wesley: Reading, MA
siehe Kapitel 3.2.2, 3.2.4

[Hetzel, 1988] Hetzel, W. (1988) Complete Guide to Software Testing, QED: Wellesley, MA
siehe Kapitel 1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 4.1, 5.1, 5.3

[Kaner, 2002] Kaner, C., Bach, J. und Pettticord, B. (2002) Lessons Learned in Software Testing: A Context-Driven Approach: John Wiley & Sons Inc.: New York
siehe Kapitel 1.1, 4.5, 5.2

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

[Linz, 2012] Linz, T und Spillner, A. (2012) Basiswissen Softwaretest, Aus- und Weiterbildung zum Certified Tester Foundation Level nach ISTQB®-Standard, 5., überarbeitete Auflage, dpunkt.Verlag: Heidelberg
Bildet den Inhalt des deutschsprachigen Lehrplans vollständig ab.

[Myers 2001] Myers, Glenford J. (2001) Methodisches Testen von Programmen, Oldenbourg Verlag: München, Wien
siehe Kapitel 1.2, 1.3, 2.2, 4.3

[van Veenendaal, 2004] van Veenendaal, E. (ed.) (2004) The Testing Practitioner (Chapters 6, 8, 10), UTN Publishers: CN Den Bosch
siehe Kapitel 3.2, 3.3

8 Anhang A – Hintergrundinformation zum Lehrplan

Zur Geschichte dieses Dokuments

Dieses Dokument wurde in den Jahren 2004 bis 2011 durch eine Arbeitsgruppe, deren Mitglieder durch das International Software Testing Qualifications Board (ISTQB®) benannt wurden, erstellt. Es wurde dann durch ein ausgewähltes Reviewteam und anschließend durch Repräsentanten der internationalen Softwaretestwelt geprüft. Die Regeln, welche der Erstellung dieses Dokuments zu Grunde lagen, sind in Anhang C aufgeführt.

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

Ziele der Basiszertifikat-Qualifizierung

- Anerkennung des Testens als einer essentiellen 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 Interessierten, eine international anerkannte Qualifizierung zu erlangen

Zielsetzungen einer internationalen Qualifizierung (angepasst vom ISTQB® Meeting in Sollentuna, November 2001)

- Kenntnisse und Fähigkeiten 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
- mehr Einfluss/Bedeutung durch ein internationales Vorgehen zu erlangen anstelle einer nationaler Strategie
- 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 ISTQB® Basiszertifikat „Softwaretesten“ ist das Interesse der Kandidaten am Softwaretesten. Es empfiehlt sich allerdings für die Kandidaten,

- zumindest ein minimales Hintergrundwissen im Bereich Softwareentwicklung oder Softwaretest zu haben (zum Beispiel sechs Monate Erfahrung als System- oder Abnahmetester oder als Entwickler)
- oder einen Kurs besucht zu haben, der nach dem ISTQB® Standard (durch ein ISTQB®-Mitglieds-Board) 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 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 von der ISTQB erkannten nationalen Testing Board), welche 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 Software Testing Board kontrolliert. Die Pflichten der nationalen Boards sind durch das ISTQB spezifiziert, müssen jedoch durch die einzelnen Länderorganisationen selbst umgesetzt werden. Dazu gehören auch die Akkreditierung von Schulungsanbietern und die Durchführung der Prüfungen.

9 Anhang B – Lernziele/Kognitive Ebenen des Wissens

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

9.1 Taxonomiestufe 1: Kennen (K1)

Der Lernende ruft im Gedächtnis gespeicherte Informationen (z.B. Begriffe, isolierte Fakten, Abfolgen, Prinzipien, Mittel und Wege) ab. Typische beobachtbare Leistungen sind erkennen, nennen, bezeichnen.

Schlüsselworte: sich erinnern, erkennen, wiedergeben, kennen

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

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

9.2 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, erklären, Beispiele geben, begründen, verstehen

Beispiel:
Beschreibt Gemeinsamkeiten und Unterschiede zwischen Integration und Systemtest:

- Gemeinsamkeiten: Mehr als eine Komponente wird getestet und können nicht-funktionale Aspekte umfassen
- Unterschiede: Integrationstest konzentriert sich auf Schnittstellen zwischen und Interaktion von Komponenten; der Systemtest ist auf den Aspekte des ganzen Systems und End-to-End-Verarbeitung ausgerichtet.

Kann erklären, warum Tests so früh wie möglich entworfen werden sollen:

- Fehlerzustände finden, wenn die Behebung billiger ist
- Die wichtigsten Fehlerzustände zuerst finden

9.3 Taxonomiestufe 3: Anwenden (K3)

Der Lernende überträgt erworbenes Wissen auf gegebene neue Situationen oder wendet sie zur Problemlösung an. Typische beobachtbare Leistungen sind ausführen, anwenden, beurteilen, ermitteln, entwerfen, analysieren.

Schlüsselworte: anwenden, einsetzen, ausführen, nutzen, Verfahren verstehen, Verfahren anwenden

Beispiel:

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

9.4 Taxonomiestufe 4: Analysieren (K4)

Der Lernende kann zum besseren Verständnis die Informationen bezüglich eines Vorgehens oder Ablaufs in ihre Einzelbestandteile aufschlüsseln und zwischen Sachverhalten und abgeleiteten Schlussfolgerungen unterscheiden. Typische Anwendungen sind die Analyse eines Dokuments, einer Software oder Projektsituation und der Vorschlag angemessener Maßnahmen zur Problemlösung.

Schlüsselworte: analysieren, organisieren, Zusammenhänge erkennen, integrieren, kurz skizzieren, zergliedern, strukturieren, zuordnen, zerlegen, differenzieren, unterschiedlich behandeln, unterscheiden, fokussieren, auswählen

Beispiel:

- Analysieren Sie Produktrisiken und schlagen Sie vorbeugende oder korrigierende Maßnahmen vor.
- Beschreiben Sie, welche Teile eines Abweichungsberichts einen Sachverhalt darstellen und bei welchen es sich um Schlussfolgerungen aus den Ergebnissen handelt.

Referenz

(für die kognitiven Ebenen von Lernzielen)

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

10 Anhang C – Verwendete Regeln bei der Erstellung des Lehrplans

Die Arbeitsgruppe wendete die unten aufgeführten Regeln auf die Erstellung und Prüfung des Lehrplans an.

10.1 Allgemeine Regeln

SG1. Der Lehrplan soll durch Personen mit null bis sechs (oder mehr) Monaten Testerfahrung verstanden und aufgenommen werden können. (6-MONATE)

SG2. Der Lehrplan ist praxisorientiert und nicht theoretisch. (PRAKTISCH)

SG3. Der Lehrplan soll für den angesprochenen Leserkreis klar und eindeutig sein. (KLAR)

SG4. Der Lehrplan soll für Personen in verschiedenen Ländern verständlich sein und soll leicht in verschiedene Sprachen übersetzt werden können. (ÜBERSETZBAR)

SG5. Das Original des Lehrplans ist in amerikanischem Englisch erstellt. (AMERICAN-ENGLISH)

10.2 Aktualität

SC1. Der Lehrplan berücksichtigt die neuen Entwicklungen im Bereich Testen und gibt die aktuellen, allgemein anerkannten und bewährten Verfahren des Softwaretestens wieder. Der Lehrplan soll alle drei bis fünf Jahre einem Review unterzogen werden. (AKTUELL)

SC2. Der Lehrplan soll möglichst zeitlos erstellt und von Marktströmungen unabhängig sein, um eine Lebensdauer von drei bis fünf Jahren zu ermöglichen. (LEBENSDAUER)

10.3 Lernziele

LO1. Lernziele unterscheiden zwischen Lerninhalten, welche erkannt (Kognitive Stufe K1), verstanden (K2) bzw. auf neue Aufgaben angewendet (K3) werden müssen und die dazu dienen, ein Dokument, eine Software oder Projektsituation im Zusammenhang zu analysieren (K4). (WISSENSEBENE)

LO2. Die Beschreibung der Themen soll konsistent zu den Lernzielen formuliert sein. (LERNZIEL-KONSISTENZ)

LO3 Um Lernziele zu illustrieren, sollen für alle Hauptteile des Lehrplans Beispielprüfungsfragen bereitgestellt werden (LO-PRÜFUNG)

10.4 Gesamtstruktur

ST1. Die Struktur des Lehrplans soll klar sein und Querverweise zwischen einzelnen Teilen, zu Prüfungsfragen und zu anderen relevanten Dokumenten erlauben. (CROSS-REF)

ST2. Inhaltliche Überschneidungen zwischen einzelnen Kapiteln sollen minimiert sein. (ÜBERSCHNEIDUNG)

ST3. Die Abschnitte des Lehrplans sind einheitlich aufgebaut. (STRUKTUR-KONSISTENT)

ST4. Der Lehrplan enthält Version, Freigabedatum und Seitennummer auf jeder Seite. (VERSION)

ST5. Der Lehrplan enthält als Leitfaden die jedem Kapitel zugewiesene Zeit (um die relative Bedeutung jedes Themas widerzuspiegeln). (ZEITDAUER)

Referenzen

SR1. Quellen und Referenzen zu Konzepten im Lehrplan erlauben es den Ausbildungsanbietern, mehr Informationen zum Thema zu finden. (REFS)

SR2. Wo es keine leicht bestimmbaren und klaren Quellen gibt, enthält der Lehrplan mehr Details. Die Definitionen von Begriffen beispielsweise stehen im Glossar, deshalb enthält der Lehrplan ausschließlich den Begriff. (KEIN-REF DETAIL)

Informationsquellen

Die verwendeten Begriffe im Lehrplan sind im ISTQB Glossary of Terms used in Software Testing definiert, das von der ISTQB-Webseite (www.istqb.org) heruntergeladen werden kann.

Eine entsprechende englisch-/deutschsprachige Fassung dieses Glossars mit der Bezeichnung „ISTQB®/GTB Standard Glossar der Testbegriffe" wird auf der Webseite des GTB e.V. (www.german-testing-board.info) bzw. anderer deutschsprachiger Testing Boards zur Verfügung gestellt.

Eine Liste empfohlener Bücher des Softwaretestens wird mit dem Lehrplan publiziert. Die direkten Referenzen befinden sich jeweils im Abschnitt Referenzen.

11 Anhang D – Hinweise für Ausbildungsanbieter

Jeder Überschrift eines Hauptkapitels im Lehrplan ist die vorgegebene Unterrichtsdauer in Minuten zugeordnet. Diese Angabe dient als Leitfaden für die relative, zeitliche Gewichtung der Kapitel in einem akkreditierten Kurs. Zusätzlich legt diese Zahl die minimale Zeitdauer für ein Kapitel fest. Ausbildungsanbieter können mehr Zeit darauf verwenden, und Kandidaten können mehr Zeit als vorgegeben in ihrem Studium und Analyse verwenden. Ein Lehrprogramm eines Kurses kann eine andere Reihenfolge der Kapitel als im Lehrplan enthalten.

Der Lehrplan enthält Referenzen zu gängigen Standards, welche für die Vorbereitung der Ausbildungsunterlagen verwendet werden müssen. Jeder Standard muss in der im Lehrplan referenzierten Version verwendet werden. Andere nicht referenzierte Publikationen, Dokumentvorlagen oder Standards können ebenso verwendet werden, sind aber nicht Gegenstand der Prüfung.

Alle K3 und K4 Lernziele machen praktische Übungen erforderlich, die in den Trainingsunterlagen enthalten sein müssen.

12 Anhang E – Release Notes

Release 2011 1.0.1

Dieses Wartungsrelease enthält die Anpassungen an das ab 19.04.2013 gültige Glossar 2.2

1. Komponententestrahmen -> Unittest-Framework
2. Ausgangskriterien -> Endekriterien
3. Strukturbasierter Test -> strukturorientierter Test
4. Akteur (= actor)

Sowie Korrekturen in der Übersetzung der Kapitel 1.4.1 und 5.1.2 und von einem bereits in Glossar 2.1 verwendeten Begriff:

1. (Strukturbasierte) Techniken -> (strukturorientierte) Verfahren (= structure-based technique)

Des Weiteren wurde eine deutschsprachige Literaturreferenz aktualisiert (Linz 2010 -> Linz 2012)

Release 2011

Das Wartungsrelease 2011 enthält folgende Änderungen:

1. Anpassung der Copyright Formulierung an das englische Original
2. Der Begriff Working Party wurde durch Working Group ersetzt
3. Kapitel 1.6: Die kognitive Stufe für dieses Kapitel wurde entfernt, da es nicht die Absicht war, ein Lernziel für die ethischen Leitlinien zu definieren
4. Kapitel 2.2.2 Aufnahme von Subsystemen als Testobjekt
5. Kapitel 2.2.2 Der Begriff Fehlerwirkung war nicht korrekt in Verbindung mit "… die Isolation von Fehlerwirkungen in einer spezifischen Komponente…".Aus diesem Grund wurde er durch den Begriff „Fehlerzustände" ersetzt.
6. Kapitel 2.3.4: Die Beschreibung des Begriffs "Debugging" wurde dem ISTQB Glossar 2.1 angepasst
7. Kapitel 2.4 Das Wort „umfassend" wurde aus "ein umfassendes Regressionstesten entfernt…" da "umfassend", wie im nächsten Satz beschrieben, von der Änderung (Umfang, Risiko etc.) abhängt.
8. Kapitel 3.2.1: Die Aktivität "Prüfen der Eingangskriterien" wurde der Hauptaktivität Planen zugeordnet.
9. Kapitel 4: Das Wort "entwickelt" wurde durch "definiert" ersetzt, da Testfälle definiert und nicht entwickelt werden.
10. Kapitel 4.2: Textänderung, um zu verdeutlichen, wie Black-Box- und White-Box-Tests in Verbindung mit erfahrungsbasierten Testentwurfsverfahren genutzt werden können.
11. Kapitel 4.3.5 Textänderung "… zwischen den Aktoren, einschließlich Anwendern und Systemen..." in " … zwischen den Aktoren (Anwender oder Systeme), … ".
12. Kapitel 4.3.5 alternative Pfade ersetzt durch alternative Szenarien
13. Kapitel 4.4.2: Um den Begriff Zweigtest in Kapitel 4.4 zu verdeutlichen, wurde ein Satz, der den Fokus des Zweigtests verdeutlichen soll, verändert.
14. Kapitel 6.1: Überschrift "6.1.1 Sinn und Zweck einer Werkzeugunterstützung für das Testen (K2)" wurde ersetzt durch "6.1.1Werkzeugunterstützung für das Testen (K2)".
15. Kapitel 7 / Bücher: [Black,2001] Die 2. Auflage wurde durch die 3. Auflage ersetzt.
16. Anhang D: Kapitel, die Übungen erfordern wurden durch die generische Anforderung ersetzt, dass Übungen für alle Lernziele K3 und höher erforderlich sind. Diese Anforderung wurde im ISTQB Akkreditierungsprozess (Version 1.26) definiert.

Release 2010

1. Änderungen der Lernziele (LO) enthalten einige Klarstellungen
    a. Bei folgenden Lernzielen wurde die Formulierung geändert (Inhalt und Taxonomiestufe blieben unverändert): LO-1.2.2,LO-1.3.1 (inkl. Kapitelname),LO-1.4.1, LO-1.5.1,LO-2.1.1, LO-2.1.2, LO-2.1.3, LO-2.3.5, LO-2-4-2, LO-4.1.3, LO-4.2.1, LO-4.2.2, LO-4.3.1,LO-4.3.2, LO-4.3.3,LO-4.4.1, LO-4.4.2, LO-4.4.3, LO-4.6.1,LO-5.1.1, LO-5.1.2, LO-5.1.4, LO-5.2.1, LO-5.2.3, LO-5.2.6, LO-5.3.1, LO-5.3.2, LO-5.5.2, LO-6.1.1, LO-6.3.2
    b. Bei folgenden Lernzielen gab es Korrekturen gemäß dem aktuellen Glossar: LO-1.1.2, LO-1.1.5, LO-2.4.3, LO-3.3.3, LO-4.1.1, LO-4.1.2, LO-4.1.4, LO-4.3.2, LO-4.4.2, LO-4.5.1, LO-5.1.4, LO-5.2.2, LO-5.2.9, LO-5.3.2, LO-5.3.3, LO-5.6.1, LO-5.6.2,
    c. LO-1.1.5 wurde umformuliert und der höheren Taxonomiestufe K2 zugeordnet, weil ein Vergleich der Begriffe in Zusammenhang mit Fehlerzuständen erwarten werden kann.
    d. LO-1.2.3 (K2) wurde hinzugefügt. Der Inhalt war bereits in der Lehrplanausgabe 2007 abgedeckt.
    e. LO-3.1.3 (K2) kombiniert nun den Inhalt von LO-3.1.3 und LO-3.1.4.
    f. LO-3.1.4 wurde aus der Lehrplanausgabe 2010 entfernt, weil es mit LO-3.1.3 teilweise redundant war.
    g. LO-3.2.1 wurde aus Konsistenzgründen zur Lehrplanausgabe 2010 umformuliert
    h. LO-3.3.2 wurde verändert und die Taxonomiestufe von K1 auf K2 erhöht, so dass Übereinstimmung mit LO-3.1.2 gegeben ist.
    i. LO 4.4.4 wurde zur Verdeutlichung verändert und von K3 auf K4 gesetzt. Begründung: Das Lernziel wurde bereits im Stil einer K4 Frage geschrieben.
    j. LO-5.2.9 wurde um Eingangskriterien erweitert.
    k. LO-6.1.2 wurde aus der Lehrplanausgabe 2010 entfernt, und durch LO-6.1.3 (K2) ersetzt. Die ID wurde nicht neu vergeben.
    l. LO-6.2.2 Lernziel um statische Analyse und Testmanagementwerkzeuge erweitert
2. Konsistente Verwendung der Begriffe gemäß der Definition im Glossar: Ausgangskriterien (statt Testendekriterien), Eingabe- und Ausgabewerte (statt Ein- u. Ausgabedaten), Testabschlussbericht, Testablauf (statt Testprozedur), Testablaufspezifikation (statt Testvorgehensspezifikation), Testbedingung (statt Testziel Kapitel 1.4), Testmanager als führender Begriff, Testentwurf, Testüberdeckung (statt Testabdeckung), Fehlerzustand, Testkonzept, Verifizierung (statt Verifikation), vorausgesagtes Ergebnis (statt erwartetes Ergebnis).
3. Kapitel 1.4 enthält nun das Konzept der Rückverfolgbarkeit zwischen Testbasis und Testfällen.
4. Kapitel 2.x enthält nun Testobjekte und Testbasis und berücksichtigt Konfigurationsdaten sowie die Themen Datenqualität, Migrations-/Konvertierungstests
5. Kapitel 3.2.1 wurde bzgl. der Aktivitäten eines formalen Reviews etwas detailliert.
6. Umbenennung des Kapitel 4 Titels gemäß Glossar und Erhöhung der Taxonomiestufe auf K4
7. Kapitel 4.3.2 „Für Tests nutzt man den exakten Grenzwert und die beiden benachbarten Werte.“ entfernt. Er spiegelt korrekt eine Lehrmeinung wider, steht so aber nicht im englischsprachigen Lehrplan.
8. Die Aspekte Datenqualität und Testen wurden an mehreren Stellen des Lehrplans mit aufgenommen: Datenqualität und Risiko in den Abschnitten 2.2, 5.5, 6.1.8.
9. Die Taxonomiestufe des Kapitels 5.2.2 Testplanungsaktivitäten wurde von K2 auf K3 verändert und dadurch die Taxonomiestufe des Kapitel 5.2 ebenfalls angehoben.
10. Das Thema Eingangskriterien wurde als neues Unterkapitel 5.2.3 eingefügt. Grund: Übereinstimmung mit Ausgangskriterien (-> Eingangskriterien wurde bei LO-5.2.9 hinzugefügt).
11. Das Kapitel 5.2.6 Teststrategie, Testvorgehensweisen wurde überarbeitet.
12. Die Liste möglicher Produkt- u. Projektrisiken in den Kapiteln 5.5.1 u. 5.5.2 Liste wurde überarbeitet.

13. Kapitel6.1 wurde gekürzt, da die Beschreibung der Werkzeuge zu umfassend war für eine Unterrichtszeit von 45 Minuten. Zum Thema Sinn und Zweck einer Werkzeugunterstützung wurde ein neues Kapitel eingefügt.
14. IEEE Std 829-2008 wurde inzwischen herausgegeben. Diese neue Ausgabe wird in der vorliegenden Version des Lehrplans noch nicht berücksichtigt. Im Abschnitt 5.2 geht es um das Dokument „Mastertestkonzept“. Der Inhalt des Mastertestkonzepts wird dahingehend abgedeckt, dass das Dokument „Testkonzept“ mehrere Stufen der Testplanung abdeckt. Für die Teststufen können „Testkonzepte“ erstellt werden, und zusätzlich kann auf Projektebene ein „Testkonzept“ erstellt werden, das mehrere Teststufen abdeckt. Das „Testkonzept“ auf Projektebene wird in diesem Syllabus und im ISTQB-Glossar als „Mastertestkonzept“ bezeichnet.
15. Die ethischen Leitlinien wurden aus dem CTAL in den CTFL übernommen.

Release 2007

1. Lernziele nummeriert (LO = Learning objective), um sie leichter zu merken
2. Wortlaut folgender Lernziele geändert (Inhalt und Taxonomiestufe blieben unverändert): LO-1.5.1,LO-2.3, LO-3-3-1, LO-4.1.3, LO-4.3.1, LO-5.2.4
3. LO-3.1.4 in Kapitel 3.3. verschoben
4. Lernziel “Einen Testausführungsplan für einen vorgegebenen Satz Testfälle schreiben und dabei Priorisierung und technische sowie fachliche Abhängigkeiten berücksichtigen” von Kapitel 4.1 in Kapitel 5.2 verschoben (= LO-5.2.5)
5. Lernziel LO-5.2.3 „Unterscheiden zwischen zwei konzeptionell verschiedenen Testvorgehensweisen wie analytisch, modellbasiert, methodisch, prozess-/standardkonform, dynamisch/heuristisch, beratend oder wiederverwendungsorientiert (K2)“ aufgenommen, da Kapitel 5.2 es behandelt und das Lernziel bisher fehlte
6. LO-5.2.6 “Testvorbereitungs- und Testdurchführungsaufgaben, die geplant werden müssen, auflisten.(K1)” hinzugefügt. Bisher war das Thema Teil von Kapitel 1.4 und durch Lernziel LO-1.4.1abgedeckt
7. Kapitel 4.1 von „Festlegen von Testkriterien und Entwurf von Testfällen“ in “Testentwicklungsprozess” umbenannt. Taxonomiestufe von 3 auf 2 verändert.
8. Kapitel 1.4.1 Details, die in Kapitel 5 behandelt werden entfernt
9. Kapitel 2.1behandelt nun das sequentielle und iterativ-inkrementelle Entwicklungsmodell
10. Begriffe werden nun nur noch in dem Kapitel, in welchem sie zuerst auftauchen referenziert. Aus den folgenden wurden sie entfernt.
11. Begriffe, die zu Beginn der Kapitel benannt werden, wurden alle in den Singular gesetzt
12. Neue Begriffe: Testgrundsatzrichtlinie/Test Policy (1.4), Testprozedur (1.4), Unabhängigkeit (1.5),iterativ-inkrementelles Entwicklungsmodelle (2.1), Statische Prüftechnik/Test (3.1) (ersetzt hier Statische Analyse (siehe 3.3)), Testausführungsplan (4.1), Testentwurf (4.1), Fault Attack (4.5),Abweichungsmanagement / Fehlermanagement (5.6)
13. Folgende Begriffe wurden entfernt: Software, Testen (1.1), Code (1.2) Entwicklung (von Software),unabhängiges Testen (1.5), vertraglicher Abnahmetest (2.2), operational u. regulativer Abnahmetest(2.2), Einzug (2.4), Modifikation (2.4), Migration (2.4), Kick-Off (3.2), Review Meeting (3.2),Review Prozess (3.2)
14. In Kapitel 6.1.5 Kennzeichnung für Entwicklerwerkzeug (E) bei Testdatengeneratoren und -editoren entfernt

15. Formulierungen in fast allen Kapiteln geändert

13 Index