Certified Tester

Foundation Level Syllabus

Version 2007

International Software Testing Qualifications Board

Deutschsprachige Ausgabe. Herausgegeben durch
German Testing Board e.V. & Swiss Testing Board

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

Änderungsübersicht deutschsprachige Ausgabe

Version	Datum	Bemerkung
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 Party Foundation Level (Ausgabe 2007): Thomas Müller (Leiter), Dorothy Graham, Debra Friedenberg, and Erik van Veendendal. Das Kernteam dankt dem Reviewkernteam (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, und Wonil Kwon) und allen nationalen Boards für die Verbesserungsvorschläge der aktuellen Version des Lehrplans.

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).

Speziell danken die Autoren der englischen Originalausgabe (Ausgabe 2005): (Österreich) Anastasios Kyriakopoulos, (Dänemark) Klaus Olsen, Christine Rosenbeck-Larsen, (Deutschland) Matthias Daigl, Uwe Hehn, Tilo Linz, Horst Pohlmann, Ina Schieferdecker, Sabine Uhde, Stephanie Ulrich, (Indien) Vipul Kocher, (Israel) Shmuel Knishinsky, Ester Zabar, (Schweden) Anders Claesson, Mattias Nordin, Ingvar Nordström, Stefan Ohlsson, Kennet Osbjer, Ingela Skytte, Klaus Zeuge, (Schweiz) Armin Born, Sandra Harries, Silvio Moser, Reto Müller, Joerg Pietzsch, (Großbritannien) Aran Ebbett, Isabel Evans, Julie Gardiner, Andrew Goslin, Brian Hambling, James Lyndsay, Helen Moore, Peter Morgan, Trevor Newton, Angelina Samaroo, Shane Saunders, Mike Smith, Richard Taylor, Neil Thompson, Pete Williams, (USA) Dale Perry.

Die Arbeitsgruppenmitglieder der deutschsprachigen Übersetzung (Ausgabe 2005) 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: Petra Bukowski (GTB), Tilo Linz (GTB), Armin Metzger (GTB), Reto Müller (STB), Thomas Müller (STB), Horst Pohlmann (GTB), Sabine Uhde (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 International Software Testing Qualifications Board stellt den Lehrplan den **nationalen Testing Boards** zur Verfügung, damit diese in ihrem Zuständigkeitsbereich Ausbildungsanbieter akkreditieren und Prüfungsfragen in der jeweiligen Landessprache erarbeiten und Prüfungsinstitutionen zur Verfügung stellen können. An Hand des Lehrplanes 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. Dies 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: wiedergeben, nennen, aufzählen, bezeichnen, erkennen
- K2: beschreiben, darstellen, erläutern, erklären, übertragen, zusammenfassen, identifizieren, ergänzen, interpretieren, schlussfolgern; verstehen, vergleichen, begründen
- K3: anwenden, ausführen, analysieren, ermitteln, vorschlagen, entwerfen, entwickeln, formulieren, beurteilen, unterscheiden

Weitere Details und Beispiele über Lernziele sind in Anhang A zu finden.

Alle Begriffe, die im Absatz direkt unter der Überschrift unter "Begriffe" genannt werden, sollen wiedergegeben werden können (K1), auch wenn dieses in den Lernzielen nicht explizit genannt wird. 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) abgelegt werden. 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

Ausbildungsanbieter, deren Ausbildungsunterlagen entsprechend diesem Lehrplan aufgebaut sind, müssen durch ein, vom ISTQB® anerkanntes, nationales Testing Board akkreditiert werden. Die Akk-

reditierungsrichtlinien können bei diesem nationalen Board (in Deutschland: German Testing Board e.V.) 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, beinhaltet 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, inklusive der 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 Lehrziele des Foundation Level relevant ist.

Lehrplanaufbau

Der Lehrplan besteht aus 6 Hauptkapiteln. Jeder Haupttitel eines Kapitels zeigt die Lernzielkategorie, 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 Taxonomiestufen) und K2 (aber nicht K3) erwartet wird 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 Ziele legen fest, was Sie nach Beendigung des jeweiligen Moduls gelernt haben sollten.

1.1 Warum sind Softwaretests notwendig? (K2)

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

LO-1.1.2 Zwischen der Ursache eines Fehlers und seinen Auswirkungen unterscheiden. (K2)

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

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

LO-1.1.5 Die Begriffe Fehlhandlung, Fehlerzustand / Defekt, Fehlerwirkung / Ausfall und die zugehörigen Definitionen kennen. (K1)

1.2 Was ist Softwaretesten? (K2)

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

LO-1.2.2 Testen in Softwareentwicklung, -wartung und –betrieb als Mittel zum Finden von Fehler, zum Vermitteln von Vertrauen und Information und zum Verhindern von Fehlern beschreiben. (K2)

1.3 Allgemeine Grundsätze des Softwaretestens (K2)

LO-1.3.1 Die fundamentalen Grundsätze des Softwaretestens erklären. (K2)

1.4 Fundamentaler Testprozess (K1)

LO-1.4.1 Die fundamentalen Aktivitäten von der Planung bis zum Testabschluss und die Hauptaufgaben während jeder Aktivität kennen. (K1)

1.5 Die Psychologie des Testens (K2)

LO-1.5.1 Wissen, dass der Testerfolg von psychologischen Faktoren beeinflusst wird (K1):
- klare Zielsetzungen für den Test bestimmen die Effektivität der Tester
- Blindheit gegenüber eigenen Fehlern
- Wichtigkeit einer sachlichen Kommunikation und Rückmeldung bezüglich Probleme.

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

1.1 Warum sind Softwaretests notwendig (K2) — 20 Minuten

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

1.1.1 Softwaresystemzusammenhang (K1)

Softwaresysteme spielen in einem zunehmend großen Teil des Lebens eine wichtige Rolle, von Business-Software (z.B. Bankanwendungen) bis zu Industrieprodukten (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 Defekt (Fehlerzustand) im Code einer Software oder eines Systems oder in einem Dokument verursacht. Wenn der Defekt im Code ausgeführt wird, wird das System nicht das tun, was es tun sollte (oder etwas tun, was es nicht tun sollte) und dabei eine Fehlerwirkung hervorrufen oder ausfallen. Defekte in Software, Systemen oder Dokumenten können, müssen aber nicht zu einer Fehlerwirkung oder einem Ausfall führen.

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

Fehlerwirkungen können aber auch durch Umgebungsbedingungen hervorgerufen werden. Strahlung, elektromagnetische Felder oder Schmutz können z.B. Defekte 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 beim Betrieb von Software (K2)

Intensives Testen von Systemen und Dokumentation kann helfen das Risiko, dass Probleme im operativen Betrieb auftreten, zu reduzieren. Und es kann dazu beitragen, die Qualität des Softwaresystems zu erhöhen, indem Fehler 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, ausgedrückt durch die Anzahl gefundener Fehler. Dies gilt sowohl für funktionale als auch für nicht-funktionale Anforderungen und Merkmale (z.B. Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit); für weitere Informationen zum Thema nicht-funktionales Testen siehe Kapitel 2; für weitere Information über Softwarequalitätsmerkmale finden Sie in der Norm ‘Software Engineering – Software Product Quality’ (ISO 9126).

Wenn wenige oder keine Fehler 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 Fehler findet und diese Fehler behoben werden, steigt die Qualität des Softwaresystems.

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

Testen sollte als eine Qualitätssicherungsmaßnahme in den Entwicklungsprozess integriert sein (z.B. neben Entwicklungsstandards, Training und Fehlerursachenanalyse).

1.1.5 Wie viel Testaufwand ist notwendig? (K2)

Um zu entscheiden, wie viel Test notwendig ist, sollte das Risikoniveau berücksichtigt werden. Dies schließt sowohl technische Risiken, wirtschaftliche Risiken und Projektrisiken, 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.

Weitere Testaktivitäten sind vor und nach der Testdurchführung angesiedelt. Dazu gehören: Planung und Steuerung der Tests, Auswahl der Testbedingungen, Testfallspezifikation und Überprüfen der Resultate, Auswertung der Testendekriterien, Berichten über den Testprozess und das zu testende System, Abschlussarbeiten (z.B. nachdem eine Testphase abgeschlossen ist). Zum Testen zählt ebenfalls das Prüfen von Dokumenten (Quellcode inbegriffen) und die statische Analyse.

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 verfolgt verschiedene Ziele:
- Das Aufdecken von Fehlern.
- Das Erzeugen von Vertrauen und Informationen bezüglich des Qualitätsniveaus des Systems.
- Fehlern vorzubeugen.

Das konsequente Erstellen von Tests schon früh im Lebenszyklus (das Prüfen der Testbasis durch den Testentwurf) kann Fehler im Programmcode verhindern. Reviews von Dokumenten (z.B. Anforderungsspezifikation) 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 Mängel zu beheben), um die Beteiligten über das Risiko einer Systemfreigabe zu einem bestimmten Zeitpunkt zu informieren. Wartungstests beinhalten oft Tests, die sicherstellen sollen, dass durch die Änderung der Software keine neuen Fehler eingebaut wurden. Beim betrieblichen Test (orientiert an Nutzungsprofilen) kann das Hauptziel sein, ein System hinsichtlich Ausprägungen wie Zuverlässigkeit oder Verfügbarkeit zu bewerten.

Debugging und Testen sind verschiedene Dinge. Testen kann Fehlerwirkungen zeigen, die durch Fehler verursacht werden. Debugging ist eine Entwicklungsaktivität, die die Ursache eines Fehlers identifiziert, den Code korrigiert und überprüft, dass der Defekt korrekt behoben wurde. Anschließende Fehlernachtests durch einen Tester stellen sicher, dass die Lösung wirklich die Fehlerwirkung behoben hat. Die Verantwortung für jede der beiden Aktivitäten ist grundverschieden: Testen erfolgt durch den Tester und Debuggen durch den Entwickler.

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

1.3 Allgemeine Grundsätze des Softwaretestens (K2)	**35 Minuten**

Begriffe
Vollständiger 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 Fehlern
Mit Testen wird das Vorhandensein von Fehlerwirkungen 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 Fehlerwirkungen im Test aufgezeigt wurden, ist dies 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
Testaktivitäten sollen im System- oder Softwarelebenszyklus so früh wie möglich beginnen und definierte Ziele verfolgen. Durch frühzeitiges Prüfen werden Fehler frühzeitig erkannt.

Grundsatz 4: Häufung von Fehlern
Oft finden sich in nur wenigen Teilen eines Testobjekts die meisten Fehlerursachen, eine Gleichverteilung der Fehlerzustände im Testobjekt ist in der Regel nicht gegeben. Dort, wo Fehlerwirkungen nachgewiesen wurden, finden sich meist noch weitere. Beim Testen muss flexibel auf diesen Umstand reagiert werden.

Grundsatz 5: Wiederholungen haben keine Wirksamkeit
Wiederholungen der immer gleichen Testfälle führen zu keinen neuen Erkenntnissen. Damit die Effektivität der Tests nicht absinkt, 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 Fehlerwirkungen nachgewiesen.

Grundsatz 6: Testen ist abhängig vom Umfeld
Je nach Einsatzgebiet und Umfeld des zu prüfenden Systems ist das Testen anzupassen. Keine zwei Systeme sind auf die exakt gleiche Art und Weise zu testen. Intensität des Testens, Definition der Testendekriterien usw. sind bei jedem System entsprechend seines Einsatzumfeldes festzulegen. Sicherheitskritische Systeme verlangen andere Prüfungen als beispielsweise E-Commerce-Systeme.

Grundsatz 7: Trugschluss: Keine Fehler bedeutet ein brauchbares System
Fehlerwirkungen zu finden und zu beseitigen garantiert noch lange nicht, dass das System auch den Vorstellungen und Erwartungen der Nutzer entspricht. Frühzeitige Einbeziehung der späteren Nutzer in den Entwicklungsprozess und die Nutzung von Prototyping sind vorbeugende Maßnahmen zur Vermeidung des Problems.

1.4 Fundamentaler Testprozess (K1) — 35 Minuten

Begriffe
Abweichung / Fehlermeldung/ Störung, Fehlernachtest, Regressionstest, Testbasis, Testbedingung, Testbericht, Testdaten, Testdurchführung, Testendekriterium, Testgrundsatzrichtlinie/ Test Policy, Testkonzept, Testmittel, Testprotokoll, Testprozedur, Teststrategie, Testsuite, Testüberdeckung

Hintergrund

Die Testdurchführung ist der sichtbarste Teil des Testens. Aber um effektiv und effizient zu sein, ist es darüber hinaus notwendig, die Tests zu planen, Testfälle zu spezifizieren und die Testdurchführung vorzubereiten, sowie den Teststatus zu bewerten.

Der fundamentale Testprozess besteht aus den folgenden Hauptaktivitäten:

- Testplanung und Steuerung
- Testanalyse und Testdesign
- Testrealisierung und Testdurchführung
- Testauswertung und Bericht
- Abschluss der Testaktivitäten

Auch wenn hier logisch sequentiell aufgelistet, können all diese Testprozessaktivitäten in der Praxis zeitlich überlappend oder parallel stattfinden.

1.4.1 Testplanung und Steuerung (K1)

Zur Testplanung gehören folgende Aktivitäten: Überprüfung und Festlegung des Aufgabenumfangs des Testens für das spezielle Projekt, 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 gegen den Plan einschließlich eventueller Abweichungen vom Plan zu überprüfen und den Status aufzuzeigen, sowie ggf. 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 in konkrete Testbedingungen und Testfälle detailliert werden.

Dies umfasst die folgenden Hauptaufgaben:

- Review der Testbasis (z.B. Anforderungen, Architektur, Design, Schnittstellen)
- Bewertung der Testbarkeit von Testbasis und Testobjekten
- Identifizierung und Priorisierung der Testziele auf Grundlage der Testobjektanalyse, der Spezifikation, des Verhaltens und der Struktur des Testobjekts
- Testentwurf (Design) und Priorisierung von Testfällen
- Identifizierung benötigter Testdaten, um Definition von Testbedingungen und Testfälle zu unterstützen
- Entwurf des Testumgebungsaufbaus und Identifikation der benötigten Infrastruktur und Werkzeuge

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, Testprozeduren 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:

- Entwicklung/ Ableitung und Priorisierung von Testfällen
- Erstellung und Priorisierung der Testprozeduren, Erstellung der Testdaten, der Testszenarien und optional Vorbereitung der Testrahmen und Entwicklung von Skripten zur Testautomatisierung
- Gruppierung von Testprozeduren, um die Testdurchführung möglichst effizient zu gestalten
- Kontrolle, ob das Testsystem korrekt aufgesetzt wurde und Sicherstellung der richtigen Konfigurationen
- Ausführung der Testprozeduren (manuell oder automatisiert) unter Einhaltung des Testplans (Reihenfolge, Testsuiten etc.)
- Protokollierung der Testergebnisse und Dokumentation der genauen Version des jeweiligen Testobjektes und der eingesetzten Testwerkzeugen und Testmittel
- Vergleich der Ist-Ergebnisse mit den erwarteten Soll-Ergebnissen
- Um den Grund eines Problems festzustellen (z.B. Fehler im Code, in spezifizierten Testdaten, im Testdokument oder ein Fehler bei der Eingabe des Testfalles) werden gefundene Fehlerwirkungen oder Abweichungen festgehalten und analysiert.
- 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) Fehler freigelegt wurden.

1.4.4 Testauswertung und Bericht (K1)

Testauswertung und -bericht sind die Aufgaben, durch die Ergebnisse der Testdurchführung und die definierten Ziele der Tests verglichen werden.
Diese Phase sollte in jeder Teststufe abgehandelt werden.

Testauswertung und -bericht hat folgende Hauptaufgaben:

- Auswertung der Testprotokolle in Hinblick auf die im Testkonzept festgelegten Testendekriterien
- Entscheidung, ob mehr Tests durchgeführt oder, ob die festgelegten Testendekriterien angepasst werden müssen
- Erstellung des Testberichts 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); 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-/Abweichungsmeldungen oder Erstellung von Änderungsanforderungen für weiter bestehende Fehler/ Abweichungen und die 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“ für spätere Projekte und Verbesserung der Testreife.

1.5 Die Psychologie des Testens (K2) — 35 Minuten

Begriff
Error Guessing, Unabhängigkeit

Hintergrund

Die Haltung bzw. Einstellung, die während der Testdurchführung und der Reviewphase benötigt wird, unterscheidet sich von derjenigen während 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 Aufwandes jedoch noch weitere Vorteile: eine unabhängige Sichtweise von geschulten, professionellen Testexperten. Unabhängiges Testen kann in jeder Teststufe angewendet werden.

Ein gewisser Grad an Unabhängigkeit ist bei der Fehlersuche (Betriebsblindheit) oft effizienter.
Unabhängigkeit ist allerdings auf keinen Fall ein Ersatz für Erfahrung (Vertrautheit) mit der Software, ebenso können Entwickler effizient viele Fehler in ihrem eigenen Code finden.
Folgende Stufen der Unabhängigkeit 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, welche ihnen vom Management oder anderen Stakeholdern vorgegeben werden, anzupassen. So versucht ein Tester Fehler in der Software zu finden, oder, wenn das Ziel vorgegeben wurde, die Richtigkeit der Software zu bestätigen, auch das Gegenteil. Daher ist es sehr wichtig das Ziel vom Test klar aufzuzeigen.

Das Aufdecken von Fehlern 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 beim Error Guessing (Vermuten von Fehlern) zurückgreifen kann.

Wenn Fehler oder Fehlverhalten positiv kommuniziert werden, können Schwierigkeiten zwischen Testern und Entwicklern, Analysten und Designern vermieden werden. Dies gilt sowohl für die Reviewphase, als auch für die Test-Phase.

Tester und Testleiter müssen eine ausgeprägte Kontaktfähigkeit und gute kommunikative Fähigkeiten besitzen, um sachbezogene Informationen über gefundene Fehler, Fortschritte und Risiken austauschen zu können. Einem Autor eines Dokumentes oder einem Entwickler kann die Information über den gefundenen Fehler helfen, seine Qualifikation zu verbessern. Werden Fehler während der Testphase gefunden und behoben, so spart dies 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, um die Beziehung und die Kommunikation zwischen den Testern und ihrem Umfeld zu verbessern:

- Beginne mit der Zusammenarbeit anstelle zu streiten – erinnere jeden an das zentrale Ziel: eine bessere Qualität der Software!
- Kommuniziere gefundene Fehler eines Produktes neutral, sachbezogen und vermeide Kritik an der verantwortlichen Person.
- Versuche, das Verhalten und die Gefühle der anderen Person zu verstehen.
- Stelle sicher, dass ihr euch verstanden habt (dass die andere Person versteht, was du aussagen wolltest und anders herum).

Referenzen

1 Linz, 2005
1.1.5 Black, 2001, Kaner, 2002
1.2 Beizer, 1990, Black, 2001, Myers 1982
1.3 Beizer, 1990, Hetzel, 1988, Myers 1982, Linz, 2005
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 Ziele legen fest, was Sie nach Beendigung des jeweiligen Moduls gelernt haben sollten.

2.1 Softwareentwicklungsmodelle (K2)

LO-2.1.1 Beziehungen zwischen Entwicklungs- und Testaktivitäten und Arbeitsergebnissen im Entwicklungslebenszyklus verstehen, Beispiele basierend auf Projekt- und Produkteigenschaften und Zusammenhängen nennen.(K2)

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

LO-2.1.3 Gründe für verschiedene Teststufen und Eigenschaften “guter” Tests in beliebigen Entwicklungslebenszyklen nennen. (K1)

2.2 Teststufen (K2)

LO-2.2.1 Verschiedene Teststufen vergleichen: 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 Beispielen vergleichen. (K2)

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

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

LO-2.3.4 Testarten basierend auf der Analyse der Struktur oder Architektur des Software-Systems identifizieren und beschreiben. (K2)

LO-2.3.5 Ziele eines Fehlernachtests und eines Regressionstests beschreiben. (K2)

2.4 Wartungstests (K2)

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

LO-2.4.2 Die Beweggründe für Wartungstests (also Modifikation, Migration und Einzug /Außerbetriebnahme/Ablösung) der Software) identifizieren. (K1)

LO-2.4.3 Die Rolle von Regressionstests und der Beurteilung der Auswirkung von Änderungen in der Softwarewartung beschreiben. (K2)

2.1 Softwareentwicklungsmodelle (K2) — 20 Minuten

Begriffe
Iterativ-inkrementelles Entwicklungsmodell, Kommerzielle Standardsoftware (COTS - Commercial off the shelf), V-Modell (allgemeines), Validierung, Verifikation

Hintergrund
Testen existiert nicht isoliert; 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 zu den vier 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 z.B. Geschäftsvorfälle, Anforderungsspezifikationen, Entwurfsdokumente und Code), die während der Entwicklung entstehen, sind oftmals die Basis für Tests in einer oder mehreren Stufen.
Referenzen für generische Arbeitspapiere können unter anderem in folgenden Standards gefunden werden:

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

Verifikation 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 kürzerer Entwicklungszyklen durchlaufen. Beispiele hierfür sind Prototyping, Rapid Application Development (RAD), der Rational Unified Process (RUP) und agile Entwicklungsmodelle. Das wachsende System kann auf verschiedenen Stufen als Teil der Entwicklung getestet werden. Eine Erweiterung, die zu den anderen vorangehenden Erweiterungen hinzugefügt wird, ergibt so ein wachsendes unvollständiges System, das ebenso getestet werden sollte. Regressionstests haben daher bei allen Iterationen nach dem ersten Zyklus eine zunehmende Bedeutung. Verifikation 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 Zielumgebung) und den Abnahmetest (beispielsweise funktionale und/oder nicht-funktionale Tests und Nutzer- und/oder betriebliche Tests) durchführen.

2.2 Teststufen (K2) — 40 Minuten

Begriffe
Alpha-Test, Anwender-Abnahmetest, Beta-Test, Feldtest, funktionale Anforderung, Integration, Integrationstest, Komponententest (auch bekannt als Unit-, Modul- oder Programmtest) nicht-funktionale Anforderung, Platzhalter (auch Stub), Robustheitstest, Systemtest, testgetriebene Entwicklung, Teststufe, Testtreiber, 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.

2.2.1 Komponententest (K2)

Der Komponententest hat das Funktionieren von Software zum Ziel, die separat getestet werden kann (z.B. Module, Programme, Objekte, Klassen, etc.), sie zu prüfen und darin vorhandene Fehler zu finden. In Abhängigkeit von Lebenszyklus und System kann dies durch Isolierung vom Rest des Systems erreicht werden. Dabei gelangen dann meist Platzhalter, Treiber und Simulatoren zur Anwendung.

Komponententest kann das Testen funktionaler wie auch nicht-funktionaler Aspekte beinhalten, so etwa das Testen des Umgangs mit den Ressourcen (z.B. Speicherengpässe), Robustheitstest oder auch struktureller Test (z.B. Zweigüberdeckung). Testfälle werden von Entwicklungsdokumenten, wie einer Komponentenspezifikation, dem Softwareentwurf oder dem Datenmodell abgeleitet.

Meistens steht beim Komponententest den Testern der Quellcode wie auch Unterstützung aus der Entwicklungsumgebung, wie zum Beispiel eine spezielle Komponententestumgebung (Unit Test Framework) oder Debugging-Werkzeuge, zur Verfügung. In der Praxis sind oft die für den Code verantwortlichen Entwickler an den Tests beteiligt. Dabei werden die gefundenen Fehler häufig sofort korrigiert und gar nicht erst formell erfasst.

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 Testfallentwicklung, der Entwicklung und Integration von kleinen Code-Stücken und der Ausführung von Komponententests bis diese bestanden sind.

2.2.2 Integrationstest (K2)

Der Integrationstest prüft die Schnittstellen zwischen Komponenten, die Interaktionen zwischen verschiedenen Teilen eines Systems, wie zum Beispiel zum Betriebssystem, Dateisystem, zur Hardware oder den Schnittstellen zwischen Systemen.

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

- Ein Komponentenintegrationstest prüft das Zusammenspiel der Softwarekomponenten und wird nach dem Komponententest durchgeführt.
- Ein Systemintegrationstest prüft das Zusammenspiel verschiedener Systemen 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, so dass Änderungen destabilisierend sein können. Geschäftsprozesse, die als Workflows implementiert sind, können eine Reihe von Systemen nutzen. Inter-Plattform-Fragen können signifikant sein.

Je größer der Umfang einer Integration ist, desto schwieriger ist die Isolation von Fehlerwirkungen in einer spezifischen Komponente oder einem System, was zur Risikoerhöhung 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 das Risiko später Fehlererkennung zu reduzieren, sind inkrementelle Integrationsstrategien normalerweise der Big-Bang-Strategie vorzuziehen.

Das Testen spezieller nicht-funktionaler Eigenschaften (z.B. Leistungsfähigkeit) kann im Integrationstest enthalten sein.

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. 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 am effizientesten Reihenfolge, entwickelt werden.

2.2.3 Systemtest (K2)

Der Systemtest beschäftigt sich mit dem Verhalten eines Gesamtsystems/-produkts, wie es in einem Entwicklungsprojekt- oder –programm definiert ist.

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

Systemtests können Tests, basierend auf einer Risikobewertung und/oder auf Anforderungsspezifikationen, Geschäftsprozessen, Anwendungsfällen oder anderen abstrakten Beschreibungen des Systemverhaltens, der Interaktionen mit dem Betriebssystem und den Systemressourcen beinhalten.

Systemtests sollen sowohl funktionale als auch nicht-funktionale Anforderungen an das System untersuchen. Diese können als Text und/oder als Modelle vorliegen. Dabei müssen sich Tester auch oft mit unvollständigen oder undokumentierten Anforderungen befassen. Funktionale Anforderungen werden im Systemtest zunächst mit spezifikationsbasierten Testentwurfsverfahren (Black-Box-Testentwurfverfahren) getestet. Beispielsweise kann eine Entscheidungstabelle für die Kombination der Wirkungen in Geschäftsregeln erstellt werden. Strukturbasierte Techniken (White-Box-Testentwurfsverfahren) können eingesetzt werden, um die Überdeckung der Tests bzgl. eines strukturellen Elementes, wie die Menüstruktur oder eine Navigationsstruktur einer Web-Site, zu bewerten (siehe Kapitel 4).

Systemtests werden oftmals durch unabhängige Testteams durchgeführt.

2.2.4 Abnahmetest (K2)

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 Fehlern ist nicht das Hauptziel beim Abnahmetest. Abnahmetests können die Bereitschaft eines Systems für den Einsatz und die Nutzung bewerten, obwohl es nicht notwendigerweise die letzte Teststufe ist. So könnte beispielsweise ein umfangreicher Systemintegrationstest dem Abnahmetest eines der Systeme folgen.

Die Durchführung von Teil-Abnahmetests in niedrigeren Teststufen kann sinnvoll sein:

- 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.

Der Abnahmetest beinhaltet normalerweise folgende Aspekte:

Anwender-Abnahmetest

Prüft die Tauglichkeit eines Systems zum Gebrauch durch Anwender bzw. Kunden.

Betrieblicher Abnahmetest

Die Abnahme des Systems durch den Systemadministrator beinhaltet etwa:

- Test des Erstellens und Wiedereinspielens von Sicherungskopien (Backup/Restore)
- Wiederherstellbarkeit nach Ausfällen
- Benutzermanagement
- Wartungsaufgaben und
- Periodische Überprüfungen von Sicherheitslücken.

Regulatorischer und vertraglicher Abnahmetest

Beim vertraglichen Abnahmetest wird kundenindividuelle Software explizit gegen die vertraglichen Abnahmekriterien geprüft. Abnahmekriterien sollten definiert werden, wenn der Vertrag abgeschlossen wird.
Regulatorische Abnahmetests werden gegen alle Gesetze und Standards durchgeführt, denen das System entsprechen muss – wie z.B. staatliche, gesetzliche oder Sicherheitsbestimmungen.

Alpha- und Beta-Test (oder Feldtest)

Hersteller kommerzieller oder Standardsoftware wollen oftmals Feedback von potentiellen oder existierenden Kunden erhalten, bevor ein Produkt kommerziell zum Kauf angeboten wird. Der Alpha-Test wird am Herstellerstandort durchgeführt. Der Beta- (oder Feldtest) wird an den Kundenstandorten durchgeführt. Beide Tests werden durch potenzielle Kunden, und nicht durch die Entwickler des Produktes, durchgeführt.

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

2.3 Testarten (K2)	40 Minuten

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

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 entweder auf ein spezielles Testziel ausgerichtet, wie das Testen einer zu erfüllenden Funktion, einer nicht-funktionalen Anforderungen, wie Zuverlässigkeit oder Benutzbarkeit, der Struktur oder Architektur der Software beziehungsweise des Systems. Es kann sich auch auf Änderungen, wie das Prüfen der erfolgreichen Beseitigung eines Fehlers (Nachtest) oder das Prüfen auf unbeabsichtigte beziehungsweise ungewollte Änderungen bzw. Seiteneffekte (Regressionstest), beziehen.

Für die funktionalen wie auch für die strukturellen Tests kann ein Modell der zu testenden Software entwickelt oder eingesetzt werden, zum Beispiel ein Prozessablaufmodell, ein Zustandsübergangsmodell oder eine Klartextspezifikation für das funktionale Testen, ein Kontrollflussmodell oder ein Menüstrukturmodell für das strukturelle Testen.

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 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. Es kommt in allen Teststufen zur Anwendung (z.B. Komponententest basierend auf der Komponentenspezifikation).

Spezifikationsbasierte 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, welche 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, der die Fähigkeit des Softwareprodukts bewertet 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 beinhaltet 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 verwendet 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 – Software Product Quality’ (ISO 9126) definiert sind.

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

Strukturelles Testen (White-Box-Test) kann in allen Teststufen angewandt werden. Strukturelle Testentwurfsverfahren werden am besten nach den spezifikationsbasierten 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 abgedeckten 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, im speziellen aber im Komponenten- und Komponentenintegrationstest, eingesetzt werden. Strukturelles Testen kann auch auf der Systemarchitektur aufbauen, so zum Beispiel auf der Aufrufhierarchie.

Der Ansatz des strukturorientierten Testens kann sinngemäß ebenso in den Teststufen System-, Systemintegration- oder Abnahmetest eingesetzt werden (z.B. auf Geschäftsmodelle oder auf Menüstrukturen).

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

Nachdem ein Fehler entdeckt und korrigiert wurde, sollte die Software nachher erneut getestet werden, um zu bestätigen, dass der Fehler erfolgreich entfernt wurde. In diesem Fall sprechen wir von einem Fehlernachtest. Das Debugging (Lokalisieren, analysieren und entfernen der Fehlerursache) ist eine Entwicklungs- und keine Testaufgabe.

Unter Regressionstest verstehen wir das wiederholte Testen eines bereits getesteten Programms nach dessen Modifikation, mit dem Ziel nachzuweisen, dass durch die vorgenommenen Modifikationen keine Fehler eingebaut oder (bisher maskierte) aufgedeckt wurden. Diese Fehler 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 Fehlern 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 betreffen funktionale, nichtfunktionale wie auch strukturelle Tests. Regressionstests werden oft wiederholt und ändern sich im Normalfall eher selten was sie damit zum bevorzugten Kandidaten für eine Automatisierung macht.

2.4 Wartungstest (K2) — 15 Minuten

Begriffe
Analyse der Auswirkung, Wartungstest

Hintergrund

Einmal in Betrieb, bleibt ein Softwaresystem oft über Jahre oder Jahrzehnte im Einsatz. Während dieser Zeit werden das System und seine Umgebung mehrmals korrigiert, gewechselt oder erweitert. 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 beinhalten geplante Erweiterungen (z.B. basierend auf dem geplanten Release), Korrekturen und Notkorrekturen, Umgebungsänderungen wie geplante Aktualisierung des Betriebs- oder Datenbanksystems, oder Patches zu erst kürzlich entdeckten Schwachstellen des Betriebssystems.

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 beinhalten.

Wartungstest beim Einzug eines Systems kann das Testen der Datenmigration oder der Archivierung (falls eine lange Aufbewahrungszeit notwendig ist) beinhalten.

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

Die Ermittlung, inwiefern ein bestehendes System durch Modifikationen beeinflusst wird, nennt man Analyse der Auswirkungen. Diese wird verwendet, um zu entscheiden, wie viele Regressionstests durchzuführen sind.

Bei veralteten oder gar fehlenden Spezifikationen kann sich der Wartungstest sehr schwierig gestalten.

Referenzen
2 Linz, 2005
2.1.3 CMMI, Craig, 2002, Hetzel, 1988, IEEE 12207
2.2 Hetzel, 1988
2.2.4 Copeland, 2004, Myers 1982
2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004
2.3.2 Black, 2001, ISO 9126
2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1988
2.3.4 Hetzel, 1988, IEEE 829
2.4 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE 829

3 Statischer Test (K2)	60 Minuten

Lernziele für den Abschnitt statischer Test

Die Ziele legen fest, was Sie nach Beendigung 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 den Nutzen statischer Methoden für die Bewertung von Arbeitsergebnissen der Softwareentwicklung beschreiben. (K2)

LO-3.1.3 Die Unterschiede zwischen statischen und dynamischen Techniken erklären. (K2)

LO-3.1.4 Das Ziel der Statischen Analyse und Reviews beschreiben und diese mit dem dynamischen Testen vergleichen. (K2)

3.2 Reviewprozess (K2)

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

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

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

3.3 Werkzeuggestützte statische Analyse (K2)

LO-3.3.1 Typische Fehlerzustände und Fehler wiedergeben, 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 auflisten. (K1)

LO-3.3.3 Typische Fehler 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, statische Prüftechnik, Statischer Test,

Hintergrund

Entgegen dem Dynamischen Testen, welches die Ausführung der Software verlangt, beruhen statische Prüftechniken auf der „manuellen“ Überprüfung (Reviews) oder automatisierten Analysen (statische Analyse) des Codes oder anderer Projektdokumentation.

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

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.

Nutzen von Reviews umfassen frühe Fehleraufdeckung und Korrektur, Verbesserung der Softwareentwicklungsproduktivität, reduzierte Entwicklungsdauer, reduzierte Testkosten und -dauer, Reduzierung der Kosten während der Lebensdauer, weniger Fehler und verbesserte Kommunikation. Reviews können Auslassungen zum Beispiel fehlende 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 Fehler 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 Fehlerzustände eher als Fehlerwirkungen.

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

3.2 Reviewprozess (K2) — 25 Minuten

Begriffe
Eingangsbedingung, Formales Review, Gutachter, Informelles Review, Inspektion, Metrik, Moderator/ Leiter der Inspektion, Peer Review, Protokollant, Technisches Review, Walkthrough

Hintergrund

Die verschiedenen Typen von Reviews variieren zwischen sehr informell (d.h. keine schriftlichen Vorgaben für die Gutachter) und sehr formal (d.h. gut strukturiert und geregelt). Der Formalismus eines Reviewprozesses ist abhängig von Faktoren wie Reife des Entwicklungsprozesses, gesetzlichen oder regulatorischen Anforderungen oder dem Bedarf an einem Prüfnachweis.

Die Art und Weise, wie ein Review durchgeführt wird, ist abhängig von den festgelegten Zielen des Reviews (z.B. das Finden von Fehlern, dem Erwerben von Verständnis oder einer Diskussion mit Entscheidung durch Konsens).

3.2.1 Phasen eines formalen Review (K1)

Ein typisches formales Review besteht aus folgenden Hauptphasen:

1. **Planung**: Auswahl der beteiligten Personen, Besetzung der Rollen; Festlegung der Eingangsbedingung und Endebedingung bei mehr formalen Reviewarten (z.B. Inspektion) und Festlegung der zu betrachtenden Dokumententeile
2. **Kick-off**: Verteilung der Dokumente; Erläuterung der Ziele, des Prozesses und der Dokumente den Teilnehmern gegenüber; und Prüfung der Eingangsbedingungen (bei formaleren Reviewarten)
3. **Individuelle Vorbereitung**: Tätigkeit, die von jedem Teilnehmer für sich allein vor der Reviewsitzung durchgeführt wird, Notierung von potentiellen Fehlern, Fragen und Kommentaren
4. **Reviewsitzung**: Diskussion oder Protokollierung, mit dokumentierten Ergebnissen oder Protokollen (bei formaleren Reviewarten). Die Teilnehmer der Sitzung können die Fehler notieren, Empfehlungen zum Umgang mit ihnen machen oder Entscheidungen über die Fehler treffen
5. **Überarbeitung**: Beheben der gefundenen Fehlerzustände typischerweise durch den Autor
6. **Nachbereitung**: Prüfung, dass Fehlerzustände behoben wurden, Sammeln von Metriken und Prüfung von Testendekriterien (bei formaleren Reviewarten)

3.2.2 Rollen und Verantwortlichkeiten (K1)

Bei einem typischen formalen Review findet man folgende Rollen:

- **Manager**: entscheidet über Durchführung von Reviews, stellt Zeit im Projektplan zur Verfügung 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. Fehler). Gutachter sollten so gewählt werden, dass verschiedene Sichten und Rollen im Reviewprozess vertreten sind und sie sollten an allen Reviewsitzungen teilnehmen können.

- **Protokollant**: dokumentiert alle Ergebnisse, Probleme und offenen Punkte, die im Verlauf der Sitzung identifiziert werden.

Dokumente aus verschiedenen Perspektiven zu betrachten und Checklisten zu nutzen, kann Reviews wirksamer und effizienter machen, z.B. eine Checkliste basierend auf dem Blickwinkel von Anwendern, denjenigen, die die Wartung durchführen, Testern oder der Betriebsführung, oder eine Checkliste mit typischen Anforderungsproblemen.

3.2.3 Reviewarten(K2)

Ein einzelnes Dokument 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 über eine Anforderungsspezifikation kann vor einem Walkthrough mit Kunden durchgeführt werden. Die Hauptcharakteristika, optionalen Bestandteile und Zwecke allgemeiner Reviewarten sind:

Informelles Review
Hauptmerkmale:
- kein formaler Prozess
- kann integriert im Paarweisen Programmieren (Pair Programming) sein oder ein technischer Leiter unterzieht Entwurf und Quellcode einem Review
- kann wahlweise dokumentiert werden
- Nutzen kann abhängig von Gutachtern variieren
- Hauptzweck: nützliche, kostengünstige Reviewart

Walkthrough
Hauptmerkmale:
- Sitzung geleitet durch den Autor
- Szenarien und Probeläufe, im Kreis gleichgestellter Mitarbeiter
- Open-End Sitzungen
- wahlweise der Sitzung vorausgehende Vorbereitung der Gutachter, Reviewbericht, Liste der Befunde und Protokollant (der aber nicht der Autor ist)
- kann in der Praxis von informell bis sehr formal variieren
- Hauptzweck: Lernen, Verständnis erzielen, Fehler finden

Technisches Review
Hauptmerkmale:
- Dokumentierter und definierter Fehlerfindungsprozess, der gleichgestellte Mitarbeiter und techn. Experten einschließt
- Kann als Peer Review ohne Teilnahme des Managements ausgeführt werden
- Idealerweise durch einen geschulten Moderator geleitet (nicht der Autor)
- Vorbereitung vor der Sitzung
- Wahlweise: Nutzung von Checklisten, Reviewberichte, Liste der Befunde und Managementteilnahme
- kann in der Praxis von informell bis sehr formal variieren
- Hauptzweck: Diskussion, Entscheidungen treffen, Alternativen bewerten, Fehler finden, technische Probleme lösen und prüfen, ob Übereinstimmung mit Spezifikationen und Standards existiert

Inspektion
Hauptmerkmale:
- Geleitet durch einen geschulten Moderator (nicht der Autor)
- Gewöhnlich Prüfung durch gleichgestellte Mitarbeiter
- definierte Rollen
- beinhaltet Metriken
- formaler Prozess basierend auf Regeln, Checklisten, Eingangsbedingungen und Endebedingung
- Vorbereitung vor der Sitzung

- Inspektionsbericht, Liste der Befunde
- Formaler Prozess für Folgeaktivitäten
- Wahlweise: Prozessverbesserung und Vorleser
- Hauptzweck: Fehler finden

Walkthroughs, Technische Reviews und Inspektionen können in einer Gruppe von gleichgestellten Kollegen (Peer) – Kollegen aus der gleichen organisatorischen Ebene durchgeführt werden. Dieser Typ Review wird auch „Peer Review“ genannt.

3.2.4 Erfolgsfaktoren für Reviews (K2)

Erfolgsfaktoren für Reviews beinhalten:

- Jedes Review hat klar vordefinierte Ziele.
- Auswahl geeigneter Personen, entsprechend den Reviewzielen
- Gefundene Fehler werden positiv aufgenommen und werden objektiv zur Sprache gebracht.
- Menschliche und psychologische Aspekte (z.B. es für den Autor zu einer positiven Erfahrung gestalten).
- Reviewtechniken werden angewendet, die für Typen und Stufe von Arbeitsergebnissen der Softwareentwicklung und 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.
- Management unterstützt einen guten Reviewprozess (z.B. durch Berücksichtigung angemessener Zeit für Reviewaktivitäten in Projektplänen).
- 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, Fehler 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 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.

Der Nutzen der statischen Analyse ist:

- Frühe Erkennung von Fehlern vor der Testdurchführung
- Frühe Warnung vor verdächtigen Aspekten in Code oder Design, durch Berechnung von Metriken wie z.B. ein hohes Komplexitätsmaß
- Identifizierung von Fehlern, die durch dynamischen Test nicht effektiv und effizient aufzudecken sind
- Aufdecken von Abhängigkeiten und Inkonsistenzen in Softwaremodellen, wie z.B. tote Links
- Verbesserte Wartbarkeit von Code und Design
- Vorbeugung von Fehlerzuständen, falls man aus Erfahrung gelernt hat und sich dies in der Entwicklung niederschlägt

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

- Referenzierung einer Variablen mit nicht definiertem Wert
- Inkonsistente Schnittstellen zwischen Modulen und Komponenten
- Variablen, die nie verwendet werden
- unerreichbarer (toter) Code
- Verletzung von Programmierkonventionen
- Sicherheitsschwachstellen
- Syntax-Verletzungen von Code und Softwaremodellen

Werkzeuge für statische Analysen werden typischerweise von Entwicklern vor oder während Komponenten- und Integrationstests genutzt (prüfen gegen vordefinierte Regeln oder Programmierstandards), und durch Designer während der Softwaremodellierung. Werkzeuge für statische Analyse 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 Berechnung von Metriken.

Referenzen
3 Linz, 2005
3.2 IEEE 1028
3.2.2 Gilb, 1993, van Veenendaal, 2004
3.2.4 Gilb, 1993, IEEE 1028
3.3 Van Veenendaal, 2004

4 Testfallentwurfsverfahren (K3)	285 Minuten

Lernziele für den Abschnitt Testfallentwurfsverfahren

Die Ziele legen fest, was Sie nach Beendigung des jeweiligen Moduls gelernt haben sollten.

4.1 Der Testentwicklungsprozess (K2)

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

LO-4.1.2 Gegenüberstellen der Begriffe Testbedingung, Testfall und Testszenario. (K2)

LO-4.1.3 Bewerten der Qualität von Testfällen, ob:
- eine klare Rückverfolgbarkeit zu den Anforderungen sichtbar wird
- das Sollverhalten beschrieben ist (K2)

LO-4.1.4 Übersetzen der Testfälle in eine wohlstrukturierte Testvorgehensspezifikation – bei einem Detaillierungsgrad, der den Vorkenntnissen der Tester angepasst ist (K3)

4.2 Kategorien von Testfallentwurfsverfahren (K2)

LO-4.2.1 Wiedergeben von Gründen, dass sowohl spezifikationsorientierte (Black-Box) als auch strukturorientierte (White-Box) Ansätze zum Testfallentwurf von Nutzen sind, und für beides gängige Verfahren aufzählen. (K1)

LO-4.2.2 Erklären der Eigenschaften von und Unterschiede zwischen spezifikationsorientiertem Testen, strukturorientiertem Testen und erfahrungsbasiertem Testen. (K2)

4.3 Spezifikationsorientierte oder Black-Box-Verfahren (K3)

LO-4.3.1 Schreiben von Testfällen anhand unterschiedlicher Softwaremodelle unter Verwendung der folgenden Testfallentwurfsverfahren: (K3)
- Äquivalenzklassenbildung
- Grenzwertanalyse
- Entscheidungstabellentest
- Zustandsbasierter Test

LO-4.3.2 Verstehen der Hauptziele der vier Verfahren, auf welchen Ebenen und in welchem Testumfeld das Verfahren eingesetzt und wie jeweils Testabdeckung gemessen werden kann. (K2)

LO-4.3.3 Verstehen des Konzepts der anwendungsfallbezogenen (Use Case) Tests und der damit verbundenen Vorteile. (K2)

4.4 Strukturorientierte oder White-Box-Verfahren (K3)

LO-4.4.1 Beschreiben des Konzepts und der Wichtigkeit von Codeüberdeckung. (K2)

LO-4.4.2 Erklären der Konzepte der Anweisungs- und Entscheidungsüberdeckung, verstehen, dass diese Abdeckungsmaße auch auf anderen Teststufen als im Komponententest eingesetzt werden können (z.B. bei Geschäftsprozessen auf Systemebene). (K2)

LO-4.4.3 Schreiben von Testfällen zu vorgegebenen Kontrollflüssen unter Verwendung der folgenden Testfallentwurfsverfahren: (K3)
- Anweisungsüberdeckungstest
- Entscheidungsüberdeckungstest.

LO-4.4.4 Überprüfen der Vollständigkeit von Anweisungs- und Entscheidungsüberdeckung. (K3)

4.5 Erfahrungsbasierte Verfahren (K2)

LO-4.5.1 Wiedergeben von Gründen, warum Testfälle auf der Grundlage von Intuition, Erfahrung und Wissen über typische Fehler geschrieben werden sollten. (K1)

LO-4.5.2 Vergleichen erfahrungsbasierter Verfahren mit spezifikationsorientierten Testverfahren. (K2)

4.6 Auswahl von Testverfahren (K2)

LO-4.6.1 Identifizieren von Faktoren, die die Auswahl von angemessenen Testfallentwurfsverfahren für eine bestimmte Problemstellung beeinflussen, wie der Typ des Systems, das Risiko, die Kundenanforderungen, die Modelle für Use-Case Modellierung, Anforderungsmodelle oder das Hintergrundwissen des Testers. (K2)

4.1 Der Testentwicklungsprozess (K2) — 15 Minuten

Begriffe
Rückverfolgbarkeit, Testausführungsplan, Testentwurf, Testfallspezifikation, Testvorgehensspezifikation, Testskript

Hintergrund
Der Prozess, 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 Organisation, der Reife des Testprozesses und des Entwicklungsprozesses, zeitlicher Beschränkungen und der einbezogenen Personen.

Während des Testentwurfs werden die dem Test zugrunde liegenden Dokumente analysiert, um festzustellen, was getestet werden muss, um also 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.

Die Herstellung einer Rückverfolgbarkeit von Testbedingungen auf Spezifikationen und Anforderungen erlaubt sowohl eine Analyse der Auswirkungen (Impact Analysis) aufgrund von 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 (vgl. 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 erwarteten Sollwerte und den erwarteten Nachbedingungen, entwickelt mit dem Ziel, ein oder mehrere Testbedingungen abzudecken. Der IEEE Standard 829 (‘Standard for Software Test Documentation’) beschreibt die Inhalte von Testvorgehensspezifikationen (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 beinhalten. 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 einem Drehbuch oder Testskript zusammengefasst. Das Testdrehbuch (oder das manuelle Testskript) 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, in der die verschiedenen Testszenarios (und gegebenenfalls die automatisierten Testskripte) ausgeführt werden, festlegt, und wann und von wem sie auszuführen sind. Der Testausführungsplan berücksichtigt dabei Faktoren wie Regressionstests, Priorisierung und logische Abhängigkeiten.

4.2 Kategorien von Testfallentwurfsverfahren (K2) — 15 Minuten

Begriffe
Black-Box-Testentwurfsverfahren, Erfahrungsbasiertes Testentwurfsverfahren, Spezifikationsorientiertes Testentwurfsverfahren, strukturorientiertes Testentwurfsverfahren, White-Box-Testentwurfsverfahren

Hintergrund

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

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

Black-Box-Verfahren (die auch spezifikationsorientierte und erfahrungsbasierte Verfahren umfassen) stellen einen Weg dar, Testbedingungen oder Testfälle für eine Komponente oder ein System ohne Berücksichtigung der internen Struktur nur aufgrund der Analyse der zugrunde liegenden Testdokumentation und der Erfahrung der Entwickler, Tester und Anwender – funktional oder nicht-funktional – abzuleiten und auszuwählen. White-Box-Verfahren (auch strukturelle oder strukturorientierte Verfahren) stützen sich auf eine Analyse der Struktur einer Komponente oder eines Systems.

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

Dieser Lehrplan bezeichnet spezifikationsorientierte oder Erfahrungsbasierte Ansätze als Black-Box-Verfahren, und strukturorientierte Ansätze als White-Box-Verfahren.

Gemeinsame Merkmale der spezifikationsorientierten Verfahren:
- Modelle, ob formal oder nicht formal, werden zur Spezifikation des zu lösenden Problems, der Software oder ihrer Komponente herangezogen.
- Von diesen Modellen können systematisch Testfälle abgeleitet werden.

Gemeinsame Merkmale der strukturorientierten Verfahren:
- Informationen über den Aufbau der Software werden für die Ableitung von Testfällen verwendet, beispielsweise der Code und das Design.
- Der Überdeckungsgrad der Software kann für vorhandene Testfälle gemessen werden. Weitere Testfälle können zur Erhöhung des Überdeckungsgrades systematisch abgeleitet werden.

Gemeinsame Merkmale der erfahrungsbasierten Verfahren:
- 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
  - Das Wissen über wahrscheinliche Fehler und ihre Verteilung

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

Begriffe
Anwendungsfallbasierter Test, Äquivalenzklassenbildung, Entscheidungstabellentest, Grenzwertanalyse, zustandsbasierter Test,

4.3.1 Äquivalenzklassenbildung (K3)

Eingabedaten für die Software oder das System werden 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 wie für ungültige Daten – also Werte, die zurückgewiesen werden sollten – gebildet werden. Die Klassen können außerdem für Ausgabedaten, interne Werte, zeitbezogene Werte (z.B. vor oder nach einem Ereignis) und für Schnittstellenparameter (z.B. beim Integrationstest) gebildet werden. Tests können so entworfen werden, dass die 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 als Verfahren eingesetzt werden, um Überdeckungsgrade im 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 bei Anwendung von Grenzwerten innerhalb der Äquivalenzklassen mit einer höheren Wahrscheinlichkeit fehlerhaft ist, sind solche Grenzen ein Bereich, in dem Testen möglicherweise Fehler aufdecken wird. Der größte und der kleinste Wert einer Klasse sind deren Grenzwerte. Für Tests nutzt man den exakten Grenzwert und die beiden benachbarten Werte. 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 Potential, Fehler aufzudecken, ist hoch; detaillierte Spezifikationen sind hilfreich.

Dieses Verfahren wird häufig als Erweiterung der Äquivalenzklassenbildung betrachtet. Es kann bei der Bildung von Äquivalenzklassen angewendet werden, gleichermaßen für Benutzereingaben am Bildschirm, wie beispielsweise bei Zeitspannen (z.B. Timeout, Transaktionsgeschwindigkeitsanforderungen) oder Grenzen von Tabellenbereichen (z.B. Tabellengröße 256*256). Grenzwerte können auch für die Auswahl von Testdaten verwendet werden.

4.3.3 Entscheidungstabellentest (K3)

Basierend auf der Ursache-Wirkungs-Graph-Analyse sind Entscheidungstabellen 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. Die Spezifikation wird untersucht, und die Bedingungen und Aktionen des Systems werden ermittelt.

Die Eingabebedingungen und Aktionen werden meist so festgesetzt, dass sie entweder „wahr“ oder „falsch“ sein können (Boolsche Werte). Die Entscheidungstabelle enthält die auslösenden Bedingungen, oft Kombinationen von „wahr“ und „falsch“ für alle Eingabebedingungen und die daraus resultierenden 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 benötigt wird, was in der Regel die Abdeckung aller Kombinationen der auslösenden Bedingungen beinhaltet.

Die Stärke des Entscheidungstabellentest 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 als Zustandsdiagramm dargestellt werden. Es ermöglicht dem Tester, die Software im Bezug auf ihre Zustände, die Übergänge zwischen den Zuständen, die Eingaben oder Ereignisse, die die Zustandsübergänge (Transitionen) auslösen und die Aktionen, die aus den Übergängen folgen können, darzustellen. 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 im Hinblick auf Abdeckung einer typischen Sequenz von Zuständen, jedes Zustands, auf Ausführung jeder, auf Ausführung bestimmter Sequenzen von Übergängen oder auch zum Test ungültiger Übergänge abgeleitet werden.

Zustandsbasierter Test wird häufig in Branchen der eingebetteten Software („Embedded Software“) und generell in der Automatisierungstechnik eingesetzt. Davon abgesehen ist dieses Verfahren genauso gut für die Modellierung von Geschäftsobjekten, die verschiedene Zustände besitzen, oder zum Test von dialogbasierten Abläufen (z.B. für Internet-Anwendungen oder Geschäftsszenarien) einsetzbar.

4.3.5 Anwendungsfallbasierter Test (K2)

Tests können auf Basis von Anwendungsfällen (Use Cases) oder Geschäftsszenarien spezifiziert werden. Ein Anwendungsfall beschreibt die Interaktionen zwischen den Aktoren, einschließlich Anwender und System, die ein aus Sicht des Anwenders gewünschtes und wahrnehmbares Ergebnis zur Folge haben.

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 manchmal mehrere alternative Zweige (Varianten).

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 Fehler in den Prozessabläufen aufzudecken. Anwendungsfälle, die oft auch als „Szenarios“ bezeichnet werden, 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 Fehler im Umfeld der Integration aufdecken, die durch den Test der einzelnen Komponenten nicht gefunden werden könnten.

4.4 Strukturorientierter Test oder White-Box-Verfahren (K3) — 60 Minuten

Begriffe
Anweisungsüberdeckung, Codeüberdeckung, Entscheidungsüberdeckung, strukturorientierter Test

Hintergrund

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

- Komponentenebene: Die Struktur ist die des Codes selbst, also Anweisungen, Entscheidungen oder Zweige.
- Integrationsebene: Die Struktur kann ein Aufruf-Baum 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 zwei codebezogene, strukturorientierte Verfahren für Codeüberdeckung, bezogen auf Anweisungen und Entscheidungen, vorgestellt. Für Entscheidungsüberdeckungstest können Kontrollflussgraphen zur Darstellung der Alternativen für jede Entscheidung herangezogen werden.

4.4.1 Anweisungstest und -überdeckung (K3)

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

4.4.2 Entscheidungsüberdeckungstest (K3)

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 Entscheidungsüberdeckungstest werden Testfälle so abgeleitet, dass bestimmte Entscheidungsergebnisse, mit dem Ziel die Entscheidungsüberdeckung zu erhöhen, ausgeführt werden.

Der Entscheidungsüberdeckungstest ist eine Form des kontrollflussbasierten Tests, da es einen speziellen Kontrollfluss durch die Entscheidungspunkte erzeugt. Entscheidungsüberdeckung ist stärker als Anweisungsüberdeckung: 100% Entscheidungsüberdeckung schließt 100% Anweisungsüberdeckung ein, aber nicht umgekehrt.

4.4.3 Andere strukturorientierte Verfahren (K1)

Es gibt stärkere strukturorientierte Überdeckungsgrade über Entscheidungsüberdeckung hinaus, beispielsweise Bedingungsüberdeckung und Mehrfachbedingungsüberdeckung.

Das Konzept der Überdeckungsgrade kann auch auf andere Teststufen übertragen werden (z.B. Integrationsebene), wobei der prozentuale Anteil von Modulen, Komponenten oder Klassen, die durch eine Testsuite ausgeführt wurden, als Modul-, Komponenten- oder Klassen-Überdeckung bezeichnet werden kann.

Werkzeugunterstützung ist beim strukturorientierten Test von Code sehr hilfreich.

4.5 Erfahrungsbasierte Verfahren (K2) — 30 Minuten

Begriffe
Exploratives Testen, Fault Attack

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 sie den formaleren Ansätzen nachgeschaltet eingesetzt wird. Abhängig von der Erfahrung des Testers kann die Wirksamkeit dieses Verfahren sehr stark variieren.

Ein weit verbreitetes erfahrungsbasiertes Testverfahren ist Error Guessing. Gewöhnlich sehen Tester Fehler auf Grund ihrer Erfahrung voraus. Eine strukturierte Herangehensweise an das Error Guessing Verfahren ist es, eine Liste möglicher Fehler zu erstellen, und dann Testfälle zu entwerfen, die auf diese Fehler abzielen. Dieser systematische Ansatz wird Fault Attack genannt. Diese Liste der Fehlerzustände und Fehlerwirkungen kann aufgrund von Erfahrung, verfügbaren Daten über Fehlerzustände und Fehlerwirkungen und von Allgemeinwissen darüber, warum Software sich falsch verhalten kann, erstellt werden.

Exploratives Testen ist gleichzeitiger Testfallentwurf, Testdurchführung, Testprotokollierung und Lernen, auf Grundlage einer Test-Charta, der die Testziele zu entnehmen sind, und 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, um zu helfen sicherzustellen, dass die schwerwiegendsten Fehler 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, behördliche Anforderungen, Kunden- oder Vertragsanforderungen, Risikograd, Art des Risikos, Testziel, verfügbarer Dokumentation, Wissen der Tester, Zeit und Geld, Softwareentwicklungsmodell, Anwendungsfallmodelle sowie frühere Erfahrungen mit gefundenen Fehlerarten.

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

Referenzen
4 Linz, 2005
4.1 Craig, 2002, Hetzel, 1988, IEEE 829
4.2 Beizer, 1990, Copeland, 2004
4.3.1 Copeland, 2004, Myers, 1982
4.3.2 Copeland, 2004, Myers, 1982, Linz, 2005
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 Ziele legen fest, was Sie nach Beendigung des jeweiligen Moduls gelernt haben sollten.

5.1 Testorganisation (K2)

LO-5.1.1 Die Wichtigkeit unabhängigen Testens erkennen. (K1)

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

LO-5.1.3 Erkennen der unterschiedlichen Teammitglieder, die für die Zusammenstellung eines Testteams berücksichtigt werden müssen. (K1)

LO-5.1.4 Aufgaben typischer Testleiter und Tester kennen. (K1)

5.2 Testplanung und -aufwandsschätzung (K2)

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

LO-5.2.2 Zusammenfassen des Ziels und des Inhalts des Testkonzepts, der Testspezifikation und der Testverfahrensdokumente auf der Basis des 'Standard for Software Test Documentation' (IEEE 829). (K2)

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

LO-5.2.4 Unterscheiden zwischen dem Gegenstand der Testplanung für ein System und die 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 (K3)

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

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

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

LO-5.2.9 Erkennen/begründen von angemessenen Testendekriterien 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, die für die Überwachung von Testvorbereitung und Testdurchführung angewendet werden, kennen. (K1)

LO-5.3.2 Testmetriken für Testberichte und Teststeuerung (z.B. aufgedeckte und behobene Fehler und bestandene und nicht bestandene Tests) verstehen und interpretieren. (K2)

LO-5.3.3 Zusammenfassen des Zwecks und Inhalts des Testberichts auf der Basis des 'Standard for Software Test Documentation' (IEEE 829). (K2)

5.4 Konfigurationsmanagment (K2)

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

5.5 Risiko und Testen (K2)

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

LO-5.5.2 Widergeben, dass Risiken durch die Wahrscheinlichkeit (des Eintritts) und die Auswirkung (Schaden im Eintrittsfall) bestimmt werden. (K1)

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

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

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

5.6 Abweichungsmanagement / Fehlermanagement (K3)

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

LO-5.6.2 Schreiben eines Fehlerberichts über die Beobachtung einer Fehlerwirkung während des Testens. (K3)

5.1 Testorganisation (K2)	**30 Minuten**

Begriffe
Tester, Testleiter, Testmanager

5.1.1 Testorganisation und Unabhängigkeit (K2)

Die Effektivität der Fehlerfindung durch Testen und Prüfungen kann durch den Einsatz unabhängiger Tester verbessert werden. Möglichkeiten für Unabhängigkeit sind:

- 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 Testziele, wie z.B. Tester für Benutzerfreundlichkeit, Sicherheits- oder Konformitätstester (die ein Softwareprodukt gegen Standards und gesetzliche Vorschriften prüfen)
- Unabhängige Tester aus Outsourcing oder aus externen Organisationen.

Für große, komplexe oder sicherheitskritische Projekte 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. Tester sollten diese prozessverwandten Rollen aber nur bei Vorliegen eines klaren Managementauftrags einnehmen.

Die Vorteile von Unabhängigkeit beinhalten:

- 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.

Die Nachteile von Unabhängigkeit beinhalten:

- Isolierung vom Entwicklungsteam (wenn als vollkommen unabhängig behandelt).
- Unabhängige Tester können als letzte Prüfinstanz einen Engpass darstellen.
- Die Entwickler können das Verantwortungsgefühl für Qualität verlieren.

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

5.1.2 Aufgaben von Testleiter und Tester (K1)

In diesem Lehrplan werden die zwei Rollen Testleiter 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 Testleiter als Testmanager 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: Testleiter und Testmanager. Typischerweise plant, überwacht und steuert der Testleiter die Testaktivitäten und die Aufgaben, wie in Kapitel 1.4 definiert.

Typische Aufgaben eines Mitarbeiters in der Rolle Testleiter können sein:

- Koordination der Teststrategie und die Planung mit Projektleitern und anderen Beteiligten.
- Erstellen oder prüfen der Teststrategie für das Projekt und einer Testgrundsatzrichtlinie für die Organisation.

- Beitragen der Testperspektive zu anderen Projektaktivitäten, beispielsweise der Integrationsplanung.
- Planung der Tests – unter Berücksichtigung des Kontexts und mit Verständnis der Testziele und Risiken – inklusive der Auswahl der Testvorgehensweise, Schätzung der Zeit, des Aufwands und der Kosten des Testens, Beschaffung der Ressourcen, Definition der Teststufen, Testdurchläufe und Planung des Abweichungsmanagements.
- Initiierung der Spezifikation, Vorbereitung, Implementierung und Durchführung von Tests, und Überwachung der Testergebnisse und Prüfung der Testendekriterien.
- Anpassung der Planung an Testergebnisse und den Testfortschritt (manchmal in den Statusberichten dokumentiert) und Einleitung aller erforderlichen Maßnahmen zum Ausgleich von Problemen.
- Aufbau eines angemessenen Konfigurationsmanagements der Testmittel zwecks Rückverfolgbarkeit.
- Einführung passender Metriken zur Messung 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.
- Entscheidung über die Implementierung der Testumgebung treffen.
- Das Schreiben von Testberichten auf der Grundlage der Informationen, die während des Testens gesammelt werden.

Typische Aufgaben eines Testers können sein:

- Mitarbeit und Prüfen von Testplänen.
- Analyse, Prüfung und Bewertung von Benutzeranforderungen, Spezifikationen und Modellen im Hinblick auf Testbarkeit.
- Erstellen von Testspezifikationen.
- Aufbau der Testumgebung (oftmals in Abstimmung mit System- und Netzwerkadministration).
- Vorbereitung oder Anforderung von Testdaten.
- Implementierung von Tests auf allen Stufen, Durchführung der Tests und ihre Protokollierung, Auswertung der Testergebnisse und Dokumentation der Abweichungen von erwarteten Ergebnissen.
- Das Einsetzen von Testadministrations- oder Testmanagement- und Testüberwachungswerkzeugen wie gefordert.
- Automatisierung von Tests (kann durch einen Entwickler oder Testautomatisierungsexperten unterstützt werden).
- Messung der Leistungsfähigkeit/Performanz von Komponenten und Systemen (wenn zutreffend).
- Prüfung der Tests, die von anderen entwickelt wurden.

Personen, die in der Testanalyse, Testspezifikation, 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 so einen gewissen Grad von Unabhängigkeit erreichen. Typische Tester auf der Komponenten- und Integrationsstufe wären Entwickler, auf der Abnahmeteststufe Fachexperten und Anwender, und für den Abnahmetest auf operativer Ebene der Betreiber.

5.2 Testplanung und -schätzung (K2)	**40 Minuten**

Begriffe
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 Projektplan oder Testkonzept und in separaten Testplänen für Teststufen, wie z.B. dem Systemtest und dem Abnahmetest, dokumentiert werden. Entwürfe für Testplanungsdokumente werden im 'Standard for Software Test Documentation' (IEEE 829) bereitgestellt.

Die Planung wird durch die Testgrundsatzrichtlinie der Organisation, den Testumfang, die Ziele, Risiken, Einschränkungen, Risikoklassen/ Kritikalität, Testbarkeit und Verfügbarkeit von Ressourcen beeinflusst. Je weiter sich die Projekt- und Testplanung entwickelt, desto mehr Information ist 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 können.

5.2.2 Testplanungsaktivitäten (K2)

Testplanungsaktivitäten können sein:

- Festlegen des Umfangs und der Risiken sowie Identifizierung der Testziele
- Definieren des allgemeinen Testansatzes (die Teststrategie), inklusive der Definition der Teststufen und der Eingangs- und Testendekriterien
- Koordinieren und integrieren der Testaktivitäten in die Aktivitäten des Softwarelebenszyklus: Beschaffung, Bereitstellung, Entwicklung, Betrieb und Wartung
- Entscheiden, was zu testen ist, welche Rollen die 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 Detaillierungsgrades, der Struktur und der Vorlagen für die Testdokumentation
- Selektieren der Metriken zur Überwachung und Steuerung der Testvorbereitung und -durchführung, Fehlerbehebung und Risikofaktoren
- Bestimmen des Detaillierungsgrades für Testszenarien, um genügend Informationen in Hinblick auf eine reproduzierbare Testvorbereitung und -durchführung zu liefern

5.2.3 Testendekriterien (K2)

Das Ziel von Testendekriterien ist, zu 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 Testendekriterien sind z.B.:

- Intensitätsmaße, wie z.B. Abdeckung von Code, Funktionalität oder Risiko
- Schätzungen über Fehlerdichte oder Zuverlässigkeitsmaße
- Kosten
- Verbleibende Risiken, wie z.B. nicht behobene Fehler oder fehlende Testüberdeckung in bestimmten Bereichen
- Zeitpläne z.B. basierend auf dem Termin der Markteinführung

5.2.4 Testaufwandsschätzung (K2)

Zwei Ansätze für die Schätzung des Testaufwands werden in diesem Lehrplan abgedeckt:

- 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 der 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), die Größe des Produkts, die Komplexität der Problembereiche, die Anforderungen an Zuverlässigkeit und Sicherheit und die Anforderungen an die Dokumentation
- Charakteristiken des Entwicklungsprozesses: die Stabilität der Organisation, benutzte Werkzeuge, Testprozess, Kenntnisse der involvierten Personen und Zeitdruck
- Den Testergebnissen: die Anzahl der Fehler und die Menge der erforderlichen Nacharbeiten

5.2.5 Testvorgehensweise (Teststrategien) (K2)

Eine Möglichkeit, Testvorgehensweise oder Teststrategien zu klassifizieren, basiert auf dem Zeitpunkt, zu dem der Großteil der Testentwurfsarbeiten begonnen wird:

- Präventive Vorgehensweise, in denen Tests so früh wie möglich entworfen werden
- Reaktive Vorgehensweise, in denen der Testentwurf stattfindet, nachdem die Software oder das System entwickelt wurden

Typische Vorgehensweisen oder Strategien enthalten:

- 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 (wie z.B. Zuverlässigkeitswachstumsmodelle) oder Systembenutzung (wie z.B. Benutzungsprofile) nutzt
- Methodische Vorgehensweisen, wie das ausfallbasierte (inklusive Error Guessing und Fault Attacks), erfahrungsbasiertes, checklistenbasiertes und qualitätsmerkmalbasiertes Testen
- Prozess- oder standardkonforme Vorgehensweisen, wie durch Industriestandards oder die verschiedenen agilen Methoden spezifiziert
- Dynamische und heuristische Vorgehensweisen, wie das explorative Testen, bei dem das Testen stärker auf Ereignisse reagiert, als vorgeplant und die Durchführung und Auswertung quasi gleichzeitige Aufgaben sind
- Beratende Vorgehensweisen, in denen die Testabdeckung primär durch Hinweise und Beratung von Technologie- und/oder Geschäftsbereichsexperten außerhalb des Testteams getrieben wird
- Beim wiederverwendungsorientierten Vorgehensweisen übernimmt man vorhandene Tests und Testumgebungen (aus früheren Projekten) als Ausgangsbasis. Ziel ist, die Tests schnell und pragmatisch aufzusetzen.

Unterschiedliche Ansätze können kombiniert werden, z.B. eine risikoorientierte dynamische Vorgehensweise.

Die Auswahl einer Testvorgehensweise sollte den Kontext berücksichtigen, inklusive:

- Risiko des Scheiterns des Projekts, Gefahren für das Produkt und Risiken von Produktausfällen für Personen, für die Umwelt und für das Unternehmen
- Qualifikation und Erfahrung der Personen in den vorgeschlagenen Werkzeugen und Methoden
- Das Ziel der Tests und den Auftrag des Testteams

- Aspekte von Regularien wie externe und interne Bestimmungen für den Entwicklungsprozess.
- Die Art des Produkts und des Geschäftsfelds

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

Begriffe
Ausfallrate, Fehlerdichte, Testbericht, Teststeuerung, Testüberwachung,

5.3.1 Testfortschrittsüberwachung (K1)

Das Ziel der Testfortschrittsüberwachung ist es, Feedback und Übersicht über Testaktivitäten zu erhalten. Zu überwachende Informationen können manuell oder automatisiert gesammelt werden. Sie können herangezogen werden, um Testendekriterien 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/fehlgeschlagenen Testfälle)
- Fehlerinformationen (z.B. Fehlerdichte, gefundene und behobene Fehler, Fehleraufdeckungsrate und Fehlernachtestergebnisse)
- Testabdeckung der Anforderungen, Risiken oder des Codes
- Subjektives Vertrauen der Tester in das Produkt
- Daten der Testmeilensteine
- Testkosten, inklusive der Kosten im Vergleich zum Nutzen durch das Auffinden des nächsten Fehlers oder für den nächsten Testdurchlauf

5.3.2 Testberichterstattung (K2)

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

- Was passierte während des Testzeitraums, z.B. an Kalenderdaten, an denen Testendekriterien erfüllt wurden
- Analysierte Informationen und Metriken zur Unterstützung von Empfehlungen und Entscheidungen über zukünftige Aktivitäten, wie eine Beurteilung der verbleibenden Fehler, die ökonomischen Vorteile fortgesetzten Testens, offen stehende Risiken und der Grad des Vertrauens in die getestete Software

Die Gliederung eines Testberichts ist im ‘Standard for Software Test Documentation’ (IEEE 829) enthalten.

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

- Die Angemessenheit der Testziele für die Teststufe
- Die Angemessenheit des gewählten Testvorgehens
- Die Effektivität des Testens in Relation zu seinen 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 sind:

- 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 der Testumgebung
- Setzen einer Eingangsbedingung mit der Maßgabe, dass Fehlerbehebungen durch einen Entwickler nachzutesten sind, bevor die Software an die nächste Teststufe übergeben wird

5.4 Konfigurationsmanagement (K2)	10 Minuten

Begriffe
Konfigurationsmanagement, Versionskontrolle

Hintergrund

Ziel des Konfigurationsmanagements ist die Etablierung und der Erhalt der Integrität der Produkte (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, einer Versionskontrolle unterworfen, Änderungen verfolgt und zueinander und zu den Entwicklungseinheiten (Testobjekten) in Beziehung gesetzt sind, 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 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, risikoorientiertes Testen

Hintergrund

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

5.5.1 Projektrisiken (K2)

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

- Organisatorische Faktoren:
  - Qualifikation und Mitarbeiterengpässe
  - Personal- und Schulungsaspekte
  - Politische Aspekte, wie
    - Probleme mit Testern, die ihre Anforderungen und Ergebnisse kommunizieren
    - Versagen bei der Verfolgung von Informationen, die im Testen und in Reviews gefunden werden (z.B. fehlende Verbesserung der Entwicklungs- und Testpraktiken)
  - Ungeeignete Einstellung gegenüber dem Testen oder ungeeignete Erwartungen an das Testen (z.B. Nichtanerkennung des Wertes von Fehlerfindung während des Testens)
- Technische Aspekte:
  - Probleme bei der Definition der richtigen Anforderungen
  - Der Umfang, in dem Anforderungen unter den gegebenen Randbedingungen erfüllt werden können
  - Die Qualität des Designs, des Codes und der Tests
- Lieferantenaspekte:
  - Versagen einer dritten Partei
  - Vertragsaspekte

Bei Analyse, Management und Abschwächung dieser Risiken folgt der Testmanager etablierten Projektmanagementprinzipien. Der Entwurf für Testkonzepte (test plan) im ‘Standard for Software Test Documentation’ (IEEE 829) fordert die Benennung von Risiken und 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, wie z.B.:

- Gelieferte fehleranfällige Software
- Das Potential, das die Software/Hardware einem Individuum oder einer Firma Schaden zufügen könnte
- Schlechte Softwareeigenschaften (z.B. fehlende/ mangelhafte Funktionalität, Zuverlässigkeit, Benutzbarkeit und Leistungsfähigkeit/Performanz).
- 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 des Risikomanagements liefert das Testen Rückmeldungen über das verbleibende Risiko, indem es die Effektivität der Behebung kritischer Fehler und von Notfallplänen misst.

Ein risikoorientiertes Testvorgehen erlaubt pro-aktive Möglichkeiten zur Reduzierung des Produktrisikos, beginnend mit den ersten Projektphasen. Es beinhaltet 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:

- Die einzusetzenden Testtechniken zu bestimmen
- Den auszuführenden Testumfang zu bestimmen
- Das Testen mit dem Ziel, die kritischen Fehler so früh wie möglich zu finden, zu priorisieren
- Um 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:

- die Bewertung (und regelmäßiger Wiederbewertung) dessen, was an Fehlern auftreten kann (Risiken)
- die Bestimmung, welche Risiken reduziert werden müssen
- die 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 Abweichungsmanagement / Fehlermanagement(K3)	**40 Minuten**

Begriffe
Abweichungsmanagement / Fehlermanagement, Abweichungsprotokollierung /Fehlerprotokollierung

Hintergrund

Da eines der Ziele des Testens das Finden von Fehlern ist, müssen Unterschiede zwischen aktuellen und erwarteten Ergebnissen als Abweichung aufgezeichnet werden. Abweichungen 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 Prozess, sowie Regeln für die Klassifizierung, etablieren.

Abweichungen können während Entwicklung, in Reviews und Tests, sowie im Einsatz von Software festgestellt werden. Sie können im Code oder dem betriebenen System oder in jeder Art von Dokumentation festgestellt werden (inklusive Anforderungen, Entwicklungsdokumenten, Testdokumenten und Benutzerinformation wie “Hilfe” oder Installationshandbüchern).

Abweichungsberichte haben folgende Ziele:

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

Ein Fehlerbericht kann die folgenden Informationen enthalten:

- Meldungsdatum, meldende Organisation und Autor
- IST- und SOLL-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, wie z.B. 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 829) behandelt.

Referenzen
5 Linz, 2005
5.1.1 Black, 2001, Hetzel, 1988
5.1.2 Black, 2001, Hetzel, 1988
5.2.5 Black, 2001, Craig, 2002, IEEE 829, Kaner 2002
5.3.3 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE 829
5.4 Craig, 2002
5.5.2 Black, 2001, IEEE 829
5.6 Black, 2001, IEEE 829

6 Testwerkzeuge (K2) — 80 Minuten

Lernziele für den Abschnitt Testwerkzeuge

Die Ziele legen fest, was Sie nach Beendigung des jeweiligen Moduls gelernt haben sollten.

6.1 Typen von Testwerkzeugen (K2)

LO-6.1.1 Verschiedene Testwerkzeuge klassifizieren/ differenzieren und den Aktivitäten des Testprozesses zuordnen. (K2)

LO-6.1.2 Werkzeuge für die Unterstützung der Entwickler beim Testen erkennen. (K1)

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

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

LO-6.2.2 Die in Testausführungswerkzeugen angewandten skriptbasierten Techniken (data driven und keyword driven) wiedergeben. (K1)

6.3 Einführung von Testwerkzeugen in eine Organisation (K1)

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

LO-6.3.2 Die Ziele einer Vorstudien-/ Pilotphase im Rahmen einer Werkzeugevaluierung benennen. (K1)

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

6.1 Typen von Testwerkzeugen (K2) — 45 Minuten

Begriffe
Analysewerkzeug, Anforderungsmanagementwerkzeug, Debugger, dynamischer Analysator, Fehler-/Abweichungsmanagementwerkzeug, Komponententestrahmenwerkzeug, Konfigurationsmanagementwerkzeug, Lasttestwerkzeug, Modellierungswerkzeug, Monitoringwerkzeug, Performanztestwerkzeug, Sicherheitsprüfwerkzeug, Stresstestwerkzeug, Testausführungswerkzeug, Testdatengenerator, Testentwurfswerkzeug, Testmanagementwerkzeug, Testrahmen, Überdeckungsanalysator, Vergleichswerkzeug, Werkzeugunterstützung für den Reviewprozess

6.1.1 Klassifizierung von Testwerkzeugen (K2)

Es existieren Testwerkzeuge, die verschiedene Aspekte des Testens unterstützen. 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. Es existieren kommerzielle Werkzeuge, die lediglich eine Testaktivität unterstützen, weiterhin werden auch so genannte Suiten oder Familien von Werkzeugen angeboten, die viele oder sämtliche Testaktivitäten unterstützen.

Durch Automatisierung sich wiederholender Testaufgaben kann die Effizienz der Testaktivitäten durch Werkzeuge verbessert werden. Des Weiteren kann durch eine Automatisierung (z.B. automatisierter Vergleich größerer Datenmengen oder simuliertes Verhalten) die Zuverlässigkeit der Testdurchführung verbessert werden.

Einige Testwerkzeuge werden als intrusiv bezeichnet; d.h. das Werkzeug selbst beeinflusst das Verhalten des Testobjekts. Zum Beispiel kann das tatsächliche Zeitverhalten, das mit verschiedenen Performanztestwerkzeugen gemessen wird, unterschiedlich sein oder es können verschiedene Überdeckungsgrade in Abhängigkeit vom verwendeten Überdeckungsanalysator gemessen werden. Ein solches intrusives Verhalten eines Testwerkzeugs wird auch als „Einflussnahme auf das System“ bezeichnet.

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

6.1.2 Werkzeugunterstützung für das Management des Testens (K1)

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

Testmanagementwerkzeuge

Testmanagementwerkzeuge können wie folgt charakterisiert werden:

- Unterstützung des Testmanagements und der durchzuführenden Testaktivitäten
- Schnittstellen zu Testausführungswerkzeugen, Abweichungsverfolgungs- und Anforderungsmanagementwerkzeugen
- Unabhängige Versionskontrolle oder Schnittstelle zu einem externen Konfigurationsmanagementwerkzeug
- Unterstützung der Rückverfolgbarkeit von Tests, Testergebnissen und Vorfällen zu den ursprünglichen Dokumenten (Basisdokumentation), wie den Anforderungsspezifikationen
- Aufzeichnung von Testergebnissen und Generierung von Fortschrittsberichten
- Quantitative Analyse (Metriken) bezogen auf den Test (z.B. Testläufe und bestandene Tests), sowie die Testobjekte (z.B. steigende Anzahl von Abweichungen), um Informationen über die Testobjekte zu liefern, und den Testprozess zu steuern und zu verbessern

Anforderungsmanagementwerkzeuge

Anforderungsmanagementwerkzeuge dienen der strukturierten Ablage von Anforderungen, prüfen auf Konsistenz sowie fehlende/undefinierte Anforderungen, ermöglichen die Priorisierung von Anforderungen und die Rückverfolgbarkeit von einzelnen Tests zu Anforderungen, Funktionen und/oder Features. Die Rückverfolgbarkeit kann im Testfortschrittsbericht dokumentiert werden. Des Weiteren kann der Grad der Überdeckung von Anforderungen, Funktionen und/oder Features durch eine Menge von Tests im Fortschrittsbericht festgehalten werden.

Abweichungsmanagementwerkzeuge

Abweichungsmanagementwerkzeuge ermöglichen die Ablage und Verfolgung von Abweichungsmeldungen, zum Beispiel Defekte, Fehlerwirkungen oder Anomalien. Sie unterstützen die Verwaltung dieser Abweichungsmeldungen, wobei folgende Möglichkeiten zur Verfügung stehen:

- Erleichterung der Priorisierung von Abweichungsmeldungen
- Zuordnung von Aufgaben zu bestimmten Personen und/oder Personengruppen (z.B. Fehlerbehebung oder Fehlernachtest)
- Zuordnung eines Status (z.B. zurückgewiesen, bereit zum Test (nach Fehlerkorrektur) oder zurückgestellt für das nächste Release)

Diese Werkzeuge ermöglichen die Verfolgung der Abweichungen über die Zeit, unterstützen statistische Analysen und liefern Berichte über Abweichungen. Diese Art von Werkzeugen wird oft auch als Fehlerverfolgungswerkzeug bezeichnet.

Konfigurationsmanagementwerkzeuge

Konfigurationsmanagementwerkzeuge sind im engeren Sinne keine Testwerkzeuge, werden aber typischerweise eingesetzt, um verschiedene Versionen und Builds, sowie Tests zu verfolgen.

Konfigurationsmanagementwerkzeuge:

- Speichern die Informationen über Versionen und Konfigurationen der Software und der benötigten Testmittel
- Erlauben die Rückverfolgbarkeit zwischen den Software-Produktkomponenten, den Varianten und den Testmitteln
- Sind insbesondere für die Verwaltung von mehreren Konfigurationen von Hardware- und Softwareumgebungen geeignet (z.B. für verschiedene Betriebssystemversionen, Bibliotheken und Compiler, Browser und Rechner)

6.1.3 Werkzeugunterstützung für den statischen Test (K1)

Unterstützende Werkzeuge für den Reviewprozess

Unterstützende Werkzeuge für den Reviewprozess dienen der Ablage von Informationen über den Reviewprozess, speichern und verteilen von Reviewanmerkungen, berichten über gefundene Abweichungen sowie den geleisteten Aufwand, stellen Referenzen zu Reviewregeln und/oder Checklisten bereit und ermöglichen die Rückverfolgbarkeit zwischen Dokumenten und Quellcode. Darüber hinaus können diese Werkzeuge eine Hilfe bei verteilten Reviews (online reviews) sein, d.h. für den Fall, dass die Mitglieder des Reviewteams an geographisch verschiedenen Orten angesiedelt sind.

Statische Analysewerkzeuge (E)

Statische Analysewerkzeuge unterstützen Entwickler, Tester und das Qualitätssicherungspersonal in der Aufdeckung von Fehlern und potentiellen Fehlern (Anomalien) vor dem dynamischen Testen. Sie bieten Unterstützung für folgende Zwecke:

- Sicherstellung der Einhaltung von Programmierkonventionen
- Analyse von Strukturen und Abhängigkeiten (z.B. verlinkte Webseiten)
- Unterstützung in der Analyse des Codes (z.B. im Rahmen der Wartung und Pflege)

Statische Analysewerkzeuge können Metriken (z.B. zyklomatische Komplexität) aus dem Code ermitteln und damit zusätzliche Informationen z.B. für die Planung und Risikoanalyse bereitstellen.

Modellierungswerkzeuge (E)

Modellierungswerkzeuge sind in der Lage aus Modellen eine Software zu validieren. Zum Beispiel kann eine Prüffunktion für Datenmodelle Fehlerzustände und Inkonsistenzen in einem Modell aufdecken; andere Modellierungswerkzeuge können Fehlerzustände in einem Zustands- oder Objektmodell aufdecken. Diese Werkzeuge werden oft zur Unterstützung der Generierung von Testfällen, basierend auf dem Modell verwendet (siehe Testentwurfswerkzeuge weiter unten).

Der Hauptnutzen von statischen Analyse- und Modellierungswerkzeugen besteht in der frühzeitigen und Kosten sparenden Aufdeckung von Fehlerzuständen im Entwicklungsprozess. Das führt in der Regel dazu, dass im Entwicklungsprozess weniger Ressourcen für die Überarbeitung bzw. Nacharbeit benötigt werden.

6.1.4 Werkzeugunterstützung für die Testspezifikation (K1)

Testentwurfswerkzeuge

Testentwurfswerkzeuge generieren Testeingaben oder ausführbare Tests aus den Anforderungen, der graphischen Benutzerschnittstelle, dem Entwurfsmodell (Zustands-, Daten- oder Objektmodell) oder aus dem Code ab. Diese Art von Werkzeugen kann auch das erwartete Verhalten (Sollwerte/Sollreaktionen) werkzeugunterstützt erzeugen (d.h. somit als Testorakel verwendet werden). Die aus dem Zustands- oder Objektmodell generierten Tests sind nützlich für die Verifizierung der Implementierung des Modells. In den wenigsten Fällen sind diese Tests aber geeignet, um alle Aspekte der Software oder des Systems zu verifizieren. Durch diese Vorgehensweise kann aber wertvolle Zeit gespart werden und eine Vollständigkeit durch die mittels des Werkzeugs generierten Tests erzielt werden.

Andere Werkzeuge dieser Kategorie bieten eine Hilfestellung durch die Generierung von strukturierten Vorlagen (“Templates”) an (auch Testrahmen genannt), welche Tests oder Platzhalter generieren und somit den Testentwurfsprozess beschleunigen.

Testdatengeneratoren und -editoren

Mit Hilfe von Testdatengeneratoren und -editoren können aus Datenbanken, Dateien oder Datenströmen zunächst Testdaten ermittelt werden und dann sämtliche für einen Test benötigte Testdaten bearbeitet werden. Ein Nutzen dieser Werkzeuge besteht darin sicherzustellen, dass die auf diese Art und Weise in der Testumgebung verwendeten Produktionsdaten aus Gründen des Datenschutzes anonymisiert werden.

6.1.5 Werkzeugunterstützung für die Testdurchführung und die Protokollierung (K1)

Testausführungswerkzeuge

Testausführungswerkzeuge ermöglichen eine automatische oder halbautomatische Ausführung von Tests unter Verwendung der aufgezeichneten Eingaben und der erwarteten Ausgaben, mittels einer skriptbasierten Sprache. Durch eine skriptbasierte Sprache ist es möglich, dass die aufgezeichneten Tests mit geringem Aufwand modifiziert werden können, um die Tests mit ähnlichen Daten zu wiederholen oder andere Teile des Systems mit ähnlichen Testschritten zu testen. In der Regel enthalten solche Werkzeuge Funktionen zum dynamischen Vergleich und liefern für jeden Testlauf ein Protokoll.

Testausführungswerkzeuge können ebenso zur Aufzeichnung von Tests verwendet werden, wobei diese als Mitschnittwerkzeuge (engl. Capture /Playback- oder Capture/ Replay Tools) bezeichnet werden. Die Aufzeichnung der Testeingaben kann auch im Rahmen von explorativen Tests nützlich sein, um einen Test reproduzieren und/oder dokumentieren zu können, falls ein (äußerer) Fehler auftritt.

Testrahmen (E)

Ein Testrahmen kann den Test einer Komponente oder Teilen eines Systems durch Simulation der Umgebung des Testobjekts ermöglichen. Er wird eingesetzt entweder weil einige Komponenten der Umgebung noch nicht zur Verfügung stehen, weil sie zwischenzeitlich durch Platzhalter und/oder Testtreiber ersetzt werden oder einfach, um eine kontrollierte Umgebung für die Lokalisierung von Fehlern in den Testobjekten zur Verfügung zu haben.

Ein Testrahmen wird erzeugt, wo Teile des Codes, des Objekts, Methode oder Funktion, Einheit oder Komponente ausgeführt werden können. Dieses geschieht durch Aufruf des zu testenden Objektes und/oder durch Auswertung der Rückmeldung an das Objekt. Dies wird durch die Bereitstellung synthetischer Eingaben für die Testobjekte und die Ersetzung der realen Ausgaben an Platzhalter durchgeführt.

Mit Hilfe eines Testrahmens kann auch die Testdurchführung in der Middleware hinsichtlich des Zusammenspiels von verschiedenen Programmiersprachen, Betriebssystemen und Hardware getestet werden.

Testrahmen werden auch als Komponententestrahmen bezeichnet, wenn sie einen besonderen Fokus auf den Komponententest haben. Diese Art von Werkzeugen unterstützt die Durchführung des Komponententests parallel zur Erzeugung des Objektcodes.

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 somit auch als Testorakel verwendet werden, insbesondere wenn der Vergleich automatisiert erfolgt.

Überdeckungsanalysatoren (E)

Überdeckungsanalysatoren können in Abhängigkeit des verwendeten Messverfahrens, zu ermittelnden Metriken und der Quellcode-Sprache als intrusiv oder nicht-intrusiv bezeichnet werden. Überdeckungsanalysatoren messen den Grad der Überdeckung eines Strukturelements (z.B. Anweisungen, Zweige oder Entscheidungen), oder von spezifischen Typen von ausgeführten Programmstrukturen (z.B. Anweisungen, Zweige, oder Entscheidungen, Module oder Funktionsaufrufe). Diese Werkzeuge zeigen, wie umfassend ein gemessenes Strukturelement durch eine Menge von Tests abgedeckt wurde.

Sicherheitsprüfwerkzeuge

Sicherheitsprüfwerkzeuge prüfen die Absicherung des Rechners oder ganzen Netzwerks z.B. gegen Computerviren und sog. Denial of Service Attacks (Angriff eines Hackers auf einen Rechner, der darauf abzielt, den Computer oder Teile seiner Dienste zu blockieren). Eine Firewall ist beispielsweise kein Testwerkzeug im engeren Sinne, aber sie kann im Rahmen von Sicherheitsprüfungen eingesetzt werden. Andere Sicherheitsprüfwerkzeuge versuchen spezielle Sicherheitslücken im System aufzudecken.

6.1.6 Werkzeugunterstützung für Performanzmessungen und Testmonitore (K1)

Dynamische Analysewerkzeuge (E)

Dynamische Analysewerkzeuge decken lediglich Fehler auf, wie sie 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. Sie simulieren die Last auf eine Applikation, eine Datenbank oder eine Systemumgebung wie z.B. ein Netzwerk oder Server. Diese Art der Werkzeuge werden oft nach dem Performanzkriterium benannt, das sie messen und sind auch als Lasttestwerkzeuge oder Stresstestwerkzeuge bekannt. Sie basieren oft auf der durch Parameter gesteuerten wiederholten Ausführung von Tests.

Testmonitore

Testmonitore sind keine Testwerkzeuge im engeren Sinne, aber sie stellen Informationen für Testzwecke zur Verfügung, die nicht auf anderem Wege bereitgestellt werden können.

Testmonitore analysieren, verifizieren und zeichnen kontinuierlich die Verwendung von spezifischen Systemressourcen auf, und geben Warnungen bzgl. möglicher Probleme in der Verwendung von Diensten aus. Sie zeichnen darüber hinaus die Version der verwendeten Software und Testmittel auf und ermöglichen somit eine Rückverfolgbarkeit.

6.1.7 Werkzeugunterstützung für spezifische Anwendungsbereiche (K1)

Einzelne Beispiele von Testwerkzeugarten können danach klassifiziert werden, dass sie bestimmte Typen von Applikationen unterstützen. Beispielsweise existieren Performanzwerkzeuge speziell für web-basierte Applikationen, statische Analysewerkzeuge für bestimmte Entwicklungsplattformen und dynamische Analysewerkzeuge für die Prüfung von spezifischen Sicherheitsaspekten.

Kommerzielle Werkzeug-Suiten können auf bestimmte Anwendungsgebiete (z.B. eingebettete Systeme) abzielen.

6.1.8 Werkzeugunterstützung unter Verwendung allgemeiner Werkzeuge (K1)

Die folgenden Testwerkzeuge werden nicht nur von Testern verwendet: z.B. Tabellenblätter, SQL, Debugger (E).

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

Begriffe
Datengetriebener Test, Skriptsprache, Schlüsselwortgetriebener Test

6.2.1 Potentieller 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 erheblichen potentiellen (oder möglichen) Nutzen und neue Chancen ermöglichen bzgl. der Unterstützung des Testens. Allerdings birgt der Einsatz auch Risiken, die berücksichtigt werden müssen.

Potentieller Nutzen der Verwendung von Testwerkzeugen beinhaltet:

- Sich wiederholende Tätigkeiten werden reduziert (z.B. für Regressionstestläufe, wiederholte Eingaben der gleichen Testdaten und Prüfungen gegen Programmierkonventionen)
- Bessere Konsistenz und Wiederholbarkeit (z.B. die gleichen Tests werden durch ein Werkzeug ausgeführt und aus Anforderungen hergeleitet)
- Objektive Messungen 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 beinhalten:

- Unrealistische Erwartungen an das Werkzeug (einschließlich Funktionalität und Benutzungsfreundlichkeit)
- Unterschätzung der Zeit, der Kosten und des Aufwands für die initiale Einführung eines Werkzeugs (einschließlich Training und externer Beratung)
- Unterschätzung der Zeit und des Aufwands, um einen signifikanten und anhaltenden Nutzen aus der Anwendung eines Werkzeuges ziehen zu können (einschließlich der Notwendigkeit von Änderungen im Testprozess und der kontinuierlichen Verbesserung in der Art und Weise wie das Werkzeug verwendet wird)
- Unterschätzung des erforderlichen Aufwands, die durch das Werkzeug erzeugten Ergebnisse zu warten
- Blindes Vertrauen in das Werkzeug (Ersatz für einen Testentwurf oder manuelles Testen)

6.2.2 Spezielle Betrachtungen zu einigen Werkzeugarten (K1)

Testausführungswerkzeuge

Testausführungswerkzeuge dienen u.a. der Wiederholung von elektronisch gespeicherten Tests, welche durch Skripte gesteuert werden. Testwerkzeuge dieser Art erfordern oft einen erheblichen Aufwand um einen signifikanten Nutzen zu erzielen.

Die Aufzeichnung der Aktionen eines manuellen Testers scheint auf den ersten Blick attraktiv, aber diese Vorgehensweise ist nicht auf eine größere Anzahl von Tests skalierbar. Ein aufgezeichnetes Skript ist eine lineare Repräsentation von spezifischen in das Skript integrierten Daten. Skripte dieser Art sind nicht stabil, sobald während der Testdurchführung unerwartete Ereignisse auftreten.

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

daten über ein Tabellenblatt eingeben und damit die Ausführung von vordefinierten Testskripts steuern.

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

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

Welches skriptbasierte Verfahren auch immer verwendet wird, die erwarteten Ergebnisse für jeden Test müssen für einen späteren Vergleich bereits abgelegt sein.

Performanztestwerkzeuge

Performanztestwerkzeuge benötigen einen Mitarbeiter mit detaillierten Kenntnissen in Performanztesten, um die Tests zu entwerfen und die Ergebnisse zu interpretieren.

Statische Analysewerkzeuge

Statische Analysewerkzeuge dienen der Analyse des Quellcodes und können die Einhaltung von Programmierkonventionen überprüfen. Jedoch ist zu berücksichtigen, dass bei Anwendung eines statischen Analysators für existierenden Quellcode sehr viele Meldungen erzeugt werden können. Wird ein Compiler mit integriertem statischen Analysator oder mit entsprechenden Compileroptionen 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 beinhalten, 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 Informationen nach den Bedürfnissen der Organisation aufzubereiten. Berichte sollten nach dem bestmöglichen Nutzen für die Organisation entworfen und ausgewertet werden können.

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 beinhalten:

- Bewertung der Reife einer Organisation, Analyse der Stärken und Schwächen und die Identifikation von Möglichkeiten für die Verbesserung des Testprozesses unterstützt durch Testwerkzeuge
- Evaluation gegen klar spezifizierte Anforderungen und objektive Kriterien (bzgl. des Nutzen und Anwendung)
- Evaluation des Werkzeuges, um die geforderte Funktionalität zu prüfen und um zu ermitteln, ob das Produkt die gesetzten Ziele erfüllen kann
- Evaluation der Anbieter (einschließlich Trainingsunterstützung, Support und der kommerziellen bzw. vertraglichen Aspekte)
- Identifikation der internen Anforderungen für das Coaching und die Schulung der Anwendung des Werkzeugs

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

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

Erfolgsfaktoren der Inbetriebnahme innerhalb einer Organisation beinhalten:

- Das Werkzeug wird schrittweise in der ganzen Organisation in Betrieb genommen.
- Adaptierung und Prozessverbesserung müssen mit dem Werkzeug harmonieren.
- Für neue Anwender müssen Trainingsmaßnahmen und Coaching bereitgestellt werden.
- Richtlinien für die Werkzeugbenutzung müssen definiert werden.
- Es müssen Verfahren gefunden und implementiert werden, um aus dem Werkzeugeinsatz lernen zu können.
- Werkzeugverwendung und der tatsächliche Nutzen sind zu überwachen bzw. zu beobachten.

Referenzen
6 Linz, 2005
6.2.2 Buwalda, 2001, Fewster, 1999
6.3 Fewster, 1999

7 Referenzen

Standards

ISTQB® Standard Glossary of terms used in Software Testing, Version 1.3

[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 829] IEEE Std 829™ (1998/2007) IEEE Standard for Software Test Documentation (currently under revision)
See sections 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™ (1997) IEEE Standard for Software Reviews
Siehe Kapitel 3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-1996, Software life cycle processes
Siehe Kapitel 2.1

[ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality
Siehe Kapitel 2.3

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 (2nd edition), John Wiley & Sons: 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, John Wiley & Sons:
Siehe Kapitel 1.1, 4.5, 5.2

[Liggesmeyer, 2002] Liggesmeyer, P. (2002) Software-Qualität, Spektrum-Verlag, Heidelberg, Berlin,

[Linz, 2005] Linz, T und Spillner, A. (2005) Basiswissen Softwaretest, Aus- und Weiterbildung zum Certified Tester Foundation Level nach ISTQB® -Standard, 3., überarbeitete Auflage, dpunkt.Verlag Heidelberg,
Bildet den Inhalt des deutschsprachigen Lehrplans vollständig ab.

[Myers 1982] Myers, Glenford J. (1982) Methodisches Testen von Programmen, Oldenbourg Verlag:
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: The Netherlands
Siehe Kapitel 3.2, 3.3

8 Anhang A – Hintergrundinformation zum Lehrplan

Zur Geschichte dieses Dokumentes

Dieses Dokument wurde in den Jahren 2004 – 2005 durch eine Arbeitsgruppe des International Software Testing Qualifications Board (ISTQB®) 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 eine essentielle und professionelle 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
- Unterstützung von Tester über die Landesgrenze hinaus einfacher Arbeit zu finden
- Durch ein gemeinsames Verständnis des Testens internationale Projekte unterstützen bzw. ermöglichen zu können
- 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 somit das Wissen über das Testen bei allen Beteiligten zu erhöhen
- Testen als Beruf in weiteren Ländern zu verbreiten
- Es Testern zu ermöglichen, eine international anerkannte Qualifikation in ihrer Muttersprache zu erlangen
- Den Wissens- und Ressourcenaustausch über Ländergrenzen hinweg zu ermöglichen
- Testern und diesem Ausbildungsgang internationale Anerkennung durch die Beteiligung möglichst vieler Länder zu verschaffen

Voraussetzungen zur Basisstufe

Voraussetzung zur Prüfung zum ISTQB® Basiszertifikat „Softwaretesten“ ist das Interesse des Kandidaten am Softwaretesten. Es sei den Kandidaten dennoch sehr empfohlen,

- zumindest ein minimales Hintergrundwissen im Bereich Softwareentwicklung oder Softwaretest zu haben (zum Beispiel sechs Monate Erfahrung als System- , 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 neue 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 dem Internationalen Zertifikat als gleichwertig anerkannt. Das Basiszertifikat besitzt kein Ablaufdatum und muss nicht erneuert werden.

In jedem teilnehmenden Land werden die lokalen Aspekte durch das jeweilige, vom ISTQB anerkannten, Software Testing Board kontrolliert. Die Pflichten der nationalen Boards sind durch das ISTQB spezifiziert, müssen jedoch durch die einzelnen Länderorganisationen selbst umgesetzt werden. Dies beinhaltet 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 Inhalt wird entsprechend den zugeordneten Lernzielen geprüft.

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.

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

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

Taxonomiestufe 2: Verstehen (K2)

Der Lernende verknüpft oder transformiert Informationen. Typische beobachtbare Leistungen sind beschreiben, zusammenfassen, vergleichen, klassifizieren, begründen, erklären. Beispiele von Testkonzepten nennen.

Beispiel:
Beschreibt Gemeinsamkeiten und Unterschiede zwischen Integration und Systemtest:

- Gemeinsamkeiten: Testen mehr als eine Komponente, umfassen nicht-funktionale Aspekte
- 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:

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

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.

Beispiel:

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

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:

10 Anhang C – Verwendete Regeln bei der Erstellung des Lehrplans

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

Allgemeine Regeln

SG1. Der Lehrplan soll durch Personen mit 0 bis 6 (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)

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)

Lernziele

LO1. Lernziele unterscheiden zwischen Lerninhalten, welche erkannt (Kognitive Stufe K1), verstanden (K2) bzw. auf neue Aufgaben angewendet (K3) werden müssen. (WISSENSEBENE)

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

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. (ÜBER-SCHNEIDUNG)

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. (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. Z.B. die Definitionen von Begriffen sind im Glossar. Deshalb enthält der Lehrplan ausschließlich den Begriff. (KEIN-REF DETAIL)

Informationsquellen

Die verwendeten Begriffe im Lehrplan sind im „ISTQB® Standard Glossary of terms used in Software Testing" definiert, die von der ISTQB-Webseite (www.istqb.org) heruntergeladen werden kann.

Eine korrespondierende englisch-/deutschsprachige Fassung dieses Glossars mit der Bezeichnung „ISTQB® /GTB / TAV 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

Jede Überschrift eines Hauptkapitels im Lehrplan hat 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.

Die unten aufgeführten Inhalte des Lehrplans erfordern praktische Übungen:

4.3 Spezifikationsorientierte oder Black-Box-Verfahren (K3)
Der Kurs soll praktische Arbeiten (kurze Übungen) zu den vier Techniken Äquivalenzklassenbildung, Grenzwertanalyse, Entscheidungstabellentest und zustandsbasierter Test enthalten. Die Vorträge und Übungen sollen auf der entsprechenden Technik zugeordneten Referenzen basieren.

4.4 Strukturorientierter Test oder White-Box-Verfahren (K3)
Der Kurs soll praktische Arbeiten (kurze Übungen) beinhalten, welche überprüfen, ob eine gegebene Menge von Tests eine 100%ige Anweisungs- oder 100%ige Entscheidungsüberdeckung erreicht. Die Kandidaten erarbeiten aus gegebenen Kontrollflüssen Testfälle.

5.6 Abweichungsmanagement / Fehlermanagement(K3)
Der Kurs soll praktische Arbeiten (kurze Übungen) beinhalten, welche das Erstellen und Bewerten von Abweichungsberichten behandeln.

12 Anhang E – Release Note Lehrplan 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-1.4.1 abgedeckt
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 Anhang F – Index