Softwaretest

Lehrplan zum Aufbaukurs

ISTQB Certified Tester, Advanced Level

Version 1.2

Juli 2003

ISTQB™ Certified Tester Advanced Level

Inhaltsverzeichnis

Teil I.
Über diesen Lehrplan

1. Änderungsübersicht

Version	Datum	Bemerkung
1.0	Mai 2003	Erster Entwurf der Arbeitsgruppe zur Abstimmung im GTB
1.1	Juni 2003	Überarbeitung aufgrund des Reviews, zur Abstimmung im GTB
1.2	Juli 2003	Marginale Änderungen, Ergänzung Literaturverzeichnis

2. Autoren

Dieses Dokument wurde von der Arbeitsgruppe »ASQF-Certified-Tester, Advanced Level« des German Testing Board erstellt. Als Leiter dieser Arbeitsgruppe bedankt sich Dr. Dirk Meyerhoff bei seinen Mitstreitern Sabine Uhde, Graham Bath sowie Dr. Norbert Magnussen. Die Arbeitsgruppe bedankt sich für die Unterstützung des gesamten German Testing Boards, wobei die Anregungen von Tilo Linz und Horst Pohlmann besonders hervorzuheben sind.

3. Leitidee

Die Informatik hat heute in praktisch sämtlichen Lebensbereichen unserer Gesellschaft wie Wirtschaft, Soziales, Kultur, Freizeit etc. Einzug gehalten und spielt in der Abwicklung von Prozessen aller Art oftmals eine zentrale Rolle. Die Folge davon ist eine immer stärkere Abhängigkeit von Software mit der Konsequenz, dass IT-gestützte Systeme trotz zunehmender Komplexität unter allen Bedingungen einwandfrei und zuverlässig funktionieren müssen.

Deshalb ist es wichtig, dass Software und IT-Systeme systematisch geprüft werden, um ihr zuverlässiges Funktionieren vor dem Einsatz nachzuweisen und sicherzustellen.

Prüfobjekte sind IT-Systeme im kommerziellen und technischen Umfeld (z. B. Buchhaltung, Produktionsplanung, Maschinen- oder Fahrzeugsteuerung). Neben dem kompletten Endprodukt sind im Verlauf der Entwicklung auch einzelne Teile des IT-Systems (Teilsysteme, Komponenten, Module usw.) sowie Entwicklungsdokumente (Anforderungsspezifikation, Design, Anwenderdokumentation usw.) Prüfungen zu unterziehen.

3.1. Konsequenzen für die Ausbildung

Die damit verbundenen Prüf- und Testaufgaben sind komplex und erfordern neben einem guten Verständnis der jeweiligen Prüfobjekte ein umfassendes Verständnis und sicheres Beherrschen geeigneter Prüf- und Testtechniken. Dies erfordert qualifiziertes, gut ausgebildetes Fachpersonal.

3.2. Ziele der Ausbildung zum Certified Tester

Im Rahmen der Ausbildung zum Certified Tester werden entsprechende Kenntnisse und Techniken vermittelt, die bei entsprechender Umsetzung zu einem strukturierten, systematischen Vorgehen beim Prüfen und Testen führen und somit zur Qualitätsverbesserung der Software beitragen.

Die Ausbildung zum Certified Tester wendet sich an alle mit dem Softwaretesten befassten Personen, die ihre Kenntnisse auf eine fundierte Grundlage stellen oder ausbauen wollen. Angesprochen werden u. a.: QS-/Test-Professionals, Programmierer, Entwickler, Fachabteilungsmitarbeiter und (Projekt-)Manager, die Testaufgaben verantworten, planen, steuern oder ausführen.

- Certified Tester sind in der Lage Prüfungen und Tests projektspezifisch zu konzipieren und zu planen. Dazu müssen sie im Besonderen die Ausgangslage systematisch analysieren, Risiken in Bezug auf die Geschäftsprozesse beurteilen, und Prüfobjekte identifizieren.
- Certified Tester sind sich bewusst, dass im Rahmen von IT-Projekten ein Prüfungskonzept durch die gewählte Architektur, durch das Risiko der Geschäftsprozesse, die Reife der Technik und durch die Fähigkeiten der Entwicklungsorganisation bestimmt wird. Sie sind fähig adäquate Prüfziele zu definieren, Prüftechniken (Review- und Testtechniken) auszuwählen, die notwendigen Prüfaufgaben zu identifizieren und zeitlich festzulegen sowie die Ressourcen auszuwählen und vorzubereiten.
- Certified Tester realisieren Prüfungen und Tests, die den Entwicklungsprozess von Software und IT-Systemen begleiten. Sie führen die Tests durch, erstellen die zugehörigen Testprotokolle und leiten Fehlermeldungen an die Entwickler weiter.
- Certified Tester sind in der Lage Reviews für Dokumente (z. B. Datenmodell, Anforderungsspezifikation oder Programmcode) zu planen, zu organisieren, zu steuern und verantwortlich durchzuführen.
- Certified Tester können beurteilen, in welcher Form Tests durch Werkzeuge unterstützt werden können und welche Infrastruktur für Tests ausgewählt werden soll.
- Certified Tester sind sich bewusst, dass vollständiges Prüfen von Software nicht möglich ist. Testfälle müssen daher entsprechend der Wahrscheinlichkeit des Auftretens von Fehlern und dem damit verbundenen Risiko ausgewählt werden. Bei der Testplanung muss beurteilt werden, welche Softwareteile wie intensiv zu testen sind.

4. Hinweise zu Lehrplan und Prüfung

Der Lehrplan definiert:

- inhaltlich den prüfungsrelevanten Stoff (Begriffe, deren Definition bekannt sein muss, sind **hervorgehoben**);
- den zeitlichen Mindestumfang, in dem dieser Stoff in akkreditierten Trainingskursen vermittelt werden muss. Der Lehrstoff ist durch geeignete Beispiele und Übungen zu veranschaulichen bzw. zu vermitteln.

Der Lehrplan definiert nicht,

- in welcher (Kapitel-)Reihenfolge der Stoff in Trainingskursen vermittelt werden muss,
- in welchem zeitlichen Umfang in akkreditierten Trainingskursen Beispiele und Übungen enthalten sein müssen.

Die Prüfung ist als Multiple-Choice-Test gestaltet. Es wird dabei die Fähigkeit abgeprüft, den Stoff und die Prozesse in der Praxis anzuwenden.

Teil II.

Lehrplan

1. Einführung 45 min

Vorstellung der Leitidee zum Certified Tester (siehe oben), die Certified-Tester-Ausbildungsbausteine, Zusammenhang von Lehrplan, Training, Prüfung und Zertifikat, beteiligte Institutionen und internationale Zusammenhänge (Prüfungs- und Zertifizierungsstellen, Trainingsprovider und Akkreditierung, nationale Boards und ISTQB).

2. Grundlagen des Softwaretestens 90 min

2.1. Überblick

Überblick über die Grundlagen des Softwaretestens, welche im »ISTQB Certified Tester, Foundation Level« behandelt wurden.

Einführung in die Kernthemen, die Philosophie sowie die Problemstellungen des Software-Testens wie sie im Lehrplan zum Basiskurs aufgeführt sind.

Einführung in den Aufbaukurs »ISTQB Certified Tester, Advanced Level«.

2.2. Testen im Software-Lebenszyklus

Erläuterung der Einbettung des Testens in den unterschiedlichen Modellen der Softwareentwicklung:

- sequenziell (Wasserfallmodell, V-Modell)
- iterativ (inkrementelle und evolutionäre Entwicklung)
- Rapid Application Development (RAD)

Erläuterung, dass der **Testprozess** kein isolierter Prozess ist, sondern mit anderen Prozessen in Verbindung steht:

- **Projektmanagement**
- Konfigurations- und **Änderungsmanagement**
- Softwareentwicklung
- technischer Support
- Erstellung von technischen Dokumentationen.

Aufzeigen wie sich im **V-Modell** eine frühe **Testplanung** und die spätere **Testdurchführung** gegenüberstehen.

Erklärung der Unterschiede zwischen **Verifikation** und **Validierung**. Erläuterung, dass Verifikation und Validierung in den frühen Phasen des Softwarelebenszyklus erfolgen sollte und in Form von Reviews durchgeführt werden kann.

Aufzeigen der Rolle des Änderungs- und **Konfigurationsmanagements**.

Definition der folgenden **Teststufen**:

- **Komponententest**
- **Integrationstest**
- **Systemtest** (funktional und nicht funktional)
- Abnahme- bzw. **Akzeptanztest**.

Erläuterung, dass im Projektkontext auch andere **Teststufen** definiert werden können. Jede **Teststufe** besitzt folgende Eigenschaften:

- **Testziele**
- **Testumfang**
- Eingangs- und Ausgangskriterien
- zu liefernde **Testergebnisse**
- anzuwendende **Testtechniken**
- zu erhebende **Maße** und **Metriken**
- einzusetzende **Testwerkzeuge**
- einzuhaltende **Teststandards**

Erklärung, dass der Entwicklungsprozess Einfluss auf den **Testprozess** hat.

Erläuterung der Begriffe **Retest** (auch **Fehlernachtest**) und **Regressionstest**.

Retest ist die wiederholte Ausführung von Testfällen, die zu einem Fehler geführt haben, mit dem Ziel nachzuweisen, dass der Fehler beseitigt wurde.

Regressionstest ist der erneute Test eines bereits getesteten Programms bzw. einer Teilfunktionalität nach deren Modifikation, mit dem Ziel nachzuweisen, dass durch die vorgenommenen **Änderungen** keine Fehler eingebaut oder (bisher maskierte Fehler) freigelegt wurden.

3. Testprozess 150 min

3.1. Fundamentaler Testprozess

Kurze Darstellung des fundamentalen **Testprozesses**, wie er im »ISTQB Certified Tester, Foundation Level« behandelt wurde:

- Testplanung
- Testspezifikation
- Testdurchführung
- Testprotokollierung
- Testendebewertung

Erläuterung der Stellung des fundamentalen **Testprozesses** zum Softwareentwicklungsprozess.

3.2. Testplanung

Dieses Thema wird in Kapitel 4 im Detail behandelt.

3.3. Testspezifikation

Erklärung, dass die Wahl eines Verfahrens zur **Testspezifikation** von folgenden Faktoren abhängt:

- den abzusichernden **Risiken**
- dem Grad des Wissens über die zu testende Software
- der zur Verfügung stehenden Dokumentation

Auflistung einiger Verfahren zur **Testspezifikation**, wie sie z. B. in [Myers82], [Balzert00] oder [BS7925-2] beschrieben sind.

Erklärung der Phasen einer **Testspezifikation**:

- Testfallermittlung
- Testdatenerstellung
- Testskripterstellung

In der **Testplanung** werden die **Testobjekte** (beispielsweise die zu testenden Funktionen) identifiziert. In der Testfallermittlung werden die Testfälle für diese Testobjekte erstellt und qualitativ beschrieben. Die Testdaten konkretisieren die Testfälle. Die **Testskripte** machen die Testfälle ausführbar.

Kriterien für die Priorisierung und Risikoeinschätzung wie sie in der Risikoanalyse und dem **Testkonzept** festgelegt wurden, können sowohl in der Testobjektabgrenzung auf Testobjekte als auch im Testfallentwurf auf Testfälle angewendet werden.

Ein anerkannter Standard für die Form und den Inhalt von Testdokumentationen ist die Norm IEEE 829–1998 [IEEE829].

Erklärung, dass Testspezifikationstechniken für alle **Teststufen** angewendet werden können.

Im **Abnahmetest** können Testfälle z. B. aus der Analyse der Geschäftsprozesse und/oder Anwendungsfälle hergeleitet werden.

Erläuterung, welche Anforderungen an die **Testfallbeschreibungen** in den **Testspezifikationen** gestellt werden. Eine Testfallbeschreibung hat folgende Struktur:

- **Testeingaben**
- **Vorbedingungen** (Systemzustand, Datenbankzustand)
- zu testende Funktionen (Ziel des Testfalls)
- **Nachbedingungen** (erwartetes Ergebnis)
- **Ausgaben** (z. B. Fehlermeldungen)

Zur Ableitung von **Sollergebnissen** der zu testenden Funktionen ist die Kenntnis der Spezifikation der zu testenden Software notwendig.

Dieses Wissen ist oftmals nicht formal z. B. in Form von Fachkonzepten spezifiziert. Aus diesem Grund ist es manchmal notwendig, auf alternative Quellen zurückzugreifen wie z. B. auf das Know-how von Fachbereichsmitarbeitern oder Entwicklern.

Ein Beispiel für einen Entwicklungsprozess, bei dem formal spezifizierte **Anforderungen** nicht verlangt werden, ist RAD (Rapid Application Development).

Fehlt die Beschreibung der Sollergebnisse in einer Testspezifikation, so kann der Test zwar ausgeführt, aber nicht hinsichtlich richtig oder falsch bewertet werden.

Zusätzlich zu den Ausgaben ist der Endzustand der zu testenden Software sowie der Umgebung Bestandteil des Testergebnisses:

- **Ausgaben**
- **Nachbedingungen**

Erläuterung der **nichtfunktionalen Qualitätsmerkmale** nach [ISO9126]/[DIN66272]:

- **Zuverlässigkeit**
- **Effizienz** (Verbrauchs- und Zeitverhalten)
- **Benutzbarkeit**
- **Änderbarkeit**
- **Übertragbarkeit** (Portierbarkeit)

sowie ihrer Untermerkmale wie zum Beispiel:

- **Installierbarkeit**
- **Interoperabilität**
- **Konformität**
- **Sicherheit**
- **Wiederherstellbarkeit**

Die nichtfunktionalen Qualitätsmerkmale sollten ebenfalls in der Softwarespezifikation beschrieben und priorisiert sein.

Erklärung, dass **Reviews**, **Inspektionen** und **statische Analysen** ebenfalls dazu geeignet sind, Fehler zu finden, und dadurch die Tests ergänzen. Statt der zu testenden Software sind die Prüfgegenstände hier Dokumente wie zum Beispiel:

- Anforderungsspezifikationen
- Entwurfsspezifikationen
- Programmquellcode

3.4. Testdurchführung

3.4.1. Vorbereitung für die Testdurchführung

Erklärung der Vorbedingungen für die **Testdurchführung**:

- **Testspezifikation**, **Testdrehbuch** bzw. **Testskripte**
- Bereitstellung der **Testumgebung** (inkl. Räume, Arbeitsmittel, Personal, Hard- und Software, Werkzeuge, Peripheriegeräte, Kommunikationseinrichtungen, Zugriffsberechtigungen)
- Verantwortliche für die Erstellung und Wartung der Testumgebung sind benannt und stehen zur Verfügung
- **Konfigurationsmanagement**, **Abweichungsmanagement** und weitere unterstützende Bereiche sind eingerichtet
- Verifikation der **Testumgebung** (lauffähig, vollständig, korrekt im Gut- und Fehlerfall)

3.4.2. Durchführung der Tests

Erklärung, dass zur Sicherstellung der Prüffähigkeit von Tests und der Wiederholbarkeit von Tests im **Retest** und **Regressionstest** die Einhaltung der **Testdrehbücher und Testskripte** notwendig ist.

Es sollte für die Tester einen formalen Weg geben weitere Tests, welche nicht im Testdrehbuch dokumentiert sind, vorschlagen und durchführen zu können.

Um das Vertrauen in die Tests und Testverfahren zu erhöhen, kann es manchmal notwendig sein beispielsweise Vertreter der anfordernden Fachabteilung oder des Auftraggebers mit den Vorgehensweisen im Test bekannt zu machen.

3.5. Testprotokollierung

3.5.1. Testüberprüfung

Erklärung, dass jede Art von **Abweichung** zwischen Ist- und Sollergebnis festgehalten und analysiert werden muss, unabhängig von der Ursache für die Abweichung. Mögliche Ursachen können z. B. sein:

- Fehler in der Software
- Fehler in den Testdaten
- Fehler in der Testumgebung
- Fehler in der Anforderungsspezifikation
- etc.

Für die Analyse der Fehlerursache ist es hilfreich, wenn die gemachten Eingaben aufgezeichnet wurden, so dass genau nachvollziehbar ist, welche Aktivitäten der Tester durchgeführt hat. Die Aufzeichnungen müssen nicht notwendigerweise automatisiert erfolgen, sondern können auch notiert oder durch Hardcopies belegt werden.

3.5.2. Testergebnisdokumentation

Erklärung, dass jede Ausführung eines Tests zum Zwecke der Prüfung (z. B. durch die Revision) und Nachvollziehbarkeit aufgezeichnet wird. Die Aufzeichnungen gelten als Beleg für:

- den erreichten Grad der **Testabdeckung**
- die Prüfung gegen **Vollständigkeits- und Abbruchkriterien**

Erläuterung, welche Anforderungen an **Testprotokollierungen** im **Komponententest** gestellt werden und wie diese Anforderungen auf andere Teststufen übertragbar sind. Anforderungen an **Testprotokollierungen** im Komponententest:

- eindeutige **Identifikation** der getesteten **Komponenten** und deren **Version**
- eindeutige **Identifikation** des dazugehörenden **Testfalls**

Mindestanforderungen für Testprotokollierungen in anderen Teststufen sind:

- eindeutige **Identifikation** der getesteten **Softwareversion**
- eindeutige **Identifikation** der dazugehörenden **Testfälle**

3.6. Prüfung auf Testendekriterien

Erklärung, dass der Testprozess erst dann beendet ist, wenn die **Testendekriterien** erfüllt sind.
Erläuterung von **Testendekriterien** für die verschiedenen **Teststufen**.

Sind die **Testendekriterien** nicht erfüllt, so müssen normalerweise noch weitere Tests durchgeführt werden und gegebenenfalls weitere spezifiziert werden.

Alternativ dazu können auch die Kriterien im **Testkonzept** angepasst werden. Die Anpassung kann in beide Richtungen (Abschwächung, Verstärkung) erfolgen. Eine Verschärfung der Kriterien ist sinnvoll, wenn sich im Test unerwartete Schwachstellen aufzeigen.

Bevor **Testendekriterien** geändert und im **Testkonzept** dokumentiert werden, sollten damit verbundene **Risiken** abgewogen werden. Desweiteren müssen diese Änderungen mit den beteiligten verantwortlichen Personen abgestimmt werden.

4. Testmanagement 390 min

4.1. Testmanagement-Dokumentation

Beschreibung der folgenden Dokumenttypen:

- Testpolitik
- Testhandbuch
- Testkonzept
- Teststufenplan

Die **Testpolitik** ist ein Dokument, in welchem die Unternehmensphilosophie in Bezug auf das Testen (oder die Qualitätssicherung) von Software beschrieben ist.

Das **Testhandbuch** ist ein Rahmendokument, in dem die generell durchzuführenden **Teststufen** und die in ihnen auszuführenden **Testaktivitäten** beschrieben sind.

Das **Testkonzept** beschreibt die durchzuführenden **Teststufen** und die Testaktivitäten in den **Teststufen** für ein bestimmtes Projekt.

Der **Teststufenplan** detailliert den Verfahrensansatz für eine Teststufe und beschreibt die Implementierung des **Testkonzeptes** für eine bestimmte **Teststufe**.

In manchen Organisationen können die vier oben erwähnten Dokumenttypen zusammengefasst oder auf weitere Dokumenttypen aufgeteilt sein. Insgesamt sollten sie jedoch die gleichen Informationen erhalten.

4.1.1. Testpolitik

Erklärung, dass der organisatorische Ansatz für das Testen mit der Testpolitik beginnt. Diese wird normalerweise durch die IT-Abteilung oder eine vergleichbare Abteilung entwickelt, spiegelt aber die Philosophie des gesamten Unternehmens wieder.

Die **Testpolitik** ist im Allgemeinen eine Ergänzung oder ein Bestandteil der **Qualitätspolitik**. Die Qualitätspolitik beschreibt die grundlegenden Absichten und Zielsetzungen eines Unternehmens, die von der Unternehmensleitung hinsichtlich Qualität verfolgt werden.

Erklärung, dass die Testpolitik ein Rahmendokument darstellt, das folgende Inhalte besitzt:

- Definition von Testen (z. B. Überprüfung, dass die Software ein Geschäftsproblem löst)
- Darstellung des **Testprozesses** (z. B. Konzeption und Durchführung eines Tests in Übereinstimmung mit Abteilungsprozessen und Benutzeranforderungen)
- Evaluierung des Testens (z. B. Messung der Kosten von Fehlern, die nach der Freigabe entdeckt werden)
- zu erreichende **Qualitätsniveaus** (z. B. nicht mehr als ein Fehler mit dem höchsten Schweregrad pro 1000 Zeilen Code in den ersten 6 Monaten nach Einführung)
- den Verfahrensansatz zur **Testprozessverbesserung** (z. B. nach jedem Abschluss eines Projektes werden Post-Projekt-Reviews durchgeführt zur Erreichung eines CMM-Levels von 3)

Die Testpolitik bezieht sich auf Testaktivitäten sowohl für Neuentwicklungen als auch für die Wartung.

4.1.2. Testhandbuch

Das Testhandbuch basiert auf der Testpolitik.

Der Anwendungsbereich des **Testhandbuchs** erstreckt sich auf die generischen Testanforderungen für eine Organisation.

Erläuterung, dass das **Testhandbuch** die **Risiken** anspricht und einen Prozess beschreibt, mit dem diese Risiken in Übereinstimmung mit der Testpolitik abgeschwächt werden. Die Verbindung zwischen Risiken und Testen wird explizit aufgezeigt.

Ein **Testhandbuch** besteht typischerweise aus zwei Hauptteilen:

- der Darstellung der Risiken, die durch das Testen der Software abgedeckt werden sollen
- der spezifischen Testaktivitäten, die durchgeführt werden, um die identifizierten Risiken abzudecken.

Erklärung, dass ein typisches **Testhandbuch** eine Beschreibung der anzuwendenden **Teststufen** enthält.

Für jede Teststufe werden folgende Aspekte grob umrissen:

- die Eingangs- und Ausgangskriterien
- der Testansatz (top-down, bottom-up, prioritätsgetrieben)
- die anzuwendenden Testspezifikationstechniken
- die Testendekriterien
- der Grad der Unabhängigkeit des Testens
- einzuhaltende Standards
- die Umgebung, in der Softwaretests durchgeführt werden
- der Ansatz zur Testautomation
- der Grad der Wiederverwendung von Software
- der Ansatz zu Retests und Regressionstests
- der anzuwendende Testprozess inklusive der Testergebnisse wie z. B. Testreports
- die aufzuzeichnenden Maße und Metriken
- der Ansatz für das anzuwendende Abweichungsmanagement

Das **Testhandbuch** muss nicht notwendigerweise aus einem Dokument bestehen, sondern kann über einen Satz von Dokumenten verteilt sein, z. B. Unternehmenstesthandbuch, Niederlassungstesthandbuch, Abteilungstesthandbuch, Projekttesthandbuch.

Erläuterung, warum unterschiedliche **Testhandbücher** für unterschiedliche Anwendungsgebiete angemessen sein können und an einem Beispiel erklären, z. B. sicherheitskritische Anwendungen und unkritische Anwendungsgebiete.

Erläuterung, dass Maßnahmen zur Verbesserung des **Testprozesses** in höheren Versionen von **Testhandbüchern** berücksichtigt werden.

4.1.3. Testkonzept

Erklärung, dass ein Testkonzept die konkrete Anwendung des Testhandbuches für ein bestimmtes Projekt darstellt.

Abweichungen vom **Testhandbuch** werden im **Testkonzept** dokumentiert und erläutert.

Das **Testkonzept** wird vom Projektplan referenziert.

Erläuterung, dass ein **Testkonzept** zusätzliche Informationen enthält, die es dem Tester erlauben:

- Projektkosten und -testzeiten zu bestimmen, um Freigabegenehmigungen für Budget und Ressourcen zu erhalten
- Testzyklen basierend auf dem Software-Releaseplan zu identifizieren das Management und die Anwender von der Notwendigkeit des Testens zu überzeugen
- Beistellleistungen, die von anderen Personen oder Abteilungen erbracht werden sollen, zu definieren und zu kommunizieren
- die zu testenden Projektergebnisse zu identifizieren

4.1.4. Teststufenplan

Erläuterung, dass der **Teststufenplan** die detaillierte Vorgehensweise für eine Teststufe beschreibt. Er stellt die Verfeinerung des Testkonzeptes für eine Teststufe dar.

Der Teststufenplan beschreibt die Implementierung des Testkonzeptes für eine bestimmte Teststufe. Normalerweise enthält er eine Abfolge der Testaktivitäten in der jeweils betrachteten Teststufe sowie einen terminierten Plan der Aktivitäten und damit verknüpfter Meilensteine.

4.2. Testkonzept-Dokumentation

Erklärung, wie ein Testkonzept bzw. Teststufenpläne in Übereinstimmung mit dem Standard [IEEE829] entwickelt werden können.

4.3. Testaufwandsschätzung

Aufwandsschätzung ist eine näherungsweise Beurteilung des Testaufwandes auf der Basis von Erfahrungswerten.

Es ist notwendig, folgende Aspekte bei der **Aufwandsschätzung** zu berücksichtigen:

- die Anzahl der Tests
- die benötigte Zeit je Teststufe
- die Anzahl der Testwiederholungen je Teststufe

Alle Aktivitäten des Testprozesses werden bei der Aufwandsschätzung berücksichtigt, d. h. sowohl die Zeit für die Testdurchführung und Testvorbereitung als auch die Zeit für die Testplanung.

Erklärung, warum bei unbekannter oder schlechter Softwarequalität der Aufwand bis zur Erreichung der Testendekriterien schwer vorhersagbar ist.

Die Basis für die Aufwandsschätzung kann sein:

- Intuition, Raten
- Erfahrung
- Firmenstandards
- eine detaillierte Aufschlüsselung für alle Testaktivitäten
- Formelbasiert:
- Function-Point-Methode
- Testpunkte
- relativ zum Entwicklungsaufwand (z. B. 40% bei einer Neuentwicklung von Software)
- Metriken
- vergleichbare neuere Testprojekte zur Abschätzung von Retestzyklen
- Berechnung des mittleren Aufwandes pro **Testobjekt** oder **Testfall** aus einem vorhergehenden Testdurchlauf und Multiplikation mit der geschätzten Anzahl von Testobjekte oder Testfälle im aktuellen Testdurchlauf

4.4. Festlegung der Testplanung

Erläuterung, warum die **Testplanung** so früh wie möglich durchzuführen ist, z. B. um Personen, von denen Beistellleistungen zu erbringen sind, möglichst frühzeitig zu informieren und ihnen genügend Zeit zur Vorbereitung zu geben; des Weiteren um Probleme frühzeitig sichtbar zu machen.

Erläuterung, wie die **Testplanung** bei der Vorbereitung des Entwicklungsplanes für die Softwarekomponenten unterstützend wirken kann. Beispielsweise kann eine hohe Priorität eines **Testobjektes** dazu benutzt werden, zu entscheiden, welche Komponenten zuerst entwickelt und geliefert werden.

Erklärung der Vorteile, **Testpläne** stufenweise freizugeben, wenn nicht alle Informationen vorliegen und Verzögerungen vermieden werden sollen.

4.5. Testfortschrittsüberwachung und -kontrolle

Erklärung unterschiedlicher Verfahren zur Steuerung des Testfortschritts inkl. der Überwachung von **Abweichungen** und **Testfällen** sowie der Nutzung statistischer Methoden.

Testfortschrittsberichte werden genutzt, um den Testfortschritt zu kommunizieren. Die Berichte müssen aber auf die jeweiligen Adressaten (Tester, Management usw.) zugeschnitten sein.

Erläuterung, wie der Testfortschritt graphisch und tabellarisch aufbereitet werden kann.

Identifizierung von Möglichkeiten, steuernd auf den **Testprozess** Einfluss zu nehmen, um die Abweichungen zum Testplan möglichst gering zu halten, z. B. durch:

- Korrekturmechanismen einschließlich der Überarbeitung von Testprioritäten
- Heranziehung zusätzlicher Ressourcen
- Verschiebung des Freigabetermins unter Beteiligung des Projektmanagements
- Änderung von **Testendekriterien**

5. Risikoorientiertes Testen 240 min

5.1. Einführung in Risikoorientiertes Testen

Erklärung des Begriffes Risiko als Wahrscheinlichkeit dafür, dass ein Problem in der Zukunft auftreten kann.

Intention des **Risikoorientierten** Testens: Unter gegebenen Rahmenbedingungen (Aufwand, Zeit, Verfügbarkeit usw.) die Tests so priorisieren, dass die gegebenen **Risiken** möglichst minimiert werden.

Erklärung, dass **Risiko** eine Kombination aus der Wahrscheinlichkeit des Auftretens eines Problems und den damit verbunden Auswirkungen darstellt.

Risiken lassen sich unterteilen in **Produkt-** und **Projektrisiken**. Ein **Produktrisiko** ist z. B. ein Softwarefehler, der zu einem Versagen eines Systems führen kann, ein **Projektrisiko** ist die Nichteinhaltung des Liefertermins.

Darstellung typischer Produkt- und Projektrisiken sowie Risiken bezogen auf:

- Betriebssicherheit
- Datensicherheit
- geschäftsbezogene Risiken
- technische Faktoren
- politische Faktoren

Erklärung, dass **Risiken** sowohl qualitativ als auch quantitativ erfasst werden können.

Erklärung, dass eine **Risikoanalyse** genutzt werden kann, um:

- zielgerichtet zu testen: unterschiedliche Risiken werden mit unterschiedlichen **Testverfahren** und **Testtiefen** abgedeckt
- priorisiert zu testen: Bereiche mit höherem Risiko erhalten eine höhere Priorität
- die Risiken einer Softwarelieferung aufzuzeigen: bei einer Verkürzung des Tests oder bei einem Verzicht auf dessen Ausführung bestehen die nicht durch das Testen abgedeckten Risiken weiterhin.

Testen kann dazu genutzt werden, um:

- **Risiken** zu mindern – ein grundlegender Weg zur Reduzierung von Produktrisiken ist das Finden von Fehlern in einem Produkt und deren Beseitigung. Projektrisiken können durch die Umsetzung eines geeigneten **Testkonzeptes** reduziert werden.
- über die **Risikobewertung** zu informieren – hohe bzw. niedrige Fehlerraten in bestimmten Bereichen des Softwareproduktes können benutzt werden, um die Güte der **Risikoanalyse** zu verbessern.

5.2. Risikomanagement

5.2.1. Einführung in Risikomanagement

Erläuterung der Kernaktivitäten des **Risikomanagements**:

- Risikoidentifikation
- Risikoanalyse
- Risikovermeidung

Idealerweise sind alle Projektbeteiligten in allen Phasen und Stufen in das **Risikomanagement** eingebunden.

5.2.2. Risikoidentifikation

Folgende Techniken und Hilfsmittel können zur Identifikation von Risiken angewendet werden:

- Experteninterviews
- unabhängige Einschätzungen (Assessments)
- Verwendung von Risikotemplates
- Durchführung von Risikoworkshops
- Brainstorming
- Checklisten
- Anwendung von Erfahrungen aus der Vergangenheit

Aufzeigen von Beispielen für die oben genannten Techniken und Hilfsmittel.

5.2.3. Risikoanalyse

Erläuterung der Risikoanalyse als Studie der identifizierten Risiken.

Erklärung, dass ein **Risiko** quantitativ berechnet werden kann, wenn sich die Wahrscheinlichkeit des Auftretens (W) eines Risikos und der damit verbundene Schaden (S) quantitativ beziffern lassen. Das **Risiko** berechnet sich dann nach der Formel $W \times S$.

In den meisten Fällen sind aber Wahrscheinlichkeit und Schaden nicht quantifizierbar, sondern nur der Tendenz nach anzugeben (z. B. hohe Wahrscheinlichkeit, geringe Wahrscheinlichkeit, hoher Schaden, mittlerer Schaden).

Das **Risiko** wird definiert als eine Abstufung innerhalb einer Anzahl von Klassen oder Kategorien.

Liegen keine verlässlichen Metriken vor, so beruht die Analyse auf empfundenen Wahrscheinlichkeiten und Schadenseinschätzungen. Da diese Risikoeinschätzungen normalerweise auf

persönlicher Empfindung beruht, sind die Ergebnisse in Abhängigkeit der beurteilenden Personen unterschiedlich.

Erläuterung der unterschiedlichen Sichtweisen eines Projektmanagers, eines Entwicklers, eines Testers und eines Anwenders.

Aus den Ergebnissen einer **Risikoanalyse** sollte der Grad der Unsicherheit erkennbar sein, mit der die Bewertung der **Risiken** erfolgte.

Zur Bewertung der **Risiken** von Softwareprodukten müssen im Allgemeinen gröbere **Risikokategorien** zugrunde gelegt werden.

Übliche Kategorien beziehen sich auf:

- Funktionale Aspekte (besonders kritische Funktionen oder Geschäftsprozesse)
- Nichtfunktionale Aspekte wie z. B. Performance, Sicherheit, Benutzbarkeit etc.

Erklärung, dass es vielfältige Ansätze zur **Risikoanalyse** mit unterschiedlichen Exaktheitsgraden gibt; von der quantitativen Berechnung bis hin zur einfachen Bestimmung einer Risikoklasse.

Aufzeigen von Beispielen für **qualitative** und **quantitative** Risikobewertung.

Risikoanalyse ist ein permanenter Prozess während des gesamten Projektverlaufes. Testergebnisse können als Eingaben für die **Risikoanalyse** genutzt werden und zur Neubewertung von **Risiken** führen. Unsicherheitsgrade können neu bestimmt werden.

Wird z. B. eine erhöhte Anzahl von Fehlern in einer Softwarekomponente gefunden, so ist die Qualität dieser Komponente geringer als erwartet. Die Ausfallwahrscheinlichkeit dieser Komponente erhöht sich und damit auch das Ausfallrisiko aller mit dieser Komponente verbundenen Teilsysteme des Softwareprodukts.

Zur Verringerung dieses erhöhten Risikos könnte eine Ausweitung des Tests der betroffenen Komponente eine Maßnahme sein.

5.2.4. Risikovermeidung

Erklärung, dass Risikovermeidung Aktivitäten umfasst, die als Reaktion auf die analysierten Risiken ergriffen werden.

Mögliche Reaktionen auf ein erkanntes Risiko können sein:

- Nichtstun
- Verteilung des Risikos
- Ergreifung von präventiven Maßnahmen zur Umgehung oder Reduzierung des **Risikos**
- Aufstellen eines Notfallplans für den Fall, dass der Schaden eintritt

Die Wahl einer geeigneten Reaktion auf ein **Risiko** ist abhängig von dem Nutzen, der mit der Beseitigung oder Reduzierung des **Risikos** verbunden ist.

Erklärung, dass Testen eine vorbeugende Maßnahme darstellt, **Risiken** durch das Finden und Beheben von Fehlern zu reduzieren.

Erklärung, dass das Wissen um die Risiken genutzt werden kann, den Inhalt des Testkonzeptes festzulegen.

Da in der Regel das Budget begrenzt ist, werden Risiken normalerweise priorisiert. Tests, die ein hohes Risiko abdecken, werden dann zeitlich zuerst ausgeführt. Dieses Wissen kann auf zwei Arten genutzt werden:

- Bestimmung der **Intensität** der durchzuführenden Tests aus dem Risikograd. Die meisten Sicherheitsstandards verwenden diesen Ansatz und schreiben für jede Risikostufe die anzuwendenden **Testfallentwurfstechniken** sowie die **Testendekriterien** vor. Zum Beispiel wird für Komponenten, die der höchsten Sicherheitsstufe angehören, eine hundertprozentige **Zweigüberdeckung** im Test gefordert.
- Bestimmung des geeigneten **Testverfahrens** aus der Art des **Risikos**. So kann das **Risiko**, welches in Zusammenhang mit der Benutzeroberfläche eines Softwareproduktes steht, durch intensive Benutzbarkeitstests gemindert werden.

Das kann bedeuten, dass Risiken niedrigerer Priorität im Test nicht abgedeckt werden, wenn Budget oder Zeit im Projekt nicht mehr zur Verfügung stehen.

Die nicht abgedeckten Risiken sind dem Projektmanagement bekannt. Vor diesem Hintergrund muss entschieden werden, ob das Softwareprodukt freizugeben ist und das verbleibende Risiko akzeptiert wird.

6. Testtechniken 1200 min

6.1. Funktionale Techniken

Einführung in die Testfallermittlungstechniken nach [BS7925-2]:

- **Klassifikationsbaummethode** [Grochtmann93]
- **Äquivalenzklassenmethode**
- **Grenzwertanalyse**
- **zustandsbezogener Test**
- Testen für bestimmte Abdeckungsmasse:
  - **Anweisungsüberdeckung**

- **Zweigüberdeckung**
- **Pfadüberdeckung**

Einführung in das **anforderungsbasierte Testen**.

Erklärung, dass systematische Testfallermittlungstechniken fast immer mit einem entsprechenden **Testendekriterium** (Ende der Testfallermittlung) verbunden sind.

6.2. Nichtfunktionale Testtechniken

Erklärung der Techniken für das Testen **nichtfunktionaler Qualitätsmerkmale** wie **Zuverlässigkeit** und **Benutzbarkeit** sowie **Effizienz**.

6.3. Dynamische Analyse

Erklärung, dass eine **dynamische Analyse** Ablaufinformationen über das zur Ausführung gebrachte Softwareprogramm liefert. Dies wird in der Regel durch eine Instrumentierung des zu testenden Programms erreicht. Die **dynamische Analyse** beobachtet den Verbrauch, die Nutzung und die Freigabe von Speicher, Memoryleaks, nicht zugewiesene Pointer, Pointerarithmetik sowie weitere Fehlerquellen, die statisch nur schwer zu untersuchen sind.

6.4. Statische Analyse

Erklärung, dass die **statische Analyse** ohne Ausführung der zu testenden Software auf einem Rechner erfolgt und mögliche Fehler wie unerreichbaren Code, nicht deklarierte Variablen, Typenunverträglichkeit von Parametern, nicht aufgerufene Funktionen und Prozeduren und mögliche Indexverletzungen bei Arrays aufdecken kann.

Erklärung des Grundprinzips der statischen Analyse: Ein Prüfobjekt, welches nach einem vorgegebenen Formalismus (Syntax) aufgebaut ist, wird gelesen und analysiert. Häufig werden mit statischer Analyse fehlerträchtige Situationen wie beispielsweise:

- **Verletzung der Syntax**
- **Konventions- und Standardabweichungen**
- **Kontrollflussanomalien**
- **Datenflussanomalien**

festgestellt oder Maßzahlen ermittelt. Beispielsweise wird die Anzahl der Programmzeilen im Quellcode über statische Analyse berechnet.

Erklärung, dass jeder Fehler, den ein Compiler aufdeckt, durch **statische Analyse** gefunden werden kann. Viele Compiler stellen auch Informationen über die Verwendung von Variablen bereit. Dies kann bei der Wartung eines Softwareprogramms nützlich sein.

Detaillierte Erläuterung der Konzepte, auf denen die Datenflussanalyse beruht. Erklärung, dass die Datenflussanalyse den Gebrauch von Daten auf Pfaden durch den Programmcode betrachtet und nach möglichen Anomalien sucht. Anomalien sind beispielsweise Definitionen von Variablen ohne deren Verwendung oder die Benutzung von Variablen nach deren Löschung.

Erklärung eines Kontrollflussgraphen und Herleitung des Graphen am Beispiel eines Softwareprogramms. Zur Notation des Kontrollflussgraphen siehe: [Hetzel85].

Erklärung der Anwendung von Komplexitätsmetriken. Berechnung der **Lines of Code** (LOC) und Berechnung der **zyklomatischen Komplexität**.

6.5. Unsystematische Testtechniken

Aufzeigen des Nutzens von unsystematischen Testtechniken wie undokumentiertes Ad-hoc-Testen (explorativ Testen), dokumentiertes intuitives Testen und schwachstellenorientiertes Testen als zusätzliche Techniken im Testprozess.

6.6. Auswahl von Testtechniken

Erklärung, wann welche Testtechnik zum Einsatz kommt.

Erklärung, dass die Wahl der Testtechnik aufgrund von Standards und/oder vertraglichen Anforderungen erfolgen kann. Beschreibung von typischen allgemeinen oder branchenspezifischen Standards, die bestimmte Testtechniken vorschreiben wie [IEC61508], [DO-178B], Standards für Signalanlagen im Eisenbahnverkehr, Standards der Nuklearindustrie, Standards in der pharmazeutischen Industrie, MISRA-Vorschriften für Software in Kraftfahrzeugen. Erläuterung des gegenwärtigen Wissens hinsichtlich der Wirksamkeit der unterschiedlichen Testtechniken.

7. Reviews 480 min

7.1. Einführung

Erläuterung des Nutzens von **Reviews**:

- das Finden von Fehlern in frühen Entwicklungsphasen
- die damit verbundene Kosteneffizienz

Erklärung der Vorteile, die mit Reviews in frühen Phasen des Softwareentwicklungsprozesses verbunden sind:

- Verhinderung der Fehlerausbreitung in weitere Entwicklungsphasen
- Sicherstellung von testbaren Anforderungen

Alle Arten von Dokumenten können einem Review unterzogen werden, z. B. Quellcode, Anforderungsspezifikationen, Konzepte, Testpläne, Testdokumente etc.

Alle Reviewarten haben das Ziel, Fehler aufzudecken.

Einführung in den Standard [IEEE1028] »Standard for Software Reviews«.

7.2. Prinzipien

Reviews werden am besten durchgeführt, sobald alle Quelldokumente (Dokumente, die die Anforderungen an das Produkt beschreiben) und die Standards, die das Produkt erfüllen soll, vorliegen. Fehlt eines der Quelldokumente oder der Standards, so können nur Fehler und Inkonsistenzen innerhalb eines Dokumentes gefunden werden, nicht aber dokumentübergreifend.

Ein Review kann zu drei möglichen Ergebnissen führen:

- das Dokument kann unverändert weiterverwendet werden
- das Dokument muss korrigiert werden, ein weiteres Review ist aber nicht notwendig
- das Dokument muss umfassend korrigiert werden, ein weiteres Review ist notwendig

Die Personen, die an einem Review teilnehmen, füllen normalerweise unterschiedliche Rollen aus:

- **Manager**: entscheidet über die Durchführung eines Reviews.
- **Moderator**: leitet als neutrale Person das Review. Er ist dabei, falls nötig, für die Vermittlung zwischen unterschiedlichen Standpunkten zuständig und oft die entscheidende Person, was den Erfolg des Reviews angeht.
- **Autor**: ist der Ersteller und/oder Hauptverantwortliche des Dokuments, das dem Review unterzogen wird.
- **Gutachter**: Fachexperten (auch Reviewer oder Inspektoren genannt), die nach der entsprechenden Vorbereitung an der Sitzung teilnehmen.
- **Protokollführer**: dokumentiert knapp und präzise alle während der Sitzung erkannten Abweichungen, Problemen, offene Punkte etc.

Die **Gutachter** führen das eigentliche Review durch. Einige **Gutachter** können auf bestimmte Aspekte spezialisiert sein, unter denen sie das Dokument betrachten. Dabei verwenden sie oft Checklisten.

Ein **Review** stellt eine Art **statische Prüfung** dar. Im Anschluss an ein Review auf Basis des Quellcodes erfolgt normalerweise eine dynamische Prüfung des Programms, um die Fehler zu finden, die durch statische Prüfungen nicht aufzudecken sind.

Es gibt unterschiedliche **Reviewarten**, die formal unterschiedlich gestaltet werden. Die Reviewarten sollen im Folgenden dargestellt und miteinander verglichen werden, um ihre Stärken, Schwächen und Einsatzmöglichkeiten aufzuzeigen. In den unterschiedlichen Reviewarten werden unterschiedliche sekundäre Ziele verfolgt.

7.3. Nichtformale Reviews

Erklärung, dass bei **nichtformalen Reviews** ein oder mehrere Gutachter (Autor ausgeschlossen) ein Dokument bearbeiten und ihre Anmerkungen dazu abgeben. Eine Reviewsitzung findet nur in den seltensten Fällen statt.

7.4. Walkthroughs

Die **Reviewdokumente** sollen eine angemessene Zeit vor der **Reviewsitzung** an die **Gutachter** verteilt werden, damit sich diese vorab mit dem Inhalt vertraut machen können. In einem **Walkthrough** führt der **Autor** durch das Dokument. Dabei werden oft Szenarios (Anwendungsfälle) und ihre Umsetzung in das technische Design durchgespielt. Weiterhin können Trockendurchläufe durch den Programmcode zum Einsatz kommen.

Mit einem Walkthrough werden folgende sekundäre Ziele verfolgt:

- Untersuchung von Alternativen und stilistischen Fragen
- Erweiterung des Know-hows der Gutachter

7.5. Technische Reviews

Expertenreview (Peer Review)

Erklärung, wie die **Reviewdokumente** an die **Gutachter** verteilt werden und dass sich die **Gutachter** durch die Bearbeitung der Dokumente vorab auf das **Reviewmeeting** vorbereiten müssen.

Der **Moderator** führt Schritt für Schritt durch das Dokument. Die **Gutachter** bringen an den entsprechenden Stellen die Anmerkungen ein, die sie bei ihrer Vorbereitung festgehalten haben. Die **Gutachter** diskutieren kontroverse Ansichten, treffen Entscheidungen und legen die nächsten Schritte fest.

In einem **Expertenreview** können mehr **Gutachter** involviert sein als in **Walkthroughs** oder **Inspektionen**.

7.6. Inspektion

Die im Folgenden dargestellte Methode der Inspektion beruht auf der von Michael Fagan (IBM) 1976 veröffentlichten Reviewtechnik (Fagan's inspections).

Der Prozess einer **Inspektion** ist formal definiert und wird strikt eingehalten. Das Ergebnis einer **Inspektion** gründet sich darauf, ob das Produkt vordefinierte Anforderungen erfüllt.

Ein **Vorbereitungsgespräch** kann benutzt werden, um die Teilnehmer mit dem **Reviewverfahren** vertraut zu machen, ihre Rollen zu erklären und die **Reviewdokumente** vorzustellen. Die Bereiche werden identifiziert, die eine tiefergehende Betrachtung durch einzelne **Gutachter** erfordern.

Die **Gutachter** führen die **Inspektion** auf die ihnen übertragenen Reviewdokumente für sich alleine durch, dokumentieren ihre Anmerkungen und senden diese vor Beginn des **Inspektionsmeetings** an den Moderator zurück.

Der **Leser** studiert ebenfalls die **Reviewdokumente** und stellt sicher, dass er in der Lage ist sie beim **Inspektionsmeeting** sinnvoll darzustellen.

Der **Moderator** muss qualifiziert sein, eine Inspektion durchführen zu können (z. B. durch eine entsprechende Schulung). Der **Moderator** ist für die Planung der **Inspektion** und Auswahl der **Gutachter** verantwortlich.

Der **Moderator** leitet jedes **Inspektionsmeeting** und stellt sicher, dass die aufgenommenen Anmerkungen entsprechend bearbeitet werden. Dem **Autor** ist es in einem **Inspektionsmeeting** nicht gestattet, die Rolle des Moderators zu übernehmen und sein Dokument vorzustellen. Während des **Inspektionsmeetings** präsentiert und interpretiert der Leser die Reviewdokumente.

Sollte es beim **Inspektionsmeeting** zu Differenzen zwischen den **Gutachtern** kommen, wird der Diskussionspunkt vermerkt und an Ende des **Inspektionsmeetings** diskutiert.

Der IEEE-Standard empfiehlt, dass der Moderator die von den Gutachtern benötigten Vorbereitungszeiten und die Anzahl der gefundenen Fehler statistisch erfasst. Die Aufwände für die Entdeckung der Fehler werden den geschätzten Kosten gegenübergestellt, die entstanden wären, wenn die Fehler nicht entdeckt worden wären. Die Gegenüberstellung zeigt die Aufwands- und Kostenersparnis, die durch den Reviewprozess erreicht wurde.

Für Details wird auf den IEEE-Standard verwiesen. Als Sekundärliteratur wird auf den Originalbeitrag von Michael Fagan [Fagan76] verwiesen.

7.7. Die Einführung von Reviews

Erklärung der Schritte, die für eine effektive Einführung von **Reviews** in Organisationen notwendig sind:

- Sicherstellung der Unterstützung durch das Management
- Durchführung von Pilotprojekten zu Reviews
- Aufzeigen des Nutzens von Reviews durch Einsparen von Kosten
- Schulung und Training von Reviewverfahren

7.8. Warum Reviews manchmal fehlschlagen

Erklärung von Ursachen, die häufig für das Ausbleiben des Erfolges von Reviews verantwortlich sind:

- Rollen und Verantwortlichkeiten werden nicht verstanden oder nicht gelebt
- nicht ausreichendes Training in den Reviewverfahren
- unzureichende Ressourcen (Zeit, Budget, Personal) zur Durchführung von Reviews
- keine Verbesserung des über allem stehenden Softwareentwicklungsprozesses
- Widerstand gegen Formalismen
- Standards und Abläufe zur Unterstützung von Reviews sind entweder nicht vorhanden oder nur in einer geringen Qualität verfügbar

Erläuterung, dass psychologische Faktoren zur Akzeptanz von Reviews eine bedeutende Rolle spielen:

- das Dokument steht auf dem Prüfstand, nicht der Autor
- die **Gutachter** unterstützen den Autor in seiner Arbeit, ein fehlerfreies Ergebnis zu liefern
- **Autoren** werden nicht bestraft oder bewertet, wenn Fehler gefunden werden
- Reviewgespräche beschränken sich auf das Finden von Fehlern und werden nicht dazu genutzt, den Autor zu kritisieren

8. Abweichungsmanagement 120 min

Erläuterung und Erklärung des **Abweichungsmanagements** nach [IEEE1044] (Standards Classification for Software Anomalies) und [IEEE1044.1] (Guide to Classification for Software Anomalies).

Aufzeigen der Prozessschritte zur Erkennung, der Überwachung und der Behebung von Abweichungen: Erkennung (Recognition), Analyse (Investigation), Bearbeitung (Action), Abschluß (Disposal).

9. Vorgehen zur Optimierung des Testprozesses 180 min

wie in [Koomen99] beschrieben

Erklärung, dass die **Prozessverbesserung** sowohl für den Software-Entwicklungsprozess als auch für den Testprozess Relevanz besitzt.

Überblick über die **Testprozessverbesserungsmodelle** des SEI Capability Maturity Model (CMMI) und [ISO15504] (SPICE). Erklärung des SEI CMMI-Modells und Gegenüberstellung zu SEI CMM und ISO/IEC 15504.

Erläuterung der **5 Reifegradstufen** in CMMI. Aufzeigen der wesentlichen Schritte zur Durchführung von Prozessverbesserungen.

Überblick über das SPICE-Referenz- und Assessment-Modell. Erläuterung der 6 Reifegradstufen in SPICE.

Gegenüberstellung CMM–CMMI–SPICE

Detaillierte Erläuterung des Testing Maturity Model (TMM, wie von IIT definiert) und Vorgehen zur Optimierung des Testprozesses (wie in [Koomen99] beschrieben).

Erläuterung der 5 Reifegradstufen in TMM.

Erläuterung der Key Areas und Levels in Vorgehen zur Optimierung des Testprozesses (wie in [Koomen99] beschrieben). Erläuterung der Test Maturity Matrix.

10. Testwerkzeuge 420 min

10.1. Überblick

Für viele Aktivitäten innerhalb des Testprozesses stehen Werkzeuge zu deren Unterstützung zur Verfügung. Erläuterung, dass Testwerkzeuge für sicherheitskritische Software entsprechend den erforderlichen Standards zertifiziert sein müssen. Die nachfolgenden Arten von Testwerkzeugen sollen beschrieben, die Vor- und Nachteile erklärt werden.

10.2. Begriffe

Erklärung der Unterschiede zwischen »intrusiven« (die Testumgebung beeinflussenden) und »nicht-intrusiven« Testwerkzeugen.

10.3. Werkzeugtypen

Werkzeuge zur Prüfung von Anforderungen unterstützen die Verifikation und Validierung von Anforderungsmodellen z. B. durch Konsistenzprüfungen und Animationen.

Werkzeuge zur statischen Analyse liefern Informationen über die Qualität des Programmcodes. Dies geschieht durch die Untersuchung des Codes und nicht durch die Durchführung von Testfällen. Solche Werkzeuge ermöglichen eine objektive Messung von verschiedenen Softwareeigenschaften, wie z. B. Komplexität und anderen Qualitätsmetriken.

Viele Werkzeuge zur statischen Analyse unterstützen die Einhaltung von **Programmierrichtlinien**. Verstösse werden als Warnung mit einem Hinweis auf die verletzten Regeln im Programmcode angezeigt. Das Werkzeug liefert einen Standardsatz von Regeln für eine bestimmte Programmiersprache, die mit eigenen Änderungen und Ergänzungen angepasst werden können, um spezifische Projektbedürfnisse zu erfüllen.

Werkzeuge zum Entwurf von Testeingaben generieren aus einer Spezifikation heraus die entsprechenden Eingaben. Die Spezifikationen liegen dabei in CASE-Werkzeugen in Form von formal spezifizierten Anforderungen vor.

Es existieren auch Werkzeuge, die Testeingaben aus der Analyse des Programmcodes generieren können. Diese Werkzeuge können die Testfälle auch automatisch ausführen, um z. B. die Fehlerbehandlung unerlaubter Dateneingaben prüfen zu können.

Testroboter (auch »**Capture-Replay**« oder **Mitschnitt-Werkzeuge** genannt) zeichnen die Eingaben des Benutzers und die Antworten des Systems am Bildschirm auf, damit diese für den späteren Vergleich verwendet werden können. Für Anwendungen mit grafischen Oberflächen (GUI) können die Werkzeuge Mausbewegungen und Tastendrucke simulieren und grafische Objekte, wie Fenster, Dialoge, Felder, Tasten und andere Kontrollelemente, erkennen. Der Zustand eines grafischen Objekts kann zum späteren Vergleich aufgenommen werden.

Sequenzen werden normalerweise als programmierbares Skript aufgenommen, das meistens manuell modifiziert wird, um Verifikationsschritte einzuarbeiten und die Wartbarkeit des Skripts zu erhöhen. Eine der wichtigsten Ziele hierbei ist oft die Trennung zwischen Testskripten und

Testdaten. Bei der Ausführung des Skripts werden die Benutzereingaben wiederholt und die Antworten des Systems mit den im Skript aufgenommenen Werten verglichen. Unterschiede zwischen den erwarteten und den tatsächlichen Antworten werden vom Werkzeug identifiziert und manchmal automatisch in ein Fehlermanagementwerkzeug aufgenommen (siehe Testmanagementwerkzeuge).

Testfälle, Testdaten, Skripte und erwartete Ergebnisse können in getrennten Testablagen gehalten werden. **Testroboter** werden meistens zur Automatisierung von **Regressionstests** verwendet.

Testrahmen und **Treiber** werden für die Ausführung von Software verwendet, die keine Benutzerschnittstelle aufweisen oder bei denen die Benutzerschnittstelle noch nicht vorliegt. Es stehen einige kommerzielle Produkte zur Verfügung, aber meist müssen **Testrahmen** und **Treiber** für ein bestimmtes Projekt individuell entwickelt werden.

Testskriptgeneratoren erzeugen Skripte anhand von Spezifikationen oder direkt auf Basis des Quellcodes.

Simulatoren werden für die Unterstützung von Tests eingesetzt, wenn der Programmcode entweder nicht zur Verfügung steht oder praktisch nicht anwendbar ist (z. B. beim Testen von Software in sicherheitskritischen Bereichen von Nuklearanlagen).

Performance-Testwerkzeuge haben zwei Hauptfunktionen: Die Erzeugung von **Last** und das Messen von Testtransaktionen.

Lastgeneratoren können mehrere Benutzer und hohe Datenvolumen simulieren. Die Last wird meistens durch **Treiber** erzeugt, die die Last von mehreren gleichzeitig arbeitenden Benutzern nachahmen. Verschiedene Messungen bezüglich der Transaktionen werden dabei zur Analyse von Antwortzeiten aufgezeichnet. Normalerweise liefern Performance-Testwerkzeuge verschiedene Berichte und graphische Darstellungen von Antwortzeit im Verhältnis zur Last.

Werkzeuge zur Analyse von Web-Sites werden für die Sammlung von Informationen über Benutzeranzahl und -verhalten verwendet.

Werkzeuge zur **dynamischen Analyse** liefern Informationen über die Software während der Ausführung. Solche Werkzeuge werden am häufigsten verwendet, um folgende mögliche Fehlerquellen zu identifizieren: Zeiger ohne Zuweisung, falsche Zeigerarithmetik, inkorrekte Zuordnung, Verwendung und Freigabe von Hauptspeicher zum Finden von Speicherlöchern und anderen Fehlern, die durch **statische Analyse** nicht zu finden sind.

Mit **Vergleichswerkzeugen** können Unterschiede zwischen erwarteten und tatsächlichen Ergebnissen aufgezeigt werden. Vergleichswerkzeuge können normalerweise mit einer Reihe unterschiedlicher Datei- und Datenbankformate umgehen. Sie sind oft Bestandteil von Testrobotern und können mit zeichenorientierten Oberflächen und/oder grafischen Objekten (GUI-Objekte, Bitmaps) umgehen. Diese Werkzeuge bieten formatabhängige Möglichkeiten zum Filtern bestimmter Informationen und zum Ausblenden von Datenzeilen bzw. Spalten.

Testmanagementwerkzeuge bieten oft vielfältige Funktionalitäten an. Sie unterstützen die Erstellung, Verwaltung und Kontrolle von Dokumenten wie z. B. Testplänen, Spezifikationen und Ergebnissen. Manche Werkzeuge unterstützen die Projektmanagementaspekte des Testens, z. B. die zeitliche Planung der Tests, das Speichern von Ergebnissen und das Aufnehmen von Problemen, die während der Testdurchführung entdeckt werden.

Abweichungsmanagementwerkzeuge (auch Fehlermanagementwerkzeuge genannt) können Aktivitäten wie Abweichungsaufnahme, -verfolgung, -kontrolle, -behebung und Testwiederholung unterstützen. Werkzeuge der Kategorie »Testmanagement« können Testfälle und deren Testabdeckung aufnehmen und mit anderen Informationen und Objekten verknüpfen. Die meisten **Testmanagementwerkzeuge** bieten vielfältige Möglichkeiten zur Berichterstellung und Analyse an.

Werkzeuge zur Messung und **Analyse der Testabdeckung** messen den **Abdeckungsgrad** der inneren Struktur des Programms bei seiner Ausführung im Test. Die zu testenden Programme werden vor der **Testdurchführung** vom Testwerkzeug instrumentiert. Die **Testabdeckung** wird über im Programm eingefügte Zählanweisungen ermittelt. Das Testwerkzeug protokolliert alle Detailinformationen und speichert sie in einer Datei. Diese Aktivitäten verlangsamen das zu testende Programm und können auch die Funktionalität beeinflussen. Nach der Testdurchführung werden gespeicherten Daten vom Testwerkzeug analysiert und Statistiken über die **Testabdeckung** erstellt. Die meisten dieser Testwerkzeuge liefern die gängigsten Testabdeckungsstatistiken (z. B. **Anweisungs- und Zweigüberdeckung**).

Hyperlink- oder **Link-Testwerkzeuge** prüfen, ob alle Links einer Web-Site noch vorhanden sind und geben nicht mehr vorhandene Links aus.

Überwachungswerkzeuge werden meistens für e-Commerce- und e-Business-Anwendungen eingesetzt und überprüfen, ob eine Web-Site die angeforderte Benutzerverfügbarkeit und Performance liefert. Die Werkzeuge laufen permanent im Hintergrund und geben eine Warnung aus, sobald eine Website oder ein Programm nicht mehr zur Verfügung steht oder Performance-Grenzwerte überschritten werden.

Werkzeuge zur Prüfung von Sicherheitsaspekten werden meistens sowohl für e-Commerce- und e-Business-Anwendungen als auch für Web-Sites eingesetzt. Ein solches Testwerkzeug prüft alle Aspekte eines webbasiserten Softwaresystems, die der Gefahr eines Missbrauchs durch nichtautorisierte Benutzung unterliegen.

Test-»**Orakel**« liefern erwartete Sollergebnisse. Da solche Werkzeuge teilweise die gleiche Funktionalität wie die zu testende Software nachbilden müssen, sind sie selten verfügbar. Ein **Testorakel** ist zum Beispiel ein Altsystem, das durch ein neues System mit der gleichen Funktionalität ersetzt wird. Das Altsystem liefert für alle Testfälle die richtigen Sollergebnisse. **Orakel** können auch für Systeme, die eine hohe Performanceleistung erbringen müssen, eingesetzt werden. Ein **Orakel** mit niedrigen Performanceeigenschaften kann verwendet bzw. erstellt werden, um die funktionalen Ergebnisse, die ein hochperformantes Softwaresystem liefern soll, zu generieren.

10.4. Werkzeugauswahl

Erklärung der Notwendigkeit einer formalen Werkzeugevaluierung und eines gründlichen Auswahlprozesses. Mit diesen Aktivitäten werden zwei Ziele verfolgt:

- Vermeidung eines unnötigen Kaufs eines Testwerkzeuges
- Sicherstellung, dass das gekaufte Werkzeug auch tatsächlich genutzt wird

Beschreibung, dass der Auswahlprozess und die **Werkzeugevaluierung** idealerweise aus folgenden Schritten besteht:

- Identifikation und Quantifizierung des Problemumfeldes
- Betrachtung möglicher Alternativlösungen
- Kosten-Nutzen-Analyse
- Identifikation von Lizensierungsmodellen und -alternativen
- Identifikation von Kosten, die mit dem Besitz des Werkzeugs verbunden sind
- Identifikation von Einschränkungen
- Identifikation der gewünschten Funktionalität und der Eigenschaften des Werkzeugs
- Identifikation von Werkzeugen für die detaillierte Evaluierung
- Durchführung der detaillierten Evaluierung
- Bei Bedarf Durchführung eines parallelen Probelaufes oder auch Probebetriebes von konkurrierenden Testwerkzeugen.

Erklärung, dass ein Evaluierungs- und Auswahlteam normalerweise aus einem Projektleiter (Vollzeit) und einigen Teammitarbeitern (temporär) besteht. Die Teammitarbeiter kommen aus unterschiedlichen Bereichen des Unternehmens, in denen das Werkzeug zum Einsatz kommen könnte.

10.5. Werkzeugimplementierung

Erklärung, dass ein formaler Implememtierungsprozess notwendig ist, um den langfristigen Erfolg des Werkzeugeinsatzes zu sichern.

Beschreibung des Implementierungsprozesses: Er beginnt mit einem kleinen Pilotprojekt von drei bis sechs Monaten, um eine Vorgehensweise für die Benutzung des Werkzeugs innerhalb der Organisation zu etablieren. Diese Vorgehensweise schließt je nach Bedarf die Architektur der Testware, die Skripterstellungsmethoden, Namenskonventionen und das Konfigurationsmanagement von Testware ein. Idealerweise arbeitet das Implementierungsteam Vollzeit im Pilotprojekt und übt dabei folgende Rollen aus:

Der **Initiator**:

- bringt die Implementierung des Testwerkzeugs voran
- versteht die verschiedenen Probleme und Aspekte, die entstehen könnten
- ist von den potentiellen Vorteilen des Werkzeugs überzeugt, strahlt Enthusiasmus aus und arbeitet gern und konstruktiv mit Kollegen

Der **Change-Manager**:

- plant, organisiert und betreibt die Werkzeugimplementierung (einschließlich des Pilotprojektes)
- kann auch die Rolle des Initiators einnehmen

Der **Werkzeugverantwortliche**:

- ist für die technische Unterstützung zuständig
- liefert intern Hilfe und Beratung bezüglich der Werkzeugverwendung

Erklärung, dass zusätzlich zu diesen Rollen und der Arbeit weiterer Teammitglieder ein Management-Sponsor notwendig ist, der das Implementierungsvorhaben sichtbar unterstützt.

Beschreibung, dass am Ende des Pilotprojektes die Ergebnisse mit den im Vorfeld definierten Kosten-/Nutzen-Aspekten verglichen werden. Bei einer positiven Beurteilung werden das Testwerkzeug und das im Pilotprojekt entwickelte Verfahren sukzessiv für andere Projekte bereitgestellt und verwendet.

11. Teamzusammensetzung 180 min

11.1. Individuelle Fähigkeiten

Die Befähigungen zum Testen von Software können durch Erfahrungen und/oder Schulungen in verschiedenen Arbeitsbereichen erlangt werden:

- Durch die Anwendung von Softwaresystemen
- Durch Tätigkeiten in der Softwareentwicklung
- Durch Tätigkeiten im Softwaretest

Die **Anwender** von Softwaresystemen kennen sich gut mit der Benutzerseite des Systems aus und haben gute Kenntnisse darüber, wie das System angewandt wird, an welcher Stelle Fehler schwerwiegende Auswirkungen haben würden und wie das normale Systemverhalten sein sollte. Auch Anwender mit weniger umfasssenden Systemkenntnissen können nützliche Hinweise zum Testen beisteuern.

Erklärung der Problemstellung bei der Mitarbeiterauswahl, wie z. B. der Sicherstellung, dass neu hinzukommende Mitarbeiter die bereits vorhandenen Kenntnisse und Persönlichkeitsprofile eines Testteams nach Möglichkeit ergänzen sollten. Aufzeigen der Vorteile, eine Vielzahl unterschiedlicher Persönlichkeitstypen in einem Testteam zu haben.

Darstellung des Konzepts der Rollenverteilung zur Erklärung des indivduellen Verhaltens von Teammitgliedern, deren Beisteuerung zum Teamergebnis und der Beziehungen innerhalb eines Teams.

Als Grundlage kann das Teamrollenkonzept von Dr. Meredith Belbin, das Team-Management-Rad von Margison-McCann oder auch das Gruppenkonzept Themenzentrierte Interaktion von Ruth C. Cohn verwendet werden ([URL:bergander], [URL:tms], [URL:tzi]).

11.2. Organisationsstruktur

Tester und andere Projektbeteiligte

Erklärung, dass Testaufgaben in unterschiedlichen Unternehmen in unterschiedlichen Strukturen organisiert sein können. So kann z. B. die Verantwortung für das Testen bei den Entwicklern selbst, bei einem Entwicklerteam oder bei einer speziellen Person des Entwicklerteams liegen. Die Verantwortung für das Testen kann aber auch einem gesonderten **Testteam** (welches mit keinerlei Entwicklungsaufgaben betraut ist), firmeninternen Testspezialisten (die mehrere Projekte beratend unterstützen können) oder externen Organisationen übertragen sein.

Beschreibung der Kommunikationsformen zwischen:

Testern und Entwicklern: **Tester** finden Fehler in der Software und müssen sie in diplomatischer Art und Weise weitervermitteln, um die Qualität des Produktes zu verbessern. Entwickler sollten die Tester darüber informieren, welche Systemteile eine erhöhte Aufmerksamkeit erfordern, weil sie etwa eine hohe Komplexität aufweisen oder neu konzipiert wurden.

Testern und Projektmanagement: **Tester** berichten dem **Projektmanagement** über den Testfortschritt und die Softwarequalität. Das Projektmanagement informiert die Tester über alle Änderungen zum Testumfang, Umgebungen, Projektplan und Lieferungsterminen.

Testern und Anwendern: **Tester** erhalten Informationen über Aspekte des Systems, die dem **Anwender** besonders wichtig sind und können somit Prioritäten für das Testen setzen. Tester erhalten Hintergrundinformationen über den Einsatzbereich des Systems. Anwender erhalten Testergebnisse, die von den Testern, falls nötig, näher erläutert werden.

11.3. Motivierung

Behandlung von motivierenden Faktoren wie Anerkennung und Bestätigung. Behandlung von demotivierenden Faktoren wie beispielsweise fehlende Aufstiegsmöglichkeiten.

Anerkennung und Respekt werden erworben, indem der **Tester** deutlich erkennbar zum Wertzuwachs des Projektes beiträgt. In einem einzelnen Projekt wird dieses am schnellsten dadurch erreicht, dass der **Tester** frühzeitig in die Reviewprozesse eingebunden ist. Im Laufe der Zeit werden Tester Anerkennung und Ansehen daraus gewinnen, dass sie ihren Anteil an der positiven Abwicklung der Projekte, in denen sie tätig waren, dokumentiert haben und transparent machen können.

Teil III.
Anhang und Literatur

Literatur

Vertiefende Literatur zu diesem Lehrplan und seinen internationalen »Geschwistern«

[Spillner02] Spillner, A.; Linz, T.: Basiswissen Softwaretest, Aus- und Weiterbildung zum Certified-Tester, 1. Aufl., Heidelberg, dpunkt-Verlag, 2003, ISBN 3-89864-178-3.
Stichworte: Testen, Testprozess, Testmanagement, Testtechniken, Abweichungsmanagement, Reviews

[Veenendaal02] Veenendaal, Erik van: The Testing Practioner; UTN Publishers, 2002.
Stichworte: Testprozess, Risikoorientiertes Testen, Testtechniken, Abweichungsmanagement, Reviews, Vorgehen zur Optimierung des Testprozesses wie in [Koomen99] beschrieben

Literatur zur Vertiefung Softwaretest allgemein

[Balzert00] Balzert, H.: Lehrbuch der Software-Technik; Band 1 und 2; Spektrum Akademischer Verlag, 1998.
Stichworte: Software-Entwicklung und Software-Qualität allgemein

[Beizer99] Beizer, B.: Software Testing Techniques, Van Nostrand Reinhold, 1999.
Stichworte: Testtechniken

[Dustin99] Dustin, E.; Rashaka, J.; Paul, J.: Automated Software Testing, Introduction, Management and Performance, Addison Wesley, 1999
Stichworte: Testwerkzeuge

[Fagan76] Fagan, M.E.: »Design and Code Inspections to Reduce Errors in Program Development«, in »IBM Systems Journal«, Vol. 15, No. 3, 1976, S. 182–211.
Stichworte: Reviews

[Fewster99] Fewster, M.; Graham, D.: Software Test Automation; Effective use of test execution tools, Addison Wesley, 1999
Stichworte: Testwerkzeuge

[Grochtmann93] Grochtmann, M. and Grimm, K., Classification Trees for Partition Testing, Software Testing, Verification and Reliability, 3:63-82, 1993
Stichworte: Testtechniken

[Hetzel85] Hetzel, W.C.: The Complete Guide to Software Testing, Collings

[Myers82] Myers, G.J.: Methodisches Testen von Programmen, Oldenbourg, München, Wien 1982. (7. Auflage 2001).

[Koomen99] Koomen, T., Pol, M.: »Vorgehen zur Optimierung des Testprozesses«, 1999, ACM Press, ISBN 0-201-59624-5.
Stichworte: Vorgehen zur Optimierung des Testprozesses

[URL:bergander] `http://www.bergander.de/`
Stichworte: Teamzusammensetzung, Teamrollenkonzept

[URL:tms] `http://www.tms-zentrum.de/`
Stichworte: Teamzusammensetzung, Team-Management-Rad

[URL:tzi] `http://www.tzi-forum.com/`
Stichworte: Teamzusammensetzung, Themenzentrierte Interaktion

Normen und Standards

[URL:sigist] `http://www.testingstandards.co.uk/`; The BCS SIGIST Standards Working Party;

[BS7925-2] British Standard 7925-2, Software testing, Part 2; Software component, 1998

[DIN66272] DIN 66272, Ausgabe:1994-10, Informationstechnik – Bewerten von Softwareprodukten – Qualitätsmerkmale und Leitfaden zu ihrer Verwendung; Identisch mit ISO/IEC 9126:1991, Originalsprache: Deutsch.

[DO-178B] Software Considerations in Airborne Systems and Equipment Certification, RTCA, 1992.

[IEEE829] IEEE Std. 829-1998, IEEE Standard for Software Test Documentation (Rev. of IEEE Std. 829-1983).

[IEEE1028] IEEE Std 1028-1997, IEEE Standard for Software Reviews.

[IEEE1044] IEEE Std 1044-1993, IEEE Standard Classification for Software Anomalies.

[IEEE1044.1] IEEE Std 1044.1-1995, IEEE Guide to Classification for Software Anomalies.

[IEC61508] IEC 61508:1998, Functional safety of electrical / electronic / programmable electronic safety related systems, Industrial Electrical Commitee.

[ISO9126] ISO/IEC 9126:1991, Bewerten von Softwareprodukten, Qualitätsmerkmale und Leitfaden zu ihrer Verwendung (identisch mit [DIN66272]).

[ISO15504] ISO/IEC 15504 (1998): »Information technology – Software process assessment«, International Organization for Standardization.