Begriffe, Zusammenfassung, Fragen & Antworten: Softwaretest

Begriffe

- **Fehler**: Nichterfüllung festgelegter Anforderung
  → Abweichung zwischen Ist- und Soll-Verhalten
  - Fehlerwirkung: Nichterfüllung einer Anforderung, Ursache ist Fehlerzustand, Ausgelöst durch Fehlerhandlung
  - Fehlerzustand: inkorrektes Teilprogramm, Zustand eines Produkts der Funktion beeinträchtigt
  - Debugging: Ursache der beim Testen gefundenen Fehlerwirkung – also den Fehlerzustand – finden
  - statische Tests ermöglichen Fehlerzustand direkt zu finden
  - Fehlermaskierung: Fehlerzustand verhindert Aufdeckung eines anderen Fehlers
  - Fehlerhandlung: menschliche Handlung, die zu Fehlerzustand in Software führt
- **Mangel**: gestellte Anforderung oder berechtigte Erwartung nicht angemessen erfüllt
- **Testen**: Bewertung einer Software
  - ermitteln, ob festgelegte Anforderungen und Zweck erfüllt sind
  - finden von Fehlerzuständen
  - **Validierung**: Prüfen ob Entwicklungsergebnis individuellen Anforderungen bezüglich einer speziellen beabsichtigten Nutzung erfüllt.
    - Prüft gegen Verwendungszweck
    - Haben wir das richtige System realisiert?
  - **Verifizierung**: Prüfung, Ob Ergebnisse einer Entwicklungsphase Vorgaben der Phaseneingangs-Dokumente erfüllen.
    - Prüft gegen festgelegte Anforderungen
    - Haben wir das System richtig realisiert?
  - **Regressionstest**: erneuter Test nach Modifikation der SW
    → durch Modifikation keine weiteren Fehlerzustände (evtl. bisher maskierte)
  - **Fehlernachtest**: testet, ob konkreter Fehlerzustand nach Änderung beseitigt
- **Testfall**
  - besteht aus: Eingabewert, Soll-Ergebnis, Vorbedingungen, Nachbedingungen
  - zeigt: Ist-Verhalten
- **Testorakel**: Quelle für Soll-Werte
- **Testbasis**: Informationsquelle für Testfälle

- **Platzhalter**: Hilfsfunktion, die von Test aufgerufen wird
- **Treiber**: ruft zu testende Funktion auf (statt z. B. über Menü)
- **Testkonzept**: großer Plan, den Testmanager am Anfang schreibt
  - Gültigkeitsbereich, Umfang, Vorgehensweise, Ressourcen, Zeitplanung, Aktivitäten, Testobjekte, Testaufgaben
  - Mastertestkonzept: bezieht sich (im Gegensatz zum Stufentestkonzept) auf mehrere Teststufen
- **Testplan**: zeitliche Planung der Testdurchführung (wer testet was und wann)
- **Testumgebung**: Hardware, Instrumentierung, Simlulatoren, Tools, …
- **Softwarewartung**: Anpassung des Produkts an geänderte Einsatzbedingungen, Fixen von Bugs (die schon immer im System waren)

Unterscheidung der Software-Qualität nach ISO

Äußere und innere Merkmale:

- Funktional: **Funktionalität** (Interoperabilität, Sicherheit, …)
- Nicht-funktional
  - **Zuverlässigkeit** (Fehlertoleranz, …): Leistungsniveau unter festgelegten Bedingungen über festgelegten Zeitraum bewahren
  - **Benutzbarkeit**
  - **Effizienz**
  - **Änderbarkeit**/Wartbarkeit (inneres Merkmal)
  - **Portierbarkeit**/Übertragbarkeit (inneres Merkmal)

V-Modell

1. Anforderungsdefinition – Abnahmetest
2. funktionaler Systementwurf – Systemtest
3. technischer Systementwurf – Integrationstest
4. Komponentenspezifikaiton – Komponententest
5. Programmierung – Änderung

gleichzeitig zu 1. bis 4. laufen Testvorbereitungen

Grundsätze des Softwaretests

- Testen zeigt Anwesenheit von Fehlerzuständen
- Vollständiges Testen ist nicht möglich
- Mit dem Testen frühzeitig beginnen
- Häufung von Fehlern (Fehler nicht gleichmäßig über SW verteilt)
- Wiederholungen haben keine Wirksamkeit

- Testen ist abhängig vom Umfeld
- Trugschluss: keine Fehler heißt, dass das System brauchbar ist

Testprozess

1. Testplanung und -steuerung
2. Testanalyse und Testentwurf
3. Testrealisierung und Testdurchführung
4. Bewertung von Endekriterien und Bericht
5. Abschluss der Testaktivitäten

Teststufen

1. **Komponententest**: testet einzelne SW-Bausteine isoliert voneinander
   - zur Erstellung Entwicklerwissen nötig
2. **Integrationstest**: testet Zusammenspiel der SW-Bausteine
   - Top-Down: mit UI anfangen, benötigt Platzhalter
   - Buttom-Up: mit einzelnen Komponenten anfangen, benötigt Treiber
   - Ad-hoc: in Reihenfolge der Fertigstellung der Komponenten
   - Big-Bang: warten, bis alles fertig → Zeitverlust, aber keine Platzhalter/Treiber
3. **Systemtest**: testet Gesamtsystem aus Kundensicht
4. **Abnahmetest**: testet möglichst nah an Produktivumgebung explizit gegen Abnahmekriterien
   - Urteil des Kunden im Vordergrund
   - zielt nicht auf Finden von Fehlerzuständen ab
   - Alpha-Tests beim Hersteller, Beta-Tests/Feldtests beim Kunden
   - Unterscheidung: vertraglich vs. regulatorisch

Vergleich

Kriterium	Komponententest	Integrationstest	Systemtest	Abnahmetest
Testziele	Fehlerzustände in SW-Bausteinen	Fehlerzustände in SW-Schnittstellen und Zusammenspiel	prüfen, ob Anforderungen erfüllt	Vertrauen in System gewinnen
Testbasis	Komponentenspez., Entwurf, Datenmodell, Programmcode	SW-Entwurf, Architektur, Nutzungsabläufe, Use Cases	Anforderungsspez., Use Cases, funktionale Spez., Geschäftsprozesse, Risikoanalyse	Benutzeranforderungen, Systemanforderungen, Use Cases, Geschäftsprozesse, Risikoanalyse

Typische Testobjekte	Klasse/Unit/Modul	zu integrierende Einzelbausteine	Handbücher, Systemkonfiguration	Geschäftsprozesse des integrierten Systems, Anwenderverfahren, Konfigdaten, Berichte
Testwerkzeuge	Entwicklungsumgebung, Debugger, stat. Analysewerkzeuge	Testmonitore	Testmanagementwerkzeuge, GUI-Automatisierungswerkzeuge	
Testumgebung	Platzhalter, Treiber, Simulatoren		möglichst nah an Produktivumgebung	

Testarten

- **funktionaler Test**
  - beschreibt, *was* System leisten soll
  - prüft von außen sichtbares Verhalten
  - Strategien für funktionale Systemtests
    - anforderungsbasiert: Anforderungsdokument als Testbasis, zu jeder Anforderung mindestens ein Testfall
    - geschäftsprozessorientiert: statt einzelner Systemfunktionen sind Geschäftsprozesse, also hintereinander ablaufende Tests, im Fokus
    - anwendungsfallbasiert: Systemfälle aus Use-Cases ableiten
- **nicht-funktionaler Test**
  - prüft, *wie gut* System arbeitet
  - Performance, Last, Benutzbarkeit, Wartbarkeit, …
- **strukturbasierter Test**
  - basiert auf interner Struktur: Kontrollfluss, Aufrufhierarchie, Struktur abstrakter Modelle (Zustandsautomat)
- **änderungsorientierter Test** / Wartungstest
  - testen im Zusammenhang mit Änderungen
  - Gründe: Modifikation, Migration, Außerbetriebnahme (Datenarchivierung oder -übertragung in neues System testen)
  - Fehlernachtest, Regressionstest

Reviewprozess

Umfasst:

1. **Planung**: Dokumente auswählen, Eingangs- und Endekriterien, Prüfkriterien

festlegen

2. **Kick-off**: Versorgung der beteiligten Personen mit benötigten Infos
3. **individuelle Vorbereitung** der beteiligen Personen
4. **Reviewsitzung**: Beurteilung des Prüfobjekts
5. **Überarbeitung**: aufgedeckte Fehlerzustände beheben
6. **Nachbereitung**: Kontrolle der Überarbeitung

Rollen

- **Manager**: entscheidet über Durchführung, prüft ob Ziele erreicht
- **Moderator**: führt durch Prozess, vermittelt zwischen verschiedenen Standpunkten
- **Autor**
- **Gutachter**: haben technisches/fachliches Hintergrundwissen, identifizieren Fehlerzustände
- **Protokollant**: Ergebnisse, Probleme, offene Punkte

Arten

- informell vs. formal
- Produkt-Reviews: beziehen sich auf Produkte, die während Entwicklungsprozess erstellt werden: Informelles Review, Walkthrough, Technisches Review, Inspektion
    - **Informelles Review**: informelle Prüfung innerhalb einer Entwicklungsaktivität („Gegenlesen“)
    - **Walkthrough**: schrittweise Präsentation durch Autor
      → Autor kann Sitzung stark beeinflussen, evtl. nachteilig
      gut wenn primäres Ziel Bekanntmachung (nicht unbedingt Fehler zu finden)
    - **Technisches Review**: dokumentierter, definierter Fehlerfindungsprozess, gleichgestellte Mitarbeiter
        - Ziele: Übereinstimmung des zu prüfenden Dokuments mit Spezifikation, Diskussion, Entscheidung treffen
        - hoher Aufwand in Vorbereitungsphase
        - *nicht* über Konsequenzen des Ergebnisses entscheiden
    - **Inspektion**: sehr formaler Ablauf
        - Hauptzweck: Fehlerzustände finden
        - Sitzung durch geschulten Moderator (nie Autor)
        - auch Bewertung des Entwicklungs- und Inspektionsprozesses

Vor- und Nachteile

- Vorteile
    - früh einsetzbar

    - geringere Zahl an Fehlerzuständen
      → Kostenreduzierung während Produkt-Lebenszeit
    - oft kürzere Entwicklungszeiten
    - verringern Aufwand für dynamisches Testen
    - Wissensaustausch im Team, gegenseitiges Lernen
    - klare und verständliche Darstellung der Sachverhalte notwendig
    - Verantwortung trägt Team
- Fallstricke
    - benötigtes Personal nicht ausreichend
    - fehlende Zeit
    - fehlende Vorbereitung
    - unklare Ziele
    - unzureichende Dokumentation
    - fehlende Unterstützung durch Management

Statische Analyse

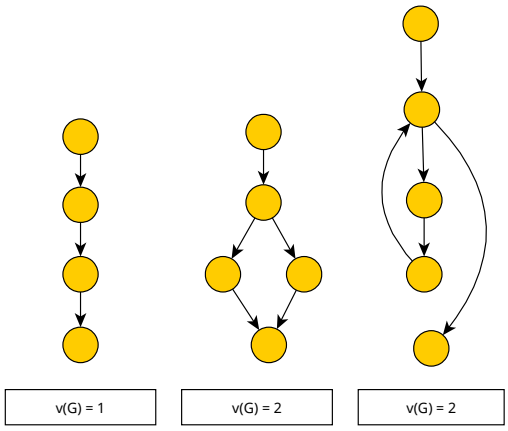
- Ziel: Aufdeckung vorhandener Fehlerzustände *ohne Ausführung* der Prüfobjekte
- nur mit Werkzeugunterstützung sinnvoll
- Dokumente benötigen Formalismus

Software-Metriken

- messen Merkmale von SW-Produkten und -Prozessen
- zur Bewertung, Planung, Überwachung
- **statische Maße**: messen Eigenschaft zu bestimmten Zeitpunkt
- **dynamische Maße**: messen Entwicklung einer Eigenschaft im Laufe der Zeit
- **Zyklomatische Zahl**: Anzahl der Verzweigungen
    - Berechnung: v(G) = Anzahl der Kanten – Anzahl der Knoten + 2
    - nur für einzelne Prozedur interessant, max. 10

Nutzen statischer Analysen

- frühe Erkennung von Fehlerzuständen
- frühe Warnung vor verdächtigen Aspekten in Code/Design
- verbesserte Lesbarkeit, Änderbarkeit und Wartbarkeit



Kontroll- und Datenflussanalyse

- sucht nach Anomalien im Programmtext

- Datenflussanomalien
  - **ur**-Anomalie: undefinierter Wert (u) wird gelesen (r)
  - **du**-Anomalie: definierter Wert (d) wird ohne verwendet zu werden ungültig (u)
  - **dd**-Anomalie: definierter Wert (d) wird ohne verwendet zu werden überschrieben

Testentwurfsverfahren

Vorgehensweise, nach der Testfälle abgeleitet werden:

- **Black-Box-Testentwurf**: Spezifikationen sind Grundlage für Testentwurf, aber nicht Programmtext oder innerer Aufbau, z. B. Äquivalenzklassenbildung, Grenzwertanalyse
  - Synonym: *spezifikationsorientierte* Testentwurfsverfahren
- **Whie-Box-Testentwurf**: basierend auf interner Struktur; Programmquelltext liegt vor und ist Grundlage, z. B. Entscheidungsüberdeckung, Anweisungsüberdeckung
  - Synonym: *strukturorientierte* Testentwurfsverfahren
  - Eingriff in Testobjekt möglich (intrusive Tests)
- **Erfahrungsbasierte** Testentwurfsverfahren

  nutzen Wissen und Erfahrung von Testern, Entwicklern, Anwendern, …. über wahrscheinliche Fehler

Blackbox-Testentwurfsverfahren

- **Äquivalenzklassenbildung**: repräsentative Eingaben, Erreichen gültiger Ausgaben
  - zerlegt Definitionsbereiche der Ein- und Ausgaben in Äquivalenzklassen
  - Werte in Klasse sollen gleiches Verhalten ergeben
  - sinnvolle Stichprobe: nur Wahl eines Testwertes pro Klasse
  - Anzahl der Testfälle minimieren: Heuristiken
    - Testfälle aus allen Repräsentanten kombinieren und anschließend nach Häufigkeit sortieren
    - Minimalkriterium: mindestens ein Repräsentant jeder Äquivalenzklasse pro Testfall
    - Repräsentanten ungültiger Äquivalenzklassen nicht mit anderen ungültigen kombinieren
  - Vorteile: Anzahl der Testfälle kleiner als bei unsystematischer Fehlersuche, geeignet für Programme mit vielen Ein- und Ausgabebedingungen
  - Nachteile: betrachtet Bedingungen für einzelne Ein- oder Ausgabeparameter, Beachtung von Wechselwirkungen der Abhängigkeiten zwischen Bedingungen aufwändig
- **Grenzwertanalyse**: prüft Wertbereiche, Wertgrenzen

    - Idee: Grenzbereiche von Bedingung testen, um „one off"-Fehler finden
    - beste Erfolge in Kombination mit anderen Verfahren
        - mit Äquivalenzklassenbildung: Grenzen der Äquivalenzklasse testen, jede Rand muss in einer Testkombination vorkommen
    - Vorteile
        - effiziente Kombination mit anderen Verfahren, die Freiheitsgrade in Wahl der Testdaten lassen
        - an Grenzen häufiger Fehler
    - Nachteile
        - Rezepte für Auswahl von Testdaten schwierig anzugeben
        - Bestimmung aller relevanten Grenze schwierig
- **zustandsbasierter Test**: prüft gültige und ungültige Zustandsübergänge
    - oft hat auch bisheriger Ablauf Einfluss
      → Testobjekt kann beginnend vom Startzustand unterschiedliche Zustände annehmen
    - Modellierung mittels **Zustandsautomat**: Berechnungsmodell aus endlicher Anzahl von Zuständen und Übergängen
    - gut wo Zustände Rolle spielen und Funktionalität von Zustand beeinflusst wird
    - gut für OO
- **Entscheidungstabellentest**: prüft Bedingungen und Aktionen
    - anwendbar bei Systemanforderungen bzw. Spezifikationen mit logischen Bedingungen und komplexen, vom System umzusetzende Regeln
      → bei regelbasierten Anforderungen
    - Entscheidungstabelle enthält Kombinationen von wahr und falsch für alle Eingabebedingungen und daraus resultierenden Aktionen
    - wenigstens ein Testfall pro Spalte
    - Entscheidungstabelle ist
        - vollständig: bei n Bedingungen 2^n Kombinationen enthalten
        - redundanzfrei: speziellere Bedingungen führen zu anderen Aktionen (betrifft Spalten oben und unten)
        - widerspruchsfrei: gleiche Bedingungen führen zu gleichen Aktionen (betrifft Spalten oben und unten)
    - Vorteile
        - leitet Kombinationen von Bedingungen ab, die andernfalls beim Test evtl. nicht ausgeführt werden
        - in allen Situationen anwendbar, in denen Abläufe von mehreren logischen Entscheidungen abhängen
        - Zwingen nicht zur Strukturierung eines Ablaufs

    - Nachteile
        - unübersichtlich bei vielen Bedingungen
        - Zusammenhänge zwischen einzelnen Bedingungen nur implizit ausdrückbar
- **anwendungsfallbasierter Test**: testet Szenarien/Anwendungsfälle der Systemnutzung
    - Anwendungsfall: Folge von Interaktionen zwischen Aktoren (Anwender oder Systeme), die zu konkretem Ereignis führen
    - Verwendung: System- und Abnahmetests
    - normalen Ablauf testen, aber auch alternative Abläufe, Ausnahmeabläufe

Bewertung der Blackbox-Testentwurfsverfahren

- Grundlage: Anforderungen, Spezifikation
- nicht geforderte Funktionalität nicht erkannt
- Prüfung der Funktionalität im Mittelpunkt

Abschnitte des zustandsbasierten Tests

1. Fokussierung auf Zustandsdiagramm
2. Prüfung auf Vollständigkeit: Zustandsdiagramm hinsichtlich Vollständigkeit untersuchen, ggf. Zustandsübergangstabelle anlegen
3. Ableiten des Übergangsbaumes für Zustands-Konformanztest
    - jeder Pfad von Wurzel bis zum Blatt ist Testfall
4. Erweitern des Übergangsbaumes für Zustands-Robustheitstest
    - Robustheit unter spezifikationsverletzenden Benutzereingaben prüfen
      → Zustand „Fehler“ für Knoten, wenn kein Übergang spezifiziert
5. Generieren der Botschaftssequenzen und Ergänzen der Botschaftsparameter
    - Pfade von Wurzel zu Blättern im erweiterten Übergangsbaum als Funktions-Sequenzen auffassen
    - Parameter ergänzen
6. Ausführen der Tests und Überdeckungsmessung

Whitebox-Testentwurfsverfahren

- **Anweisungsüberdeckung** (C0-Überdeckung, alle Knoten des Kontrollflussgraphen)
    - Anweisungsüberdeckung
      = Anzahl durchlaufener Anweisungen / Gesamtzahl Anweisungen
    - schwaches Kriterium, da „leere Kante“ (Kante, die nur andere Konten überbrückt) ohne Bedeutung (z. B. if-Block ohne else)
    - fehlende Anweisungen nicht erkannt
- Entscheidungs- und Zweigüberdeckung
    - **Entscheidungsabdeckung** (C1-Überdeckung, auch alle ausgehenden Kanten)

        - Entscheidungssüberdeckung
          = Anzahl durchlaufener Entscheidungen / Gesamtzahl Entscheidungen
        - Entscheidungstest: Testfälle in Hinblick auf Entscheidungsausgänge entwerfen
        - Entscheidung: Knoten mit mindestens 2 ausgehenden Kanten
        - Entscheidungsausgang: Ergebnis einer Entscheidung
        - 100 % Entscheidungsüberdeckung: alle Fälle jeder Entscheidung abgedeckt
          → 100 % Zweigüberdeckung und 100 % Anweisungsüberdeckung
    - **Zweigüberdeckung** (C2-Überdeckung, alle Kanten im Kontrollfluss abdecken, auch „leere" Kante müssen durchlaufen werden)
        - Zweigüberdeckung
          = Anzahl durchlaufener Zweige / Gesamtzahl Zweige
    - für OO nur unzureichend geeignet, da hier Komplexität in Beziehungen zwischen Klassen verborgen ist
- **Bedingungsüberdeckung**: Anteil der (atomaren) Teilbedingungen, die durch Gruppe von Testfällen ausgeführt wurden
  → 100 % bedeutet, jede atomare Teilbedingung in jeder Entscheidung mindestens einmal wahr/falsch
- Mehrfachbedingungstest: fordert Überdeckung atomarer Teilbedingungen einer Entscheidung mit wahr/falsch in allen Kombinationen
- minimal bestimmende Mehrfachbedingungsüberdeckung: Anteil aller einfachen Bedingungsergebnisse, die unabhängig voneinander Entscheidungsausgang beeinflussen
- **Instrumentierung**: zusätzliche Anweisungen und Zähler zur Ermittlung der Überdeckung

Erfahrungsbasiertes Testen

- Erfahrung/Wissen der Tester wird bei Definition der Testfälle einbezogen
- lässt sich nicht eindeutig in Blackbox oder Whitebox einordnen
- eher in höheren Teststufen
- **Error Guessing**: Erfahrung/Wissen, um Fehlerzustände vorherzusagen
    - Systematische Vorgehensweise
        - Fehlerkatalog mit möglichen Fehlerzuständen und Fehlerwirkungen erstellen
        - Testfälle, die darauf abzielen entwerfen
    - Fehlerkataloge: Liste möglicher Fehler und fehlerverdächtige Situationen
- **exploratives Testen**: informelles testen, keine Testvorbereitung, keine erkennbaren Testentwurfsverfahren
    - gleichzeitige Testfallanalyse, Testfallentwurf, Testrealisierung und -durchführung

    - Test-Charta: Testziele, aber kein detaillierter Testplan
    - zeitlich begrenzte Sitzung
    - Sitzungsprotokolle zur Auswertung und Fortschrittskontrolle
    - Vorteile
        - ergänzt systematischen Testentwurf
        - deckt Fehler auf, die schwer durch systematische Verfahren aufgedeckt werden
        - kurzzeitig einsetzbar
        - anwendbar, wenn wenig Doku und Domänenwissen vorhanden
    - Nachteile
        - Verfall in bestimmte Denkmuster verhindert Aufdecken wichtiger Probleme
        - vom Wissen/Erfahung der Tester abhängig
        - keine Automatisierbarkeit

Dynamischer Test

- Dynamischer Test: Prüfung durch Ausführung
- Testbasis: alle Dokumente, aus denen Anforderungen ersichtlich werden bzw. Dokumentation
- Testbedingung: Einheit/Ereignis, z. B. Funktion, Transaktion, Feature, Qualitätsmerkmal, strukturelles Element
- Ziel: stichprobenhafter Nachweis der Erfüllung der festgelegten Anforderungen
- Testentwurfsspezifikation: Ergebnisdokument mit Testbedingungen für Testobjekt, detaillierter Testvorgehensweise, logischen Testfällen
    - Testfallspezifikation: spezifiziert Menge von Testfällen
    - Testablaufspezifikation: legt Folge von Schritten zur Testausführung fest
- Testsuite: Zusammenstellung mehrerer Testfälle
- Testskript: automatisierte Testablaufspezifikation
- Testausführungsplan: Plan für Ausführung von Testskripten
- Testablauf: Ausführung der Testfälle mit bestimmter Version des Testobjekts
- **Rückverfolgbarkeit**
    - vertikal: Rückverfolgung von Anforderungen durch Ebenen der Entwurfsdokumentation bis zu Komponenten
    - horizontal: Verfolgen von Anforderungen einer Teststufe über Ebenen der Testdokumentation
- **Testrahmen** (test harness, test bed): Sammlung aller Programme (z. B. Treiber, Platzhalter), um Testfälle auszuführen, auszuwerten und Protokolle aufzuzeichen

- **Eingangskriterien**: Menge der Bedingungen in Prozess, um in Aktivität fortzuschreiten
- **Endekriterien**: Menge abgestimmter Bedingungen, von allen akzeptiert; verhindern, dass Aktivität als abgeschlossen betrachtet wird, obwohl Teile WIP

Vorgehensweise

1. Entwerfen von Tests durch Ermittlung von Testbedingungen
2. Spezifizieren der Testfälle
3. Spezifizieren der Testablaufspezifikation
4. Testausführungsplan

Testmanagement

Unabhängiges Testen

- Testaufgaben können übernommen werden von Personen …
  - … in spezifischer Testerrolle
  - … oder anderer, z. B. Projektmanager, Qualitätsmanager, Entwickler
- Vorteile von unabhängigen Testern
  - unabhängige Tester sind unvoreingenommen und sehen dadurch andere Fehlermöglichkeiten
  - unabhängige Tester können Annahmen überprüfen, die von Entwicklern während Spezifikation gemacht wurden
- Nachteile von unabhängigen Testern
  - hoher Kommunikationsaufwand
  - unabhängiges Testteam als letzte Prüfinstanz Engpass
- Je größer, komplexer oder sicherheitskritischer, desto wichtiger unabhängiges Testen
- niedrige Teststufen (Komponententests, Integrationstests) können von Entwicklern durchgeführt werden

Aufgaben und Mitarbeiterqualifikation

- Zur Durchführung sollen Spezialisten zur Verfügung stehen
- zu besetzende Rollen: Testmanager (Testleiter), Tester (Testdesigner, Testautomatisierer, Testadministrator, …)
- benötigte Qualifikation: Teamfähigkeit, scheinbare Tatsachen hinterfragen, Durchsetzungskraft, Kreativität, Exaktheit
- Testmanager
  - **Testrichtlinie erstellen**: Vorgehensweise auf hohem Abstraktionsniveau
  - **Teststrategie erstellen**: abstrakte Beschreibung der Teststufen mit Ein- und Endekriterien

    - **Planung und Umsetzung koordinieren**
- Tester
    - Anforderungen, Spezifikationen und Modelle auf Testbarkeit prüfen
    - Mitarbeit an Erstellung des Testkonzepts
    - Erstellen der Testspezifikation
    - Implementierung der Testumgebung
    - Ausarbeitung von Testdaten
    - Umsetzung/Durchführung von Tests auf allen Stufen
    - Protokollierung, Auswertung der Testergebnisse
    - Testautomatisierung
    - Performance messen

Testplanung – Wann mit Testen beginnen?

- so früh wie möglich
  → im V-Modell am Ende jeder Phase Verifikation der Ergebnisse
- Testvorbereitung startet früher (parallel zu Entwicklungsschritten im linken Ast)

Aktivitäten

- Festlegung der projektspezifischen Teststrategie, Teststufen, Ein- und Endekriterien
- Integration und Koordination der Testaktivitäten mit den Aktivitäten im Softwarelebenszyklus
- Entscheidung treffen
    - welche Teile wie intensiv testen
    - von wem, wann und wie Testaktivitäten auszuführen
    - wie Testergebnisse bewerten
    - wann Endekriterien erfüllt sind
- Zuordnung notwendiger Ressourcen
- Umfang, Detaillierungsgrad, Struktur der Testdokumentation festlegen
- Metriken auswählen
- Detaillierungsgrad der Beschreibung des Testvorgehens festlegen
  → reproduzierbare Testvorbereitung und -durchführung sicherstellen

Weitere Punkte

- Entwicklungsstand: zu Beginn des Testzyklus tatsächlich verfügbare Software
- Testergebnisse: in vorgehenden Testzyklen aufgedeckten Probleme machen evtl. Änderung in Priorisierung nötig
- Ressourcen: Planung des aktuellen Testzyklus muss mit aktuellen Projektplan in

Einklang stehen; zu beachten
    - Auswirkungen der aktuellen Urlaubsplanung
    - momentane Verfügbarkeit der Testumgebung, spezieller Werkzeuge

Einflussfaktoren

- Reifegrad des Entwicklungsprozesses
- Testbarkeit der Software
- Testinfrastruktur
- Mitarbeiterqualifikation
- Qualitätsziele
- Testrichtlinie der Organisation und Teststrategie

Testvorgehensweisen

- Auswahl soll Rahmenbedingungen berücksichtigen
    - Risiko des Scheiterns des Projekts
    - Risiken für Produkt und Risiken für Personen, Umwelt, Unternehmen, …
    - Qualifikation/Erfahrung der Personen
    - Testziel und Auftrag
    - Regularien
    - Art des Produkts/Geschäftfelds
- Mix von Vorgehensweisen auswählen
- Kosten des Testens geringer als Fehlerkosten

Fehlkosten

- Prüfungen und Tests im Umfang reduziert oder ganz eingespart
  → Zahl der unentdeckten Fehlerzustände/Mängel
- direkte Fehlkosten: entstehen Kunden beim Betrieb, z. B. Datenverlust
- indirekte Fehlkosten: Verlust von Kunden, Imageschaden
- Fehlerkorrekturkosten: entstehen Hersteller im Zuge der Fehlerkorrektur

→ Fehler möglichst früh finden, denn Kosten steigen rapide über Entwicklungsphasen an

Testaufwand

- Hängt ab von
    - Merkmalen des Produkts
    - Merkmalen des Entwicklungsprozesses
    - von Ergebnissen der Tests
- Aufwandsschätzung

- Metrikenschätzung: normalerweise besser
- Expertenschätzung: hohe Subjektivität, große Schätzunsicherheit, einfachstes Verfahren, kein Rechenaufwand

Metriken zur Testfortschrittsüberwachung und -steuerung

- **Testmetrik**: Messbare Eigenschaft eines Testfalls, Testlaufs oder Testzyklus mit Angabe zugehöriger Messvorschrift
  - **fehlerbasiert**, z. B. Zahl gefundener Fehlerzustände, gefundene/behobene Fehler
  - **testfallbasiert**, z. B. Anzahl geplanter Tests, Verhältnis spezifizierter zu geplanten Testfällen
  - **testobjektbasiert**, z. B. Codeüberdeckung
  - **kostenbasiert**: Kostenvergleich für Auffinden des nächsten Fehlers und nächsten Testdurchlauf
  - subjektives **Vertrauen** der Tester **in Produkt**
- Überwachung, Erfolgsprotokolle anhand objektiver Testmetriken, Endekriterien
- nur Metriken verwenden, die regelmäßig, zuverlässig, einfach zu messen sind
- Standortbestimmung
  - gemessene Daten dienen zur Standortbestimmung
    - Wie weit ist Test vorangekommen?
    - Kann Test beendet und Produkt ausgeliefert werden?
  - Bedingungen zur Bestimmung des Testendes
  - Endekriterien ebenfalls im Testkonzept festgelegt
  - Testendekriterium muss sich aus laufend erhobenen Testmetriken berechnen lassen
- Endekriterien
  - Ziel: Festlegung, wann Testen fertig
  - z. B.: Überdeckungsmaße, Zuverlässigkeitsschätzung, Kosten, Termin der Auslieferung
  - Metriken möglichst automatisch sammeln
- Teststatusbericht
  - von Testmanager nach jedem Testzyklus erstellt
  - Teststatus: geplant, durchgeführt/gelaufen, blockiert
  - Fehlerstatus: neu, offen, korrigiert
  - Risiken: neue, veränderte, bekannte
  - Ausblick: Planung des nächsten Testzyklus
  - Gesamtbewertung: Beurteilung verbleibender Fehler, weiteres Testen ökonomisch sinnvoll?, …

    - Aspekte: Eignung der Testziele für Teststufe, Eignung der Teststrategie, Effektivität der Tests zur Erreichung festgelegter Ziele
- Teststeuerung
    - Verzögerung gegenüber Projekt- oder Testplanung → Gegenmaßnamen
        - neu erkannte Risiken → neue Prio
        - Anpassung zeitlicher Planung
        - korrigierte Fehlerzustände nachtesten
    - zusätzlich Ressourcen anfordern, wenn nicht möglich
        - unwichtige Tests streichen
        - Anzahl der Testvarianten verringern
    - Testmanager muss Planänderung dokumentieren

Risikomanagement

- **Risiko**: potentielles Problem, Wahrscheinlichkeit * Schadensausmaß
- Projektrisiken: beziehen sich auf Erreichung der Ziele
    - Lieferantenaspekte (Versagen 3. Partei, Vertragsaspekte)
    - organisatorische Aspekte: Qualifikation, Mitarbeiterengpässe, Personal-, Trainingsaspekte
    - politische Aspekte: Kommunikationsprobleme, …
    - technische Aspekte: ungenaue Anforderungen, schlechtes Design des Codes, Daten, Tests, …
- Produktrisiken: beziehen sich auf Testobjekt und seinen Einsatz
    - durch Produkt verursachte Schäden
    - Risiken bezüglich Qualität des Produkts: Auslieferung fehlerhafter Software
- **Risikomanagement**: systematische Anwendung von Praktiken für Risikoidentifizierung, Risikoanalyse, Risikopriorisierung, Risikosteuerung
    - Aktivitäten: Ermittlung Risikokontext, Risikoidentifikation, Risikoanalyse/Priorisierung, Risikosteuerung/Bewältigung, Risikoprüfung/Überwachung
- **Risikoorienter Testansatz**
    - pro-aktive Möglichkeit zur Reduzierung des Produktrisikos
    - beginnend mit Projektstart
    - Identifikation von Risiken und ihre Berücksichtigung bei Steuerung der Testplanung, Erstellung, Durchführung von Tests
    - Identifizierte Risiken bestimmen
        - Testverfahren, Umfang
        - Priorisierung

        - weitere Tests notwendig?
    - nutzt Wissen und Kenntnisse der Stakeholder, um Risiken zu analysieren

Fehler- und Abweichungsmanagement

- Verfahren zur Übermittlung/Verwaltung von Fehlermeldungen nötig, um Mängel, Probleme und Fehlerwirkungen zu korrigieren
- beginnt nach Ende eines Testlaufs, umfasst
    - Testprotokoll
    - Abweichungsbericht/Fehlermeldung
    - Fehlerklassifikation
    - Fehlerstatus
- Testprotokoll
    - Auswertung angefallener Testprotokolle
    - Fehler dokumentieren
    - Ursachenanalyse → machen Entwickler
- Fehlerdatenbank: erfasst Probleme/Mängel/Fehlerwirkungen
- Problemmeldeverfahren/Fehlermeldeverfahren
    - alle offenen Probleme werden gemeldet
    - dokumentiert Korrektur am Fehlerobjekt
    - Bild über Anzahl, Status, … der Fehler
- **einheitliches Schema der Fehlermeldung, enthält**
    - Testobjekt, Identifikation, Status, Fehlerklasse, Priorität, Testumgebung, Reproduzieren/Lokalisieren, Name des Testers
    - nach IEEE 829
        - Datum der Entdeckung
        - Identifikation der Software
        - Aktivität
        - Nummer des Testfalls
        - Schwere
        - Dringlichkeit/Priorität
        - Status: offen, zurückgestellt, abgewiesen, Duplikat, Korrektur, Nachtest, erledigt
        - Schlussfolgerungen, Empfehlungen
        - globale Probleme
        - Änderungshistorie zur Korrektur
    - außerdem: objektiv, minimaler Aufwand für Entwickler

- Fehlerstatusmodell: siehe Skript

Konfigurationsmanagement

- SW-System besteht aus Vielzahl von Einzelbausteinen, die zueinander passen müssen
  → KM veraltet diese Bausteine
- Ziel: Schaffung/Erhaltung der Integrität
- Symptome von unzureichenden KM
  - Entwickler überschreiben gegenseitig gemeinsam genutzten Code
  - Integrationsarbeiten behindert
  - Fehleranalyse, Fehlerkorrektur erschwert, weil unklar wie Code zwischen Versionen genau geändert
  - Test, Testauswertung behindert, weil unklar, welche Testfälle zu welchem Versionsstand gehören
- Versionsverwaltung
- Konfigurationsverwaltung: Bestimmung und Verwaltung aller Daten in jeweils passenden Version
  - Voraussetzung ist Versionsverwaltung
  - unterscheidet zwischen Konfigurationen, z. B. Deutsche vs. Englische Version, Windows- vs. Linux-Version
- Anforderung an KM
  - Statusverfolgung von Fehlern und Änderungen
  - Konfigurationsaudits

Zusammenfassung

- Entwicklungs- und Testaktivitäten sollen organisatorisch getrennt sein
  → je klarer Trennung, umso wirksamer kann getestet werden
- je nach Aufgaben Mitarbeiter mit rollenspezifischen Testkenntnissen benötigt (neben flachlichen Fähigkeiten auch soziale Kompetenz gefragt)
- **Aufgaben des Testmanagers**: Plaung, Überwachung und Steuerung der einzelnen Testzyklen, Testkonzept (Testvorgehensweise/Teststrategie, Testumfang, Werkzeuge, …)
- hohe Fehlkosten durch übersehene Fehler/Mängel möglich
  → optimales Verhältnis zwischen Testkosten, verfügbaren Ressourcen, drohenden Fehlerkosten
- Risikomanagement, risikoorientierter Test: helfen, kritische Fehler frühzeitig zu finden und Testaufwand zu optimieren
- Fehlermanagement, Konfigurationsmanagement (Verwaltung der SW-Bauseine): Basis für effizienten Testprozess
- Fehlermeldungen benötigen einzheitliches Schema

- Normen und Standards: Empfehlungen zur fachgerechten Durchführung von SW-Tests

Testwerkzeuge

- Fehlerdatenbank bzw. Fehlermanagementwerkzeug
- Anforderungsmanagementwerkzeuge
- Konfigurationsmanagementwerkzeug
- Testausführungswerkzeug (Testautomatisierung)
- Statische Analysewerkzeug
- Werkzeuge für Codeüberdeckung (Whitebox)
- Debugger (aber kein typisches Testwerkzeug)

Typen

- **intrusive Werkzeuge**: ändern das Testobjekt
- Werkzeugstützung für
  - Testmanagement
    - Anforderungsmanagementwerkzeuge: Erfassung, Katalogisierung, strukturierte Ablage, Verwaltung und Änderungsmanagement, Priorisierung von Anforderungen (z. B. DOORS)
    - Fehler- und Abweichungsmanagementwerkzeuge: Verfolgung der Fehler über Zeit, liefern Berichte
    - Konfigurationsmanagementwerkzeuge: keine Testwerkzeuge im engeren Sinne, Identifikation, Verwaltung, Bereitstellung, Speicherung der Information über Versionen/Konfiguration der Software und Testmittel
  - statische Tests
    - Analysieren Programmcode: strukturelle Eigenschaften, Datenflussanomalien, Konventionen
    - Fortgeschrittene Funktionalität: Visualisierung von Metriken, Klonerkennung, Zykluserkennung
    - Modellierungswerkzeuge
  - Testspezifikation
    - Testentwurfswerkzeuge: generieren Testfälle, Testorakel
  - Testdurchführung und Protokollierung
    - (halb)automatische Ausführung von Testfällen
    - Typen
      - Testausführungswerkzeuge
        - automatisieren funktionale Tests
        - Capture & Replay, skriptbasiertes Testen, datengetriebenes Testen,

schlüsselwortgetriebenes Testen,
Testrahmen/Komponententestrahmen

- (Komponenten)testrahmen
- Simulatoren
- Vergleichswerkzeuge
- Coverage-Tools
- Sicherheitsprüfungwerkzeuge

- Performancemessung, Monitoring
  - Performance-, Last-, Stresstestwerkzeuge: testen nicht-funktionale Anforderungen, generieren „synthetische Last“, z. B. Datenbankabfragen
  - Testmonitore: Aufzeichnung, Analyse, Netzwerkverkehr, Datenbankbelastung; warnen vor Problemen mit Diensten
- spezifische Anwendungsbereiche
  - z. B. Unterstützung für GUI-Tests, Werkzeug-Suiten für eingebettete Systeme

Vor- und Nachteile

- Nutzen
  - effizienter → Einsparung von Ressourcen
  - erhöhen Standardisierung und Transparenz der Dokumentation
  - objektive Messung
  - hoher Automatisierungsgrad
- Risiken
  - unrealistische Erwartungen
  - Unterschätzung der Zeit, Kosten, Aufwands
  - blindes Vertrauen in Werkzeug
  - Vernachlässigung der Komplexität
  - Risiko, dass Werkzeug nicht mehr weiterentwickelt wird oder Support unzureichend
- Einführung
  - zuerst Pilotprojekt
  - intellektuell anspruchsvolle Aktivitäten zuletzt automatisieren
  - bedingt funktionierenden Testprozess

Fragen

- Warum ist jeder Test nur eine stichprobenartige Prüfung?
  - Vollständiges Testen nicht möglich, da selbst einfache Programme zu viele

Testfälle benötigen, um sie in der Praxis komplett durchzutesten.

- Bsp.: 5 unabhängige Verzweigungen in Schleife mit 20 Wiederholungen → über 10^{14} Testfälle

- Warum sollte Entwickler nicht seine eigenen Programme testen?
  - Entwickler evtl. blind gegenüber eigener Fehler, da
    - selbes Denkmuster wie schon beim Entwickeln
    - Aufgabenstellung schon falsch verstanden
  - aber: Entwickler kennt Testobjekt → kein Einarbeitungsaufwand
- Welche Gründe sprechen dafür, Tests in separater Testinfrastruktur durchzuführen?
  - Produktivumgebung des Kuden nicht durch Fehlerwirkungen beeinträchtigen
  - Testumgebung ermöglicht reproduzierbare Tests, da Tester bessere Kontrolle über Umgebung haben
- Lasttest vs. Performancetest vs. Stresstest?
  - bei Lasttest Messung *des Verhaltens* bei steigender Last
  - bei Performancetest Messung *des Antwortzeit für bestimmte Use Cases* bei steigender Last
  - bei Stresstest Bebachten des Verhaltens bei *Überlast*
- Fehlernachtest vs. Regressionstest
  - Fehlernachtest testet, ob *konkreter, zuvor gefundener* Fehlerzustand nach Änderung beseitigt wurde
  - Regressionstest testet *allgemein*, ob durch Änderungen *keine neuen* Fehlerstände hinzugekommen sind
- Warum sind Reviews effizientes Mittel zur Qualitätssicherung?

  Fehler werden frühzeitig erkannt, weniger Aufwand für dynamisches Testen, Wissensaustausch im Team
- Zusammenhang zwischen statische Analyse und Reviews

  Statische Analyse kann vor Review durchgeführt werden, um bereits im Vorfeld Fehlerzustände/Unstimmigkeiten zu finden.
  → spart Arbeit bei Review
- Warum kann statische Analyse nicht alle im Programm enthaltenen Fehlerzustände aufdecken?

  Manche Fehlerzustände erst zur Laufzeit ermittelbar.
  (z. B. Variable, die als Divisor verwendet wird, kann zur Laufzeit 0 werden)
- Typische Fehlerzustände, die mit statischer Analyse gefunden werden können

  Verletzung von Syntax, Sicherheitslücken, Abweichung von Konventionen, Daten- und Kontrollflussanomalien
- Unterschied zwischen dynamischen und statischen Tests?

Bei dynamischen Tests wird Testobjekt ausgeführt, bei statischen nicht.

- Unterschied zwischen Whitebox- und Blackbox-Testentwurfsverfahren?

  Bei Whitebox wird innere Struktur bzw. Programmcode verwendet, bei Blackbox nicht.

- Worauf basieren funktionale Tests?

  entweder anforderungsbasiert, geschäftsprozessorientiert oder anwendungsfallbasiert (siehe Testarten)

- Was ist Äquivalenzklassenbildung?
  - zerlegt Definitionsbereiche der Ein- und Ausgaben in Klassen, mit äquivalentem Verhalten
  - Stichprobe: ein Testwert pro Klasse
- Wann ist Grenzwertanalyse einsetzbar?
  - bei Bedingungen mit Grenzbereiche
  - in Kombination mit anderen Verfahren, z. B. Äquivalenzklassenbildung
- Wann ist zustandsbasierter Test einsetzbar?

  wenn Zustände eine Rolle spielen und Funktionalität von Zustand beeinflusst wird, z. B. objektorientierte Programmierung

- Wie ist Entscheidungstabelle aufgebaut?

Bedingungen mögliche Zustände	Regeln Kombinationen von Bedingungswerten
Aktionen die abhängig von Regeln auszuführen sind	Aktionszeiger Beledungen der Bedingungen mit Aktionen

  z. B.

Bestellmenge > 0	N	J
Bestellmenge > Art-Lagermenge	-	J
Melde „Bestellmenge ungültig“	X	
Melde „Menge nicht ausreichend“		X

- Wann ist anwendungsfallbasierter Test einsetzbar?
  - für Szenarien der Systembenutzung, also in System- und Abnahmetests
- Worin unterscheiden sich Anweisung- und Entscheidungsüberdeckung?

  Anweisungdüberdeckung unterscheidet lediglich, ob Anweisungen ausgeführt wurden oder nicht. Die Entscheidungsüberdeckung fordert dagegen, dass alle Möglichkeiten einer Entscheidung getestet sind (also z. B. auch Fall, indem Anweisungen innerhalb von If-Block *nicht* ausgeführt werden).

- Wozu Instrumentierung?

  Instrumentierung ist Voraussetzung, um Coverage zu ermitteln.

- Erfahrungsbasiertes Testen?

  Erfahrung der Tester und Fehlerkataloge werden bei Definition der Testfälle mit einbezogen.
- Modelle für Aufgabenteilung zwischen Entwicklertn und Test?
  - Entwickler testen eigenen Code
  - Entwickler testen ihren Code gegenseitig
  - innerhalb der Organisation unabhängige Testteams
  - für spezielle Tests unabhängige Testspezialisten
  - externe Organisation übernimmt testen
- Typesche Aufgaben des Testmanagers und der Tester?
  - Testmanager
    - **Testrichtlinie** erstellen: Vorgehensweise auf hohem Abstraktionsniveau
    - **Teststrategie** erstellen: abstrakte Beschreibung der Teststufen mit Ein- und Endekriterien
    - **Planung** und **Umsetzung koordinieren**
    - **Einleiten von Analyse und Entwurf und der Realisierung und Durchführung von Tests**
    - **Testberichte** erstellen (z. B. Teststatusbericht)
    - **Konfigurationsmanagement** einrichten
    - **Metriken** auswählen und erheben
    - **Automatisierungsgrad** festlegen
    - **Werkzeugwahl**
    - Realisierung der **Testumgebung** festlegen
  - Tester
    - **Anforderungen, Spezifikationen, Modelle** des Testobjekts prüfen
    - bei **Testkonzept** mithelfen
    - **Testspezifikation** erstellen
    - **Testumgebung** implementieren
    - **Testdaten** ausarbeiten
    - **Tests** umsetzen/durchführen
    - Testautomatisierung
    - Performance messen
- Zwei Verfahren zur Messung des Testaufwands?

  Metrikenschätzung, Expertenschätzung
- Metriken zur Überwachung des Testaufwands?

  fehlerbasiert (z. B. Anzahl gefundener Fehler), testfallbasiert (z. B. Anzahl

geplanter Tests), testobjektbasiert (z. B. Coverage), kostenbasiert

- Was enthält Teststatusbericht?
  - Teststatus, Fehlerstatus, Risiken, Ausbklick, Gesamtbewertung, …
- Aktivitäten des Resikomanagements?

  Ermittlung des Risikokontextes, Risikoidentifikation, Risikoanalyse und -priorisierung, Risikosteuerung und -bewältigung, Risikoüberprüfung und -überwachung
- Wie berechnet man Risiko?

  Wahrscheinlichkeit * Schadensausmaß
- Typische Produkt- und Projektrisiken?
  - Projektrisiken: Lieferantenaspekte, Mitarbeiterqualifikation, Kommunikationsprobleme, ungenaue Anforderungen, schlechtes Design des Codes, …
  - Produktrisiken: Auslieferung fehlerhafter Software, Schaden durch Produkt
- Welche Typen von Testwerkzeugen werden unterschieden?
  - intrusive und nicht-intrusive Werkzeuge (intrusive Werkzeuge ändern Testobjekt)
  - Einteilung nach Anwendungsgebiet
    - Testmanagement: Anforderungsmanagementwerkzeuge, Fehlerdatenbank, Konfigurationsmanagementwerkzeuge
    - statische Tests: Tools zur Programmcode-Analyse, Modellierungswerkzeuge
    - Testspezifikation: Testentwurfswerkzeuge zur Generierung von Testfällen, Testorakel
    - Durchführung und Protokollierung: Testausführungswerkzeuge zur Automatisierung, Simulatoren, Vergleichswerkzeuge, Coverage-Tools
    - Performance: Tools die Last generieren
    - Monitoring: Testmonitore
- Zweck von Werkzeugunterstützung?
  - erhöhen Effizienz → Einsparung von Ressourcen
  - erhöhen Standardisierung und Transparenz der Dokumentation
  - objektive Messung
  - ermöglichen hohen Automatisierungsgrad
- Grundlegende Funktionen von Testmanagementwerkzeugen?
  - Erfassung, Katalogisierung, Verwaltung und Priorisierung von Testfällen
- Grundlegende Funktionen von Fehlermanagementwerkzeugen?
  - „Bugtracker“: Fehler erfassen, verwalten, über Zeit verfolgen

    - liefern Berichte
- Typische Funktionalität von Werkzeugen im Review-Prozess?

    Verwaltung der Dokumente, Unterstützung der Gutachter, Auswertung, Unterstützung der Reviewsitzung
- Welche typischen Charakteristika des Programmcodes können mit Hilfe von Werkzeugen zur statischen Analyse analysiert werden?

    strukturelle Eigenschafte (z. B. Zyklomatische Zahl), Datenflussanomalien, Einhaltung von Programmierkonventionen, Einhaltung von Konventionen zur sicheren Programmierung
- Ansätze zur Automatisierung der Testdurchführung?

    Capture & Reply, skriptbasiertes Testen, datengetriebenes Testen, schlüsselwortgetriebenes Testen, Testrahmen/Komponentenrahmen
- Welche Werkzeuge bei Komponenten- bzw. Integrationstests geeignet?

    Entwicklungsumgebung, Debugger, stat. Analysewerkzeuge, Testmonitore
- Vorteile und Risiken der Werkzeugunterstützung?
    - Vorteile: siehe Zweck
    - Risiken: unrealistische Erwartungen, Zeit/Kosten/Aufwand unterschätzen, blindes Vertrauen in Werkzeug, Vernachlässigung der Komplexität, schlechter Support
- Schritte der Werkzeugwahl?

    Anforderungsspezifikation , Marktstudie (Kandidation auschecken), Demos, engere Wahl evaluieren, Review der Ergebnisse
- Tätigkeiten bei Einführung eines Testwerkzeugs?

    Pilotprojekt, Review der Pilotprojekterfahrungen, Prozessanpassung, Anwenderschulung, Breiteneinführung und begleitendes Coaching
- Ziele eines Pilotprojekts im Rahmen der Einführung eines Testwerkzeugs?
    - genaueres Kennenlernen des Werkzeugs
    - Analyse, inwieweit Werkzeug mit existierenden Werkzeugen und Prozessen zusammenpasst
    - Kosten-Nutzen-Bewertung
    - Entscheidung über Standardisierung