ISTQB Certified Tester Foundation Level

Prof. Dr. Bettina Buth

[1]Department Informatik
HAW Hamburg

09/2019

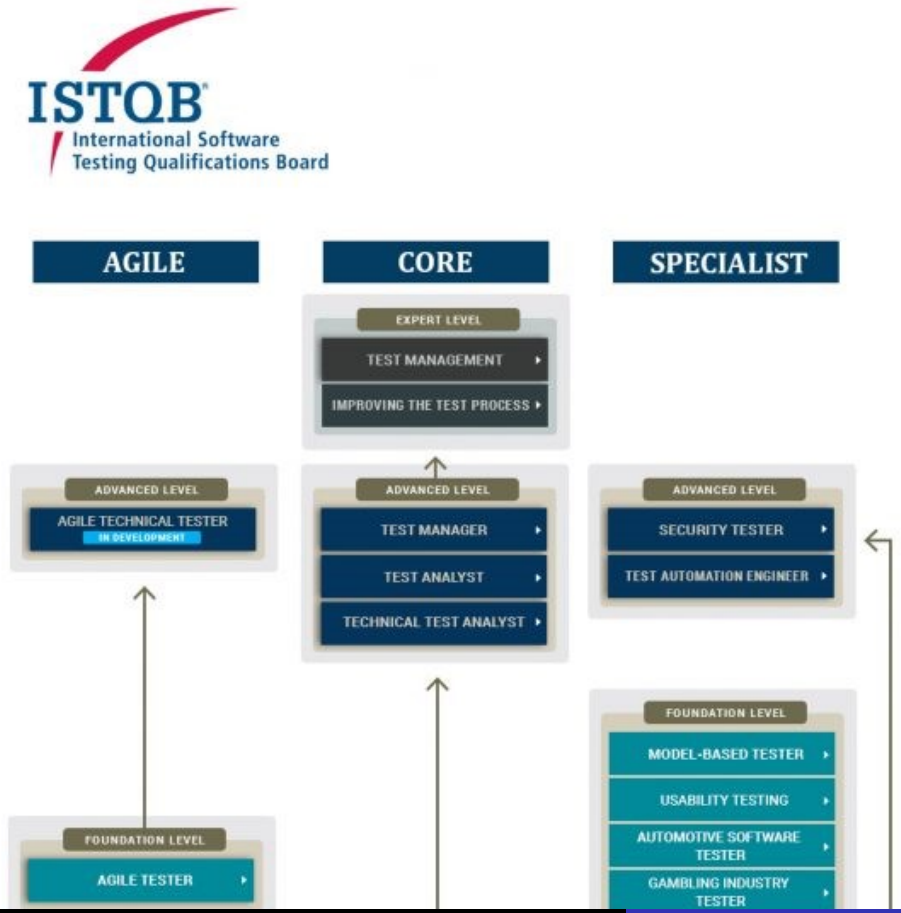
Teil I

Einleitung

Certified Tester - warum und wofür

- Standardisierung der Sprechweisen, Terminologie
- durch ISTQB (International Software Testing Qualifications Board), GTB (German Testing Board)
- Ziel: professionelles Testen einheitlich vermitteln
  - Definitionen (remember, recognize, recall)
  - Verstï¿½ndnis (understand, explain, give reasons, compare, classify, categorize, give example, summarize)
  - Anwendung (apply, use)
- hohe Akzeptanz - in England schon obligatorisch für Tester
- Gesamtprogramm:
  - Foundation Level
    - neu: Extensions FL - Agile Testing, Automotive Testing, Model-based Testing
  - Advanced Level - Testmanager, Test Analyst, Technical Test Analyst
  - geplant: Expert Level - verschiedene Topics in Vorbereitung
  - TTCN3
- Voraussetzung für Advanced Level: CTFL + 18 Monate Testpraxis
- CTFL mittlerweile auch als Kurs an Hochschulen

CTFL im ISTQB Kursrahmen



Kursübersicht nach CTFL

- Grundlagen des Software Testens (Fundamentals of Software Testing)
- Testen im Entwicklungszyklus (Testing throughout the Software Lifecycle)
- Statisches Testen (Static Test)
- Dynamisches Testen (Dynamic Analysis - Test Design Techniques)
- Test Management
- Testwerkzeuge (Test Tools)

Prüfungen für CTFL nach ISTQB (1)

- orientiert an Umfang Inhalt und Kenntnis-Level Kx
  - K1: remember, recognize, recall – Definitionen
  - K2: understand, explain, give reasons, compare, categorize, give examples, summarize – Verstündnis
  - K2: apply, use – Anwendung
  - K2: analyze – Analysieren
- CTFL Syllabus: Ausweisen der K-Level bei den Learning Objectives
- Fragen nach Kx-Level: 20 K1, 12 K2, 8 K3+K4 = 40

s. auch `http://www.istqb.org/certification-path-root/foundation-level/foundation-level-learning-objectives.html`

Prüfungen für CTFL nach ISTQB (2)

- 40 Multiple Choice Fragen in 60 min (oder 75 für non-native speaker)
- Fundamentals of Software Testing - ca 7 Fragen
- Testing Throughout the Software Life Cycle - ca 6 Fragen
- Static Techniques - ca 3 Fragen
- Dynamic Analysis - Testing Design Techniques - ca 13 Fragen
- Test Management - ca 8 Fragen
- Tool Support for Testing - ca 4 Fragen

s. auch `http://www.istqb.org/certification-path-root/foundation-level/foundation-level-exam-structure.html`

Vorgehen im Kurs hier

- Material enthült Info nach Spillner, Linz resp. Spillner, Linz, Schaefer
  - Spillner, Linz: Basiswissen Softwaretest, dpunkt.verlag 6. Auflage 2019
  - Spillner Linz, Schaefer: Software Testing Foundations, 4th edition, rockynook 2014
- Angepasst so weit bekannt auf Vorkenntnisse
- Grundlage für weitere eigene Vorbereitung auf Zertifikaterwerb
- Zusätzlich: Online-Aufgaben zur Vertiefung speziell bei Methoden
- Bitte um Rückmeldung wenn zu viel schon bekannt ist in einem Abschnitt!

Teil II

Fundamentals of Testing

Kursübersicht

- Grundlagen des Software Testens (Fundamentals of Software Testing)
- Testen im Entwicklungszyklus (Testing in the Software Life Cycle)
- Statisches Testen (Static Test)
- Dynamisches Testen (Dynamic Analysis - Test Design Techniques)
- Test Management
- Testwerkzeuge (Test Tools)

Überblick Grundlagen des SW Testens

- Motivation und Begriffe (Terms and Motivation)
- Fundamentaler Testprozess (Fundamental Test Process)
- Psychologie des Testens (Testing Psychology)
- Grundprinzipien des Testens (Principles of Testing)
- Ethik des SW-Testens (neu im Syllabus 2010)
- Zusammenfassung

Grundlagen des Software Testens

Motivation

- Bei der Herstellung: Prüfung ob (Teil-)produkte ihre Anforderungen (Requirements) erfüllen
- Anforderungen: je Produkt unterschiedliche Qualitätsanforderungen (inkl Funktion)
- SW ist immateriell $\Rightarrow$ das Endprodukt ist nicht separat prüfbar (keine "optische Prüfung")
- Ziel des Testens: Fehleraufdeckung, dadurch Risikenminimierung
- Art und Umfang des Tests festgelegt als Teil
  - des Vertrags oder
  - eines übergeordneten Standard oder
  - des Firmenstandards

Fehler und Mangel

- nicht anforderungsgerechtes Verhalten ?
  - erfordert Identifikation des erwarteten, korrekten Verhaltens
  - allgemein: Nichterfüllen einer Anforderung
  - $\Rightarrow$ Abweichung des Istverhalten vom Sollverhalten (actual result vs. expected result)
  - Mangel (failure) = wenn gerechtfertigte Nutzererwartung nicht eingehalten
    - Abweichung in der spezifizierten Funktionalität
    - Beeinträchtigung des Verhaltens (zu langsam, schwer zu verwenden) obwohl Funktionalität vorhanden
      $\rightarrow$ Qualitätsanforderungen

Error, Fault, Failure etc.

- Ursachenkette (Causal Chain)
  - Mangel/ Fehlerwirkung (failure) ist das sichtbare Fehlverhalten eines Produkts
  - verursacht durch einen Fehlerzustand (defect, fault) in diesem Produkt
  - als Ergebnis einer Fehlhandlung (error) eines Produktverwenders oder -entwicklers
- Weiter Begriffe in diesem Zusammenhang:
  - Anomalie (anomaly): jede Abweichung vom Spezifizierten; durch Analysen, Reviews, Test, Ausführung aufdeckbar
  - Fehler (defect,flaw): Problem im System oder einer Komponente, die die Funktion beeinträchtigen und die bei Ausführung zu einem failure führen kann
  - Abweichung (deviation): Unterschied zwischen charakteristischem Wert und dem entsprechenden Referenzwert oder Abweichung von erwarteten Lieferwert oder Service (vgl. incident)
  - Fehler (incident): jedes Ereignis, das beim Testen auftritt und eine Untersuchung erfordert [IEEE 1008]
  - Problem (problem) = defect
  - Bug (bug) = defect
  - Fehlermodus (failure mode): Physikalische oder funktionale Manifestation eines fault

Test vs. Debugging

- Fehlerbehebung erfordert Lokalisierung des Fehlerzustands (fault)
- Lokalisierung und Behebung (localization and correction) = debugging
- dabei: Erhöhung der Qualität des Produkts - sofern keine neuen Fehler eingeführt werden
- Testen = Ausführung eines Testobjekts zu seiner Untersuchung
- Verschiedene Testzwecke
  - Ausführung eines Programms um failures zu finden
  - Ausführung eines Programms um Qualität zu messen
  - Ausführung eines Programms zur Erhöhung des Vertrauens
  - Analyse eines Programms oder seiner Dokumentation um Defekte zu verhindern

Verschiedene Begriffe

- Testdaten (test data)
- Planung, Design, Implementierung, Analyse eines Tests ($\rightarrow$ Testmanagement)
- Testprozess (test process)
- Testlauf / Testsuite (test run / suite)
- Testfall (test case) = test conditions, inputs, expected outputs / behaviour
- Testszenarien (test scenarios), auch Kettentest (chain test) = aufeinander aufbauende Testfälle

Testen und Korrektheit

- Kein komplexes (nicht-triviales) System ist frei von Fehlern (faults)
  - oft werden Randbedingungen oder Ausnahmen vergessen oder übersehen
  - trotzdem können Systeme über lange Zeiträume ohne Fehlerwirkungen (faults) laufen
- Vollständige Korrektheit kann durch Test nicht erreicht werden (im Allgemeinen)
  - anders ausgedrückt: das erfolgreiche Durchlaufen aller Tests ist keine Garantie dafür dass keine Fehler (faults) im System sind.

Testen - verschiedene Aspekte

- häufig werden verschiedene Aspekte des Testens vermischt:
  - Testebenen (test level, $\rightarrow$ lifecycle) - Komponenten, Integrations-, System-, Abnahmetest
  - Testziel (test objective) oder Testtyp (test type) - z.B. Lasttest,
  - Testtechniken (test techniques) - z.B. business-proces-based test, boundary-value test
  - Testobjekt (test object) - z.B. GUI Test, Datenbanktest
  - Testperson (test person) - z.B. user acceptance test, Entwicklertest
  - Testausmaß (test extent) - z.B. regression test

Qualität nach ISO/25010 (1)

- SW Qualität soll durch Test erhöht werden
- Qualität wird durch Qualitätskriterien oder auch -attribute definiert
- 2 Kategorien nach ISO 25010:
  - Nutzungsqualität (quality in use)
  - Produktqualität (product quality)
- Generell: nicht alle Qualitätsziele können optimal erreicht werden
  $\Rightarrow$ Priorisierung von Q-Zielen ist daher notwendig

Anmerkung: ISO 25010 plus ISO/IEC 25012 ersetzen ISO/IEC 9126

Qualität nach ISO 25010 (2)

Hauptkriterien und Unterkriterien für Nutungsqualität nach ISO 25010:

- Effektivität (effectiveness)
- Effizienz (efficiency)
- Nutzungszufriedenheit (satisfaction)
  - Nützlichkeit (usefulness)
  - Vertrauen (trust)
  - Vergnügen (pleasure)
  - Komfort (comfort)
- Risikofreiheit (freedom from risk)
- Kontextabdeckung (context coverage)

Qualität nach ISO 25010 (3)

Hauptkriterien und Unterkriterien für Produktqualität nach ISO 25010:

- Funktionelle Eignung (functional sustainability)
  - Funktionale Vollständigkeit (functional completeness) Sind alle spezifizierten Aufgaben und Benutzerziele abgedeckt
  - Funktionale Korrektheit (functional correctness) Werden im Produkt bzw System die richtigen Ergebnisse in erforderlicher Genauigkeit geliefert?
  - Funktionale Angemessenheit (functional appropriateness) Inwieweit werden Aufgaben/Ziele erreicht durch die Funktionen?
- Leistungseffizienz (performance efficiency)
  - Zeitverhalten (timing behaviour)
  - Verbrauchsverhalten (resource utilisation)
  - Kapazität (capacity)
- Kompatibilität (compatibility)
  - Co-Existenz
  - Interoperability
  - ....

Qualität nach ISO 25010 (4)

Hauptkriterien und Unterkriterien für Produktqualität nach ISO 25010 (ctd):

- Benutzungsfreundlichkeit (usability)
  - Verständlichkeit
  - Erlernbarkeit
  - Bedienbarkeit
  - Attraktivität
- Zuverlässigkeit (reliability)
  - Reife (maturity)
  - Verfügbarkeit (availability)
  - Fehlertoleranz (fault tolerance)
  - Wiederherstellbarkeit (recoverability)
  - ....
- Sicherheit (security)
  - Vertraulichkeit
  - Integrität
  - Authenzität
  - ....

Qualität nach ISO 25010 (4)

Hauptkriterien und Unterkriterien für Produktqualität nach ISO 25010 (ctd):

- Änderbarkeit/Wartbarkeit (maintainability)
  - Modularität (modularity)
  - Analysierbarkeit (analysability)
  - Modifizierbarkeit (changeability)
  - Stabilität (stability)
  - Testbarkeit (testability)
  - ....
- Portabilität (portability)
  - Anpassbarkeit (adaptability)
  - Installierbarkeit (installability)
  - Ersetzbarkeit (replaceability)
  - ....

Testaufwand - Allgemeine Erkenntnisse

- vollständiges / erschöpfendes (exhaustive) Testen ist nicht möglich
- im folgenden: Beispiele
  - Parameter mit einfachen Datentypen
  - Komplexität des Kontrollflussgraphen
  - "einfache" Dreiecksbestimmung (Spillner, GI-Vortrag)
- Grundsätze des Testens (nach Spillner)
  - Es ist nicht möglich, ein nicht-triviales Programm vollständig zu testen.
  - Testen ist ein Optimierungsproblem bezüglich der Vollständigkeit des Tests (Tests sind Stichproben).
  - Es muss immer gegen Erwartungsprobleme getestet werden.
  - Jeder Test muss quantifizierbare und erreichbare Ziele haben.
  - Testfälle müssen erlaubte und unerlaubte Eingabedaten vorsehen.
  - Testfälle müssen wiederholbar sein.
  - Testfälle müssen in das Konfigurationsmanagement eingebunden und archiviert

Testaufwand: Beispiel 1

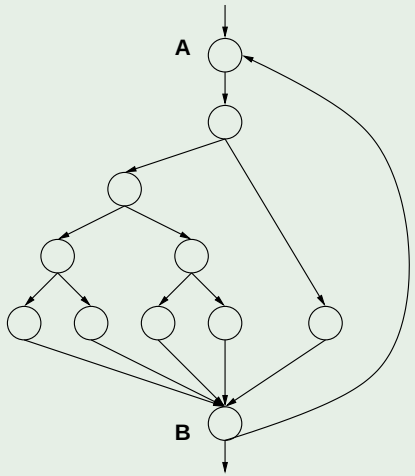
Programm mit drei Integerparametern

- Ein einfaches Programm soll getestet werden, dass drei ganzzahlige Eingabewerte hat. Übrige Randbedingungen haben keinen Einfluss auf das Testobjekt.
- Jeder Eingabewert kann bei 16 Bit Integerzahlen 2^{16} unterschiedliche Werte annehmen.
- Bei drei unabhängigen Eingabewerten ergeben sich $2^{16} * 2^{16} * 2^{16} = 2^{48}$ Kombinationen.
- Jede dieser Kombinationen ist zu testen.
- Wie lange dauert es bei 100.000 Tests pro Sekunde?
  $\Rightarrow$ Gesamtdauer: 89 Jahre

Testaufwand: Beispiel 2

Programm mit Schleife und Verzweigung

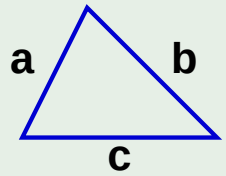
- Ein einfaches Programm soll getestet werden, das aus vier Verzweigungen (IF-Anweisungen) und einer umfassenden Schleife besteht.
- Unter der Annahme, dass die Verzweigungen voneinander unabhängig sind und bei einer Beschränkung der Schleifendurchläufe auf maximal 20, ergibt sich folgende Rechnung: $5^1 + 5^2 + ... + 5^{18} + 5^{19} + 5^{20}$
- Wie viel unterschiedliche Abläufe gibt es?
  $\Rightarrow$ 100 Billiarden unterschiedliche Durchläufe
- Wie lange dauert das Austesten bei fünf Mikrosekunden pro Test
  $\Rightarrow$ Gesamtdauer: 19 Jahre



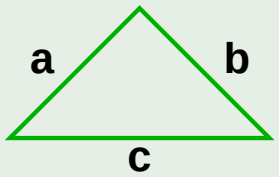
Testaufwand: Beispiel 3 (1)

Testfälle bei der Dreiecksberechnung

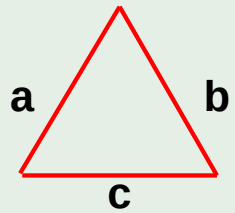
- Ein Programm soll getestet werden, das 3 ganzzahlige positive Werte einliest und anhand der Längen bestimmt welche Art Dreieick vorliegt
- Das Programm gibt aus ob es sich um ein ungleichseitiges, gleichschenkliges oder gleichseitiges Dreieck handelt.
- Wieviele Testfälle braucht man?
  Hinweis: Tests planen für
  - Eingaben, die Dreiecke ergeben
  - Eingaben, die keinem Dreieck entsprechen
  - ungültige Eingaben



a != b != c



a = b !=c



a = b = c

Testaufwand: Beispiel 3 (2)

Testfälle bei der Dreiecksberechnung

- Eingaben, die Dreiecke ergeben
  1. zulässiges ungleichseitiges Dreieck z.B. $(2, 3, 4)$
  2. zulässiges gleichschenkliges Dreieck z.B. $(2, 2, 1)$
  3. zulässiges gleichseitigs Dreieck z.B. $(2, 2, 2)$
  4./5. Permutationen zu 2.; z.B. (1,2,2), (2,1,2)
  6. Test mit maximalen Werten (Minint/Maxint) - alle Kombinationen
  ? Weitere Permutationen von 1.

Testaufwand: Beispiel 3 (3)

Testfälle bei der Dreiecksberechnung

- Eingaben, die keinem Dreieck entsprechen
  7. eine Seite $= 0$, z.B. $(1, 0, 3)$ - plus Permutationen?
  8. Dreieck kollabiert; Summe der kurzen Seiten $=$ lange Seite, z.B. $(1, 2, 3)$

  9./10. Permutationen von 8.

  11. Dreieck kollabiert; Summe der kurzen Seiten $<$ lange Seite, z.B. $(1, 2, 4)$

  12./13. Permutationen von 11.

  14. Seitenlängen ergeben sonst kein Dreieck z.B. 2 oder 3 Eingaben 0

Testaufwand: Beispiel 3 (4)

Testfälle bei der Dreiecksberechnung

- ungültige Eingaben
  15. eine Seite < 0, z.B. $(1, -1, 3)$ - plus Permutationen?
  16. unzulässige Eingabe: keine Ganzzahl
  17. unzulässige Eingabe: Zahl
  18. unzulässige Eingabe: mehr oder weniger als 3 Parameter
  19. weitere

Anmerkung: noch mehr Varianten mit zusätzlicher Unterscheidung spitz-, stumpf-, rechtwinkligen Dreiecken

Testaufwand - Daumenregel

- Man will wissen "wieviel Test nötig ist"
- Allgemein ist der Aufwand nicht einfach zu berechnen - abhängig von Produkt und Einsatzbedingungen
- Daumenregeln:
  - Testaufwand zwischen 25% und 50% des Gesamtaufwands
  - je kritischer das Produkt, desto höher der erwartete Testaufwand
- Abwägung zwischen Risiko im Fall eines Fehlers und Testaufwand
- auch: abhängig von Qualitätszielen
- wichtig dabei: Auswahl angemessener Testprozeduren und Testtechniken
- oft vergessen: Test auf unerwünschte Funktionalität
- Kernproblem: vermeiden von Testfall-Explosion
  - effektiver Einsatz der vorhandenen Ressourcen (Zeit, Kosten, Personal)
- Testmanager verantwortlich für die geplante, effiziente Teststrategie $\Rightarrow$ definierter Testprozess

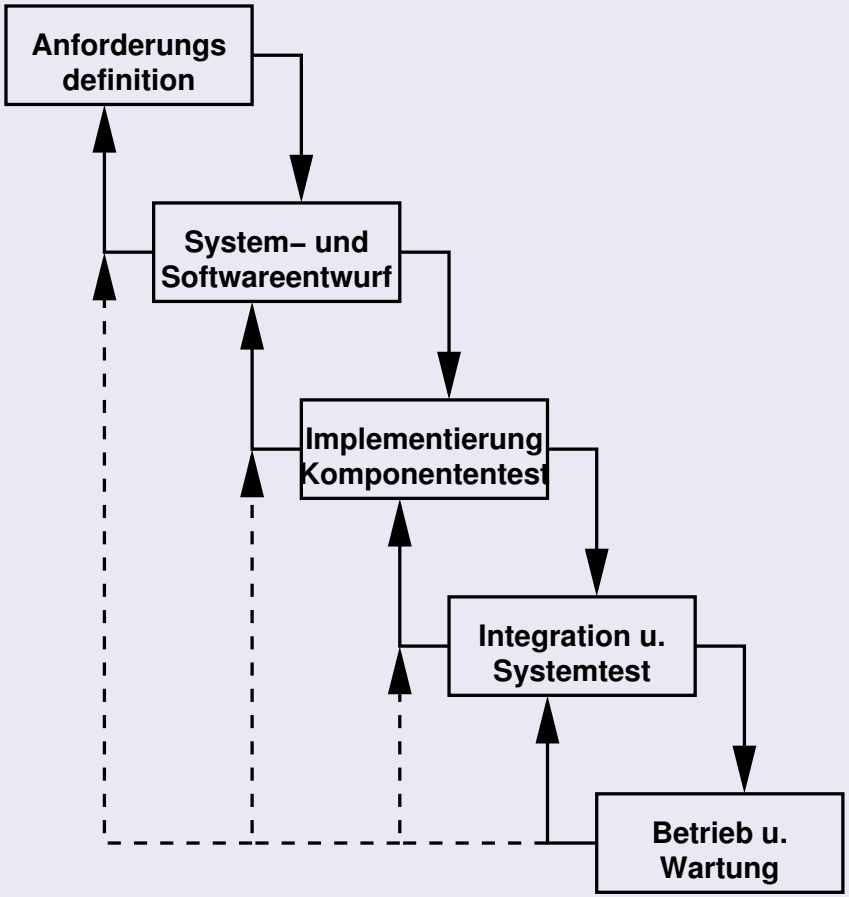
Grundlagen des Software Testens

Prozesse und Testen

- Entwicklungsprozesse (lifecycle processes)
  - regeln Abläufe der Entwicklung
  - definieren Unterprozesse und deren Abhängigkeiten
- Typische Vertreter (ohne Anspruch auf Vollständigkeit)
  - Wasserfallmodell nach Boehm
  - das allgemeine V-Modell nach Boehm
  - W-Modell
  - Vorgehensmodell des Bundes und der Länder, neu: V-Modell XT
  - Spiralmodell
  - aktuell: agile Prozesse, meist iterativ
    - XP nach Beck
    - (R)UP nach Jacobsen
    - Scrum
- typische Festlegungen:
  - Aktivitäten in der zeitlichen Abfolge
  - Phasen mit Meilensteinen
  - Rollen und Dokumente

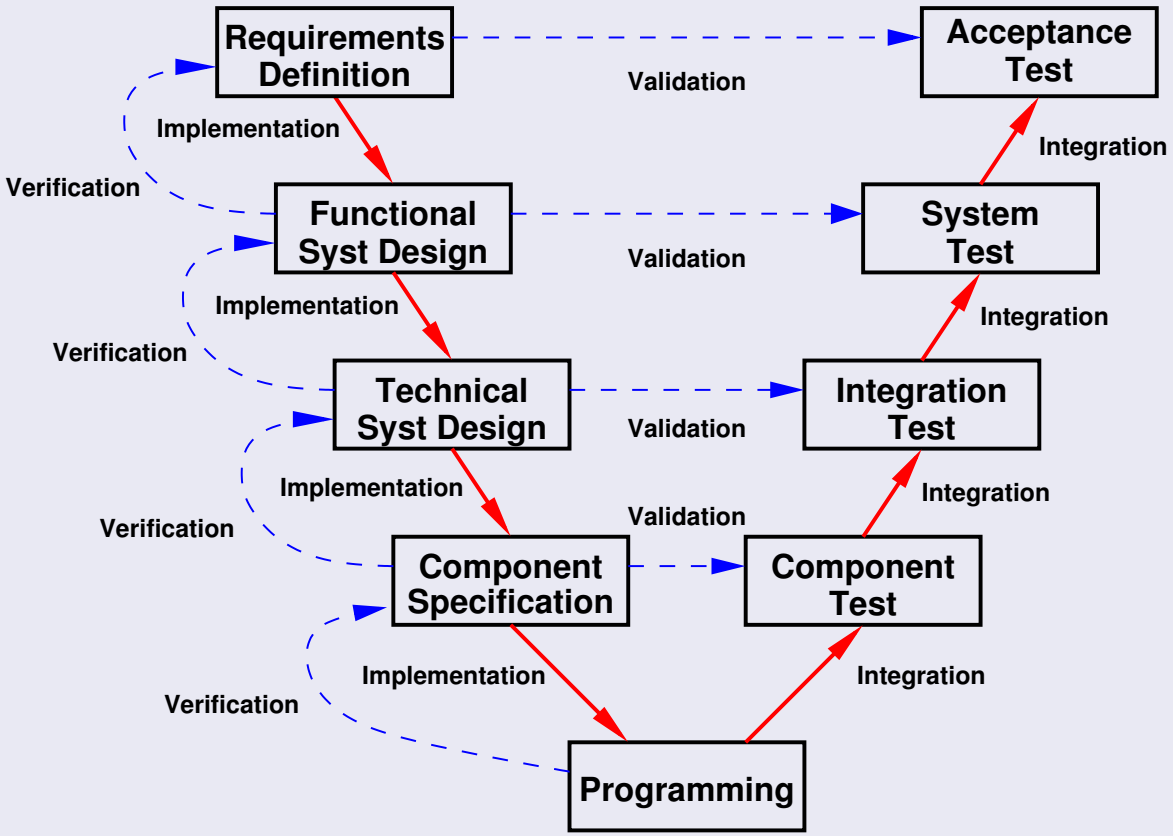
Wasserfallmodell

nach Sommerville, Software Engineering, Abbildung 3.1

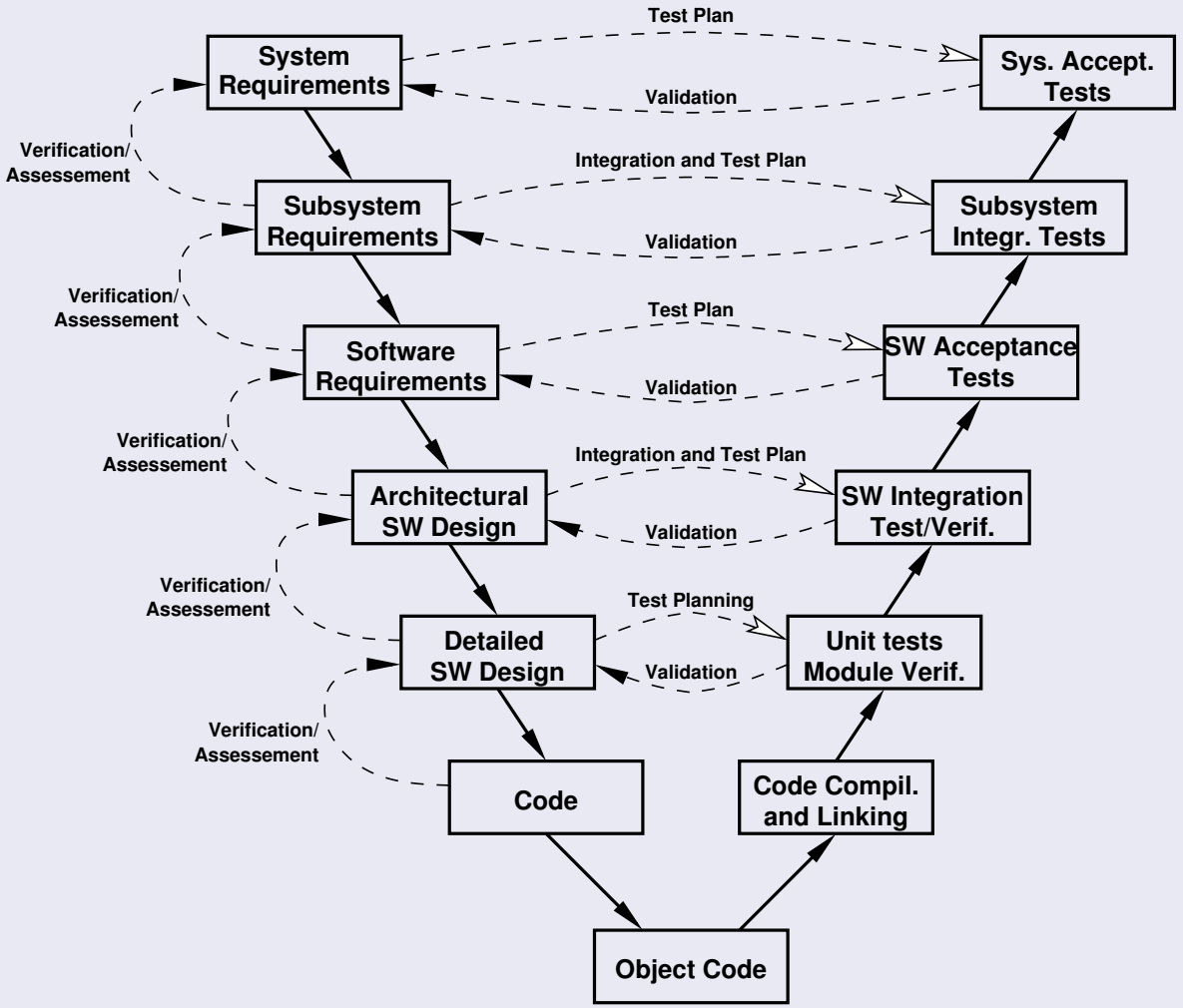


Allgemeines V-Modell

General V–Model (nach Boehm)

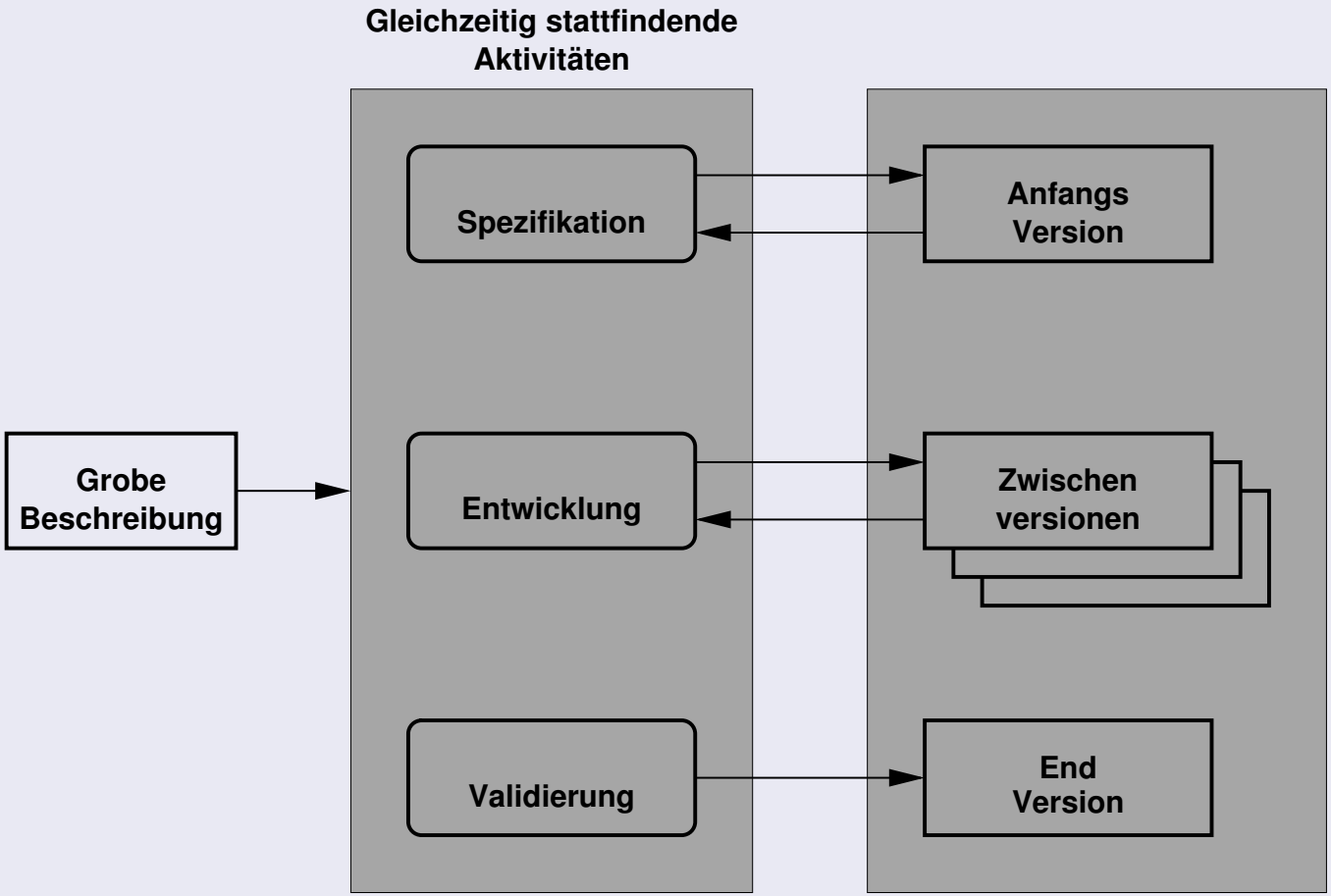


V-Modell des Bundes und der Länder



Inkrementelle SWE

nach Sommerville, Software Engineering, Abbildung 3.2



Testen in den gängigen Vorgehensmodellen

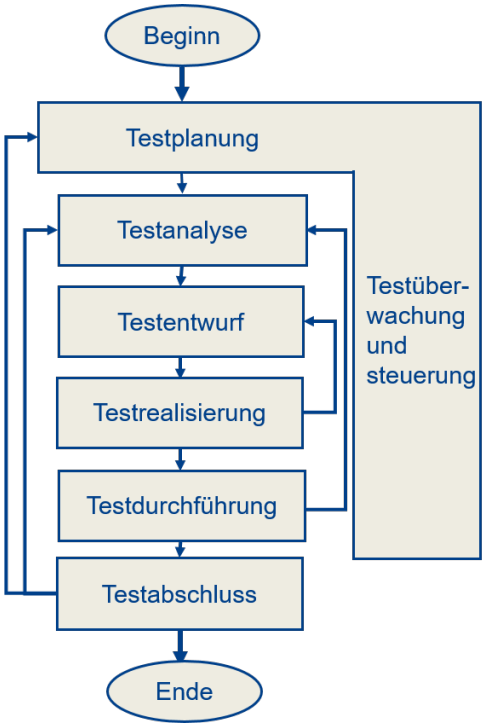
- Wasserfall: Testen als späte Phase
- V-Modell: Testvorbereitung in den Entwicklungsphasen (konstruktive Phasen), Testdurchführung in den Integrationsphasen (testdurchführende Phasen)
- W-Modell: Testprozess parallel zur Entwicklung
- XP - allgemein iterativ: Testen von Iterationen; testen neuer Funktionalität und Regressionstest der Ergebnisse früherer Phasen

Motivation des Fundamentalen Testprozesses

- Ziel: Test in SW-Projekten strukturiert durchführen
- Einordnung in das Lebenszyklusmodell (s. Kapitel 3)
- Definition eines verfeinerten Ablaufplans für Testarbeiten (kleinere Arbeitsschritte)
- Allgemeine Anmerkungen:
  - Aufgaben werden sequentiell angegeben,
  - können aber zeitlich überlappen oder parallel durchgeführt werden
- Der fundamentale Testprozess soll in jeder testrelevanten Phase durchgeführt werden

Fundamentaler Testprozess im Überblick

nach GTB Slides v3.1



Test Planung(Planning)(1)

- Planung der Ressourcen
  - Definition der Aufgabe und Zielsetzung und Einigung darüber
  - Abschätzung der benötigten Ressourcen (Zeit, Ausrüstung, Mitarbeiter)
  - Dokumentation im Testplan (test plan)
  - ev. Trainingsmaßnahmen für Mitarbeiter planen und durchführen
  - Aufsetzen einer Organisationsstruktur für das Testmanagement
  - Steuerung (test control) begleitet alle weiteren Aktivitäten
    - Beobachten der Testaktivitäten; abgleichen mit dem Plan
    - wenn notwendig: Anpassen (update) des Plans
    - Administration und Maintenance von Testprozess, Testinfrastruktur, Testmittel (test ware)
    - Bericht an alle Mitarbeiter, ev. auch automatisch erzeugte Daten
- Festlegen der Test Strategie (test strategy)
  - Ziel: optimale Verteilung der Tests (vgl auch Abschnitt 6.4)
  - Priorisierung der Tests auf Basis von Risikoanalyse
  - Verteilung der Testaktivitäten auf Subsysteme, Komponenten unter Berücksichtigung der Kritikalität
  - auch möglich: unkritische Subsysteme nicht individuell testen
  - Basis: z.B. Erfahrungswerte

Test Planung (Planning)(2)

- Definition der Testintensität für Subsysteme und spezielle Testaspekte
  - Stichwort: Test Coverage (Testüberdeckung) als Testendekritierium (test exit criteria)
  - Oft: Erfüllung der Kundenanforderungen
  - Manchmal: Anforderungen aus Standards
- Priorisierung der Tests
  - Sicherstellen dass kritische Teile des Systems vorrangig getestet werden
  - $\Rightarrow$ ev. Weglassen von Tests weniger kritischer Teile bei fehlender Zeit
- Testwerkzeuge
  - speziell wenn keine Werkzeuge vorhanden: Auswahl und Einführung früh beginnen
  - s. auch Kap 7
  - vorhandene Werkzeuge für das aktuelle Projekt evaluieren
  - Planung des Aufsetzens von Test Harness (test bed) für Subsystem/Komponententests
  - Prüfen der Einsetzbarkeit von Testframeworks wie XUnit

Test Control

- Wie weit ist der Test fortgeschritten?
  - Beobachtung der festgelegten Kriterien (Strategie), Testziele,
  - Testende durch test exit criteria (auch test completion criteria)
  - kontinuierliche Beobachtung auch um weitere Testfälle ergänzen wenn nötig
  - exit criterion für jede verwendete Testtechnik
- Sind weitere Testaufwände gerechtfertigt?
  bzgl Kosten, Aufwand, Zeit
- manchmal sind Testendekriterien nicht erfüllbar
  Anmerkung: nicht-erfüllbare Kriterien können Hinweis auf Inkonsistenzen oder Ungenauigkeiten in den Requirements sein
- Weitere Kriterien zur Beendung des Test
  - failure rate - Gesamtanzahl von Fehlern unter einer bestimmten Anzahl
  - defect detection percentage - neue Fehler einer bestimmten Kategorie unter einem bestimmten Wert
  - in der Praxis: oft Zeit und Kosten - obwohl bekannt ist, dass Fehleraufdeckung Kosten spart

Testanalyse - Analyse der Testbasis

- Testbasis (test basis) = Spezifikation dessen was getestet werden soll
- Untersuchung der Testbasis auf Testbarkeit (testability)
- erfordert u.U. Überarbeitung der Anforderungen
- requirements + expected Behaviour + structure ⇒ precondition + requirements für Testfalldesign (test case design)
- All dieses führt zur Verfeinerung der Teststrategie im Testplan inkl Festlegung der Testtechniken

Testentwurf - Definition logischer und konkreter Testfälle

- Definition der Testfälle (test cases) in zwei Schritten
  1. Definition logischer Testfälle (logical test cases)
     - Festlegen unterschiedlicher Testfälle abgeleitet aus Testbasis
     - z.B. bei Blackboxtest identifizieren unterschiedlicher Fälle in den Requirements
     - z.B. bei Whiteboxtest Analyse des Codes zum Finden verschiedener Ausführungsmöglichkeiten
     - Abhängig vom Level der Testprozessdurchführung
     - Bereiche von Eingabedaten und abstrakte Sollwerte
     - Sollwerte oft als Testorakel realisiert
  2. Festlegung konkreter Testfälle (concrete test cases) je logischem Testfall
     - Precondition und Umgebungsbedingungen (Randbedingungen) für den Test
     - Eingabedaten konkret
     - Sollwerte konkret
     - wenn relevant: konkrete Postcondition
- beim explorativem Testen auch 2. vor 1.

Beispiel logische und konkrete Testfälle

Gegeben der folgende Spezifikationstext:

> *Mitarbeiter mit einer Firmenzugehörigkeit von mehr als 3 Jahren erhalten eine Weihnachtsgratifikation von 50% ihres Monatsgehalts. Mitarbeiter mit einer Zugehörigkeit von mehr als 5 Jahren erhalten 75%. Bei einer Zugehörigkeit von mehr als 8 Jahren wird ein volles Monatsgehalt als Gratifikation gezahlt.*

Logische Testfälle:

test case no	input x (affil in years)	expected result (bonus in %)
1	$x \leq 3$	0
2	$3 < x \leq 5$	50
3	$5 < x \leq 8$	75
4	$8 < x$	100

Konkrete Testfälle:

test case no	input x (affiliation in years)	expected result (bonus in %)
1	2	0
2	4	50
3	6	75
4	13	100

Test Implementierung

- Überführung von Testbedingungen und logischen Testfällen in konkrete Testfälle
- Aufsetzen der Testumgebung, Durchführung und Logging
- Bei der Ausführung berücksichtigen:
  - in der Strategie definierten Prioritäten
  - detaillierte Beschreibungen der Testfälle (Precond, Rahmenbedingungen etc.)
  - Gruppierung von Testfällen in Suiten und Szenarien
  - rechtzeitige Definition und Implementierung von test harness, test driver, simulator
  - Sicherstellung der korrekten Funktion der Testumgebung
  - Testausführung mit laufendem Testling manuell oder automatisch
  - Empfehlung: zuerst Hauptfunktionalität testen - bei Abweichungen: Stop!
    $\rightarrow$ sollte in der Teststrategie festgehalten werden

Test Ausführung

Generelle Empfehlungen zur Testausführung

- kein Test ohne Protokoll - sonst ist er wertlos
- Wiederholbarkeit (reproducability) ist essentiell - dazu müssen Testumgebung, Testdaten, Testlogs dokumentiert sein ($\rightarrow$ Configuration Management)
- Abweichungen im Test entdeckt?
  - kann ein gefundener Fehlzustand sein oder
  - ein Fehler im Test oder
  - Problem in der Testumgebung
  - ein Fehler in der Testbasis

  Notwendig:
  - Analyse der Fehler auf Basis der Logs und weiterer Info
- Im Verlauf der Ausführung: Testcoverage messen (wie auch immer definiert)
- Korrektur eines Fehlerzustands kann zu neuen Fehlern führen (maskiert oder neu eingeführt)
  $\Rightarrow$ Retests durchführen und neue Testfälle
- generell auch hier: kritische Test zuerst(risk-based testing)

Testabschluss

- Testbericht
- Erfahrungen auswerten und festhalten – oft vernachlässigt
  - Notwendig im Sinne des continuous improvement
  - Lernen aus Erfahrungen - Ursachen für Abweichungen zwischen Plan und Realität
  - Wann wurde das System freigegeben (system release)?
  - Wann wurde der Test beendet?
  - Wann wurden Milestones erreicht oder Wartungsversion abgeschlossen?
- Evaluierung z.B. auf Basis von Fragen wie
  - Welche geplanten Ergebnisse wurden erreicht?
  - Welche unerwarteten Ereignisse sind aufgetreten (ev. mit Ursachen)
  - gibt es offene Änderungsanforderungen (change requests)? Warum nicht implementiert?
  - Wie gut war die Nutzerakzeptanz nach Auslieferung (deployment) des Systems?
- Prozessevaluierung!
- Archivierung der Gesamten Testumgebung inklusive Resultaten für Maintenance Phase

Grundlagen des Software Testens

Psychologie des Testens (1)

Einige grundlegende Erkenntnisse

- Leute machen Fehler, geben es aber nicht gerne zu.
  Tester decken aber die Fehler von Leuten auf
- SW Entwicklung wird als konstruktiv empfunden,Testen als destruktiv.
  Aber: Testen erfordert ein hohes Maß an Kreativität und ist eine intellektuelle Herausforderung (Myers)

Psychologie des Testens (2)

- Entwicklertests
  - Pro
    - Entwickler kennen ihr Produkt (und seine Spezifikation?)
    - frühes Finden von Fehlern (auf Komponentenebene)
  - Contra
    - oft fehlt der genügende Abstand zum eigenen Produkt
    - Entwickler sind zu optimistisch
    - Betriebsblindheit und unsystematisches Testen verhindern Fehleridentifikation
    - Wiederholen der Fehler bei Implementierung/Design
- Unabhängige Testteams
  - gefordert in einigen Standards - für bestimmte Risikolevel oder Testebenen
  - Pro
    - betrachten Testling ohne Voreingenommenheit
    - machen oft nicht die gleichen Fehlannahmen wie Entwickler
    - besseres Wissen über systematische Tests, Testtechniken etc.
  - Contra
    - z.T. hoher Einarbeitungsaufwand in den Testling und seine Spek
    - auf den unteren Testebenen existieren zT. keine vernünftigen Spezifikationen
    - Tester werden nicht als Teil des Teams

Psychologie des Testens (3)

- Potentielle Konflikte zwischen Testern und Entwicklern
  - Tester als "die Bösen, die alles kaputt machen"
  - Tester werden nicht als Teil des Teams gesehen / akzeptiert (besonders bei unabhängigen Testern)
  - wichtig: Festlegung, was ein Fehler / eine Abweichung sein soll und wie sie zu berichten sind – auch bei Entwicklertests
  - Ziel des Projekt- und Testmanagement sollte ein wechselseitiges Verständnis zwischen Entwicklern und Testern sein (auch auf der Management-Ebene!)

Grundlagen des Software Testens

7 Prinzipien des Testens (Principles of Testing)

1. Testing shows the presence of defects
   not their absence!
2. Exhaustive testing is not possible
   außer bei trivialen Testobjekten
3. Testing activities should start as early as possible
   spät erkannte Fehler sind teuer
4. Defects tend to cluster together
   Fehler sind nicht gleichmäßig über die SW verteilt
5. The Pesticide Paradox (Zunehmende Testresistenz)
   Testfälle regelmäßig prüfen, erweitern, modifizieren
6. Testing is context dependent
   Tests anpassen an Einsatzumgebung und Anwendungsrandbedingungen
7. The fallacy of assuming that no failures means a useful system
   Spezieller Bereich: Usability Tests; Nutzeranforderungen mit Prototypen ausloten

Grundlagen des Software Testens

Code of Ethics

- basierend auf Anforderungen von ACM und IEEE code of ethics for engineers
- Certified SW Tester hat Verantwortung gegenüber
  - public: consistent with public interests
  - client and employer: best interest vertreten, consistent with public interest
  - product: highest professional standard possible
  - judgement: maintain integrity, independence in professional judgement
  - management: test manager and test leader subscribe to and promote ethical approach
  - profession: shall advance integrity and reputation consistent with public interest
  - colleagues: be fair and supportive of test colleagues, promote cooperation with SW developers
  - self: participate in life-long learning, promote ethical approach

Grundlagen des Software Testens

Zusammenfassung Kapitel SW Test Grundlagen

- Technische Begriffe im Bereich Testen werden oft verschiedenen definiert und unsauber verwendet.
  Hier: Festlegung der Begriffe,
- Bezug hergestellt zwischen Testen und Qualitätsbegriffen
- Definition des fundamentalen Testprozesses
- Begriffe: test case, precondition, actual vs expected value/behaviour, test oracle
  im Zusammenhang mit Testspezifikation
- die menschliche Seite des Testens
- 7 Testprinzipien

Teil III

Testen im Entwicklungszyklus

Kursübersicht

- Grundlagen des Software Testens (Fundamentals of Software Testing)
- Testen im Entwicklungszyklus (Testing in the Software Life Cycle)
- Statisches Testen (Static Test)
- Dynamisches Testen (Dynamic Analysis - Test Design Techniques)
- Test Management
- Testwerkzeuge (Test Tools)

Überblick Testen im SW Lifecycle

- V-Modell - allgemein
- Komponententest (Component Test)
- Integrationstest (Integration Test)
- Systemtest (System Test)
- Abnahmetest (Acceptance Test)
- Test neuer Produkt-Versionen
- Allgemeine Testtypen
- Zusammenfassung

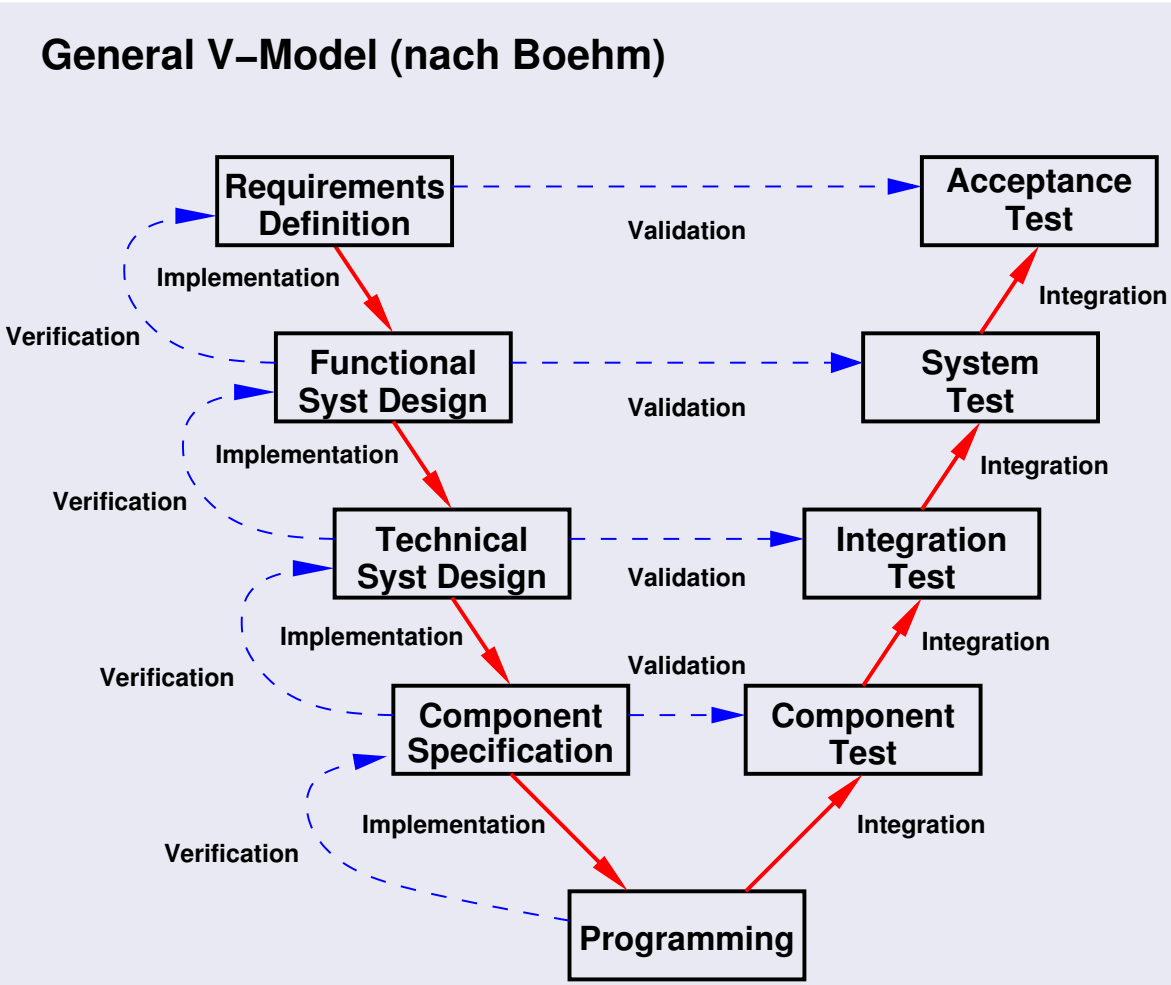
In diesem Kapitel

- Detaillierte Definition der Testaktivitäten anhand des allgemeinen V-Modells
- Definition der Teststufen Komponententest, Integrationstest, Systemtest und Abnahmetest bzgl.
  - speziellen Begriffen (terms)
  - Testobjekten (objects)
  - Testumgebung (environment)
  - Testzielen (objectives)
  - Teststrategie (strategy)
- Abgrenzung zum Versionstest
- Trennung Testebenen von Testtypen
- Grundlegende Unterscheidung
  - Validation = Bauen wir das richtige System?
    Abgleich mit der jeweiligen Ebenen-Spezifikation
    Erfüllt das Produkt seine Aufgaben?
  - Verifikation = Bauen wir das System richtig?
    Ist das Ergebnis eine Entwicklungsphase korrekt und vollständig zum Input (Dokumenten des vorherigen Levels)
- Test zu Verifikation und Validation, Validationsaspekt stärker in den oberen Ebenen

Muss sinngemäß auf andere Lebenszyklusmodelle übertragen werden

Testen im Entwicklungszyklus

Allgemeines V-Modell



Testen im Entwicklungszyklus

Komponenten-Test Terms

- Verantwortung der Entwickler selbst; basiert meist auf intuitivem Verständnis - oft als Entwicklertest bezeichnet (Nicht CT!)
- Andere Bezeichnungen (je nach Programmiersprache): unit test, module test, program test, class test
- Hier: component oder unit und component testing

Komponenten-Test objects

- Komponenten die getestet werden können
  - Einzelne Funktionen / Prozeduren / Methoden (besonders wenn komplexe Algorithmen darin implementiert)
  - Klassen mit Attributen und Methoden (als Objekt) - Test des Datentyps oder der Zustandsmaschine bis in die Details
    ! Test aller Operationen, Manipulation aller Attribute, alle Methoden in allen Zuständen - als Optimum
  - Zusammengesetzte Komponenten aus mehreren Objekten mit definierter Schnittstelle in die gemeinsame Umgebung aber Test bzgl. interner Eigenschaften
- Wichtig: Test in Isolation – u.U. mit Stubs

Komponenten-Test environment

- Testen eng mit Entwicklungaktivitäten verknüpft
- eher hoher Aufwand für Entwicklung von Testumgebung, speziell: Treiber
- Erstellen der Umgebung erfordert tiefe Kenntnis der Komponente, Treiberprogrammierung erfordert Programmierkenntnisse auf dem selben Niveau wie Entwicklung
- component test != debugging!! (vgl Def in Kapitel 2)
- Heute üblich: Verwendung von Komponententestframeworks (XUnit)

Komponenten-Tests - objectives

- Ziel: Sicherstellen der korrekten und vollständigen Umsetzung der Funktionalität (functional testing)
- Funktionality = input/output behaviour
- Aber: systematisches Vorgehen zur Spezifikation der Testfälle
- Ziel: Defect Testing - Fehler in den Komponenten finden
- Typische Fehler (defects) die hier gefunden werden
  - wrong calculation
  - missing or wrongly chosen program paths
  - forgotten special / exceptional cases
  - possible misuse (not according to spec)
- weitere Ziele
  - robustness - using invalid input data negative tests
  - efficiency - Ressourcennutzung: Zeitverhalten, memory usage, CPU-Nutzung, Network access etc.
  - maintainability - code structure, modularity, quality of comments, adherence to standards, coding rules, quality of documentation

Anmerkungen zu Negativtests

- es gibt mindestens soviele vernünfitge Negativ- wie Positivtests
- Testtreiber müssen so erweitert werden, dass die Ausnahmebehandlung mitgetestet werden kann
- Exception handling ist im Schnitt 50% des Gesamtcodes $\Rightarrow$ Robustness testing wird teuer

Komponenten-Tests - strategy

- beim Test direkter Zugang zum Code ⇒ white box testing
- Zugang zu allen Attributen des Codes
- Verwendung von Hilfswerkzeugen (z.B. debugger) auch bei Testdurchführung
- Möglichkeiten zur Manipulation des internen Zustands
- praktisch: Komponententest oft als black box test (! Verlust von Coverage)
- Automatisierungspotential (XUnit, Coveragemessung)
- Alternativen: test-first programming, test-driven development

Testen im Entwicklungszyklus

Integrationstest - terms

- Grundannahme: die Komponenten sind lokal getestet, Defekte beseitigt
- Integration = Zusammenfügen mehrerer Komponenten zu einer strukturierten Einheit
- Integrationstest = Sicherstellen, dass die Komponenten richtig zusammenarbeiten
- Prüfen auf Interface-Probleme (collaboration and interoperability problems)
- Speziell: Schnittstellen zur Systemumgebung testen - system integration test
  Hier: potentielle Änderungen berücksichtigen - kein Einfluss möglich

Integrationsstrategien

- Bottom-up Integration: erst elemantare Komponenten, schrittweise Integration
- Top-Down Integration: erst Top-Level Komponenten, dann schrittweise untere Ebenen (nur sinnvoll bei Schichtarchitektur)
- Big-Bang Integration: Alles einzeln testen, dann alles gleichzeitig zusammenfügen
  um jeden Preis vermeiden, Probleme mit Fehlerlokalisierung vorprogrammiert
- Ad-hoc Integration: Integration von Komponenten sobald sie fertig sind
- Backbone Integration: Skelett/Backbone erstellen, dann nach und nach weitere Komponenten zufügen

beeinflussen Teststrategie (s. u.)

Integrationstest - objects

- units bestehend aus mehreren Komponenten, z.T. Schrittweise erweiterung oder Integration zu Subsystemen
- Berücksichtigung von vorhandenen Komponenten, of-the-shelf products
  oft: keine konkrete Info über Komponententests

Interationstest - Environment

- Wie beim Komponententest, aber abhängig von Integrationsstrategie: Testtreiber
- Wiederverwendung der Testtreiber der Komponententests
  $\Rightarrow$ Wiederverwendung (bei der Planung berücksichtigen)
- Problem: schlechte Komponententesttreiber oder Bedienschnittstellen
- zusätzlich nötig: monitors - Werkzeuge, die Datentransfer zwischen Komponenten lesen und loggen
  z.B. Monitore für network protocols; z.T. kommerziell verfügbar

Integrationstest - objectives

- Aufdecken von interface and cooperation problems
  - inkompatible Schnittstellenformate
  - fehlende Dateien
  - Komponentenrealisierung anders als im Design festgelegt
  - Probleme, die sich aus dem Zusammenspiel ergeben (schwer zu finden)

    notwendig: dynamischer Test!
- Unterscheidung unterschiedlicher fault types:
  - component transmits syntactically wrong or no data; receiving component crashes or can not operate (functional fault, incompatible interface formats, protocol fault)
  - communication works, but involved components interpret data in a different way (functional fault, contradicting or misinterpreted spec)
  - data transmitted correctly, but at wrong time, is send/received too late (timing problems) or intervals between transmissions are too short (throughput, load or capacity problems)
- Motivation: diese Fehlertypen werden beim Komponententest nicht systematisch gefunden
- Außerdem: testen nicht-funktionaler Eigenschaften - mit Bedacht (performance, capacity of interfaces)

Kann der vorangehende Komponententest weggelassen werden

Ja, aber mit gravierenden Nachteilen

- Komponenten werden nur implizit getestet - kein systematischer Zugang zu allen Facetten mehr möglich
- wegen des indirekten Zugangs kein systematisches Triggern von Failures; viele faults werden so nicht gefunden
- bei auftretenden failure kann die Ursache nur schwer oder gar nicht eindeutig in den Komponenten lokalisiert werden

Die zusätzlichen Kosten für umständliche Fehlerlokalisierung und nicht gefundene Fehler können die Kosten für den Komponententest leicht aufwiegen

Integrationstest - strategy

- Grundlage:
  - Komponenten werden zu unterschiedlicher Zeit fertig
    auch abhängig von der Architektur
  - Entwicklungsreihenfolge sollte mit Testplan abgestimmt sein, damit frühe Tests stattfinden können
    sonst: nur ad-hoc Integration und Test möglich $\Rightarrow$ hohe Kosten für Stubs
  - Besser: Entwicklung und Test unter Berücksichtigung der Kritikalität
- Damit ergeben sich als Randbedingungen (constraints) für die Integrationstests
  - system architecture
  - project plan
  - test plan

Vor- und Nachteile unterschiedlicher Integrationsstrategien

- Bottom-up integration
  - Pro: keine stubs benötigt
  - Contra: higher-level components (in einer Schichtenarchitektur) müssen simuliert werden (Testtreiber)
- Top-down integration
  - Pro: keine oder nur einfache Testtreiber (verwenden der higher-level components)
  - Contra: stubs für lower-level components benötigt - ev. hohe Kosten
- Ad-hoc integration
  - Pro: Zeitersparnis, Integrationstest sobald Komponente implementiert
  - Contra: Stubs und Testtreiber benötigt (Aufwand/Kosten)
- Backbone integration
  - Pro: Integration in beliebiger Reihenfolge
  - Contra: Erstellung des backbone / skeleton ev. mit hohem Aufwand
- Big-Bang Integration
  - Pro: keine
  - Contra: absolutes Chaos, Fehlerlokalisierung nur sehr aufwändig, Zeit besser für vernünftige Integrationsplanung nutzen

Testen im Entwicklungszyklus

Systemtest - terms

- system test = Testen des voll integrierten Systems gegen die requirements
- Fokus: Perspektive der Kunden und Endnutzer
- Einige Funktionen sind erst im Gesamtsystem sichtbar und müssen hier getestet werden.

Systemtest - objects and environment

- Object: System as a whole
- in Testumgebung, die Einsatzumgebung weitgehend entspricht
- Idealerweise: keine stubs oder Treiber, sondern reale Systemumgebung (HW+SW) auf der Testplattform
- zu vermeiden: system test in customers operational environment
  - Schaden durch Fehlerwirkungen am Kundensystem vermeiden (z.B. Datenverlust im Produktivbereich)
  - keine oder wenig Kontrolle über Parameter und Konfigurationen dh. nicht alles ist testbar sondern abhängig von realer Einsatzumgebung
  - u.U. keine Reproduzierbarkeit der Tests
- Aufwand für Erstellung der Systemtestumgebung nicht unterschätzen!!

Systemtest - objectives

- validate whether complete system fulfills functional and non-functional requirements
- detect failures from incorrect, incomplete, inconsistent implementation of requirements
- identify undocumented or forgotten requirements

Systemtest - strategy

- Ableiten aus der Kritikalität/Priorität einzelner Anforderungen (s. Testplan)
- lediglich die Priorisierung und Intensität der Tests kann hier noch gesteuert werden.

Systemtest - Potentielle Probleme

- unclear requirements
  - unvollständig
  - nur in den Köpfen der Entwickler festgehalten
- missed decisions
  - bei der Entwicklung geänderte Spec aber nicht dokumentiert
  - fehlende reviews oder explizite releases von Entscheidungen
  - Hoher Aufwand, die Änderungen während des Tests zu identifizieren
- Konsequenz: Projekte schlagen fehl
  - ohne Requirements keine sinnvoller Test möglich
  - hohes Risiko von Kundenunzufriedenheit
  - Testen kann dann nur als Indikator für das Scheitern des Projektes dienen

Testen im Entwicklungszyklus

Akzeptanztest - terms

- Akzeptanztest = Test des Gesamtsystems oder von Teilen in seiner Zielumgebung gegen die Nutzer/Customer-Anforderungen
- Fokus: Customer perspective and judgement
- unter Einbeziehung / in Anwesenheit des Kunden
- auch möglich: der Kunde ist vollständig für den Test verantwortlich
- Ausführung auf verschiedenen Ebenen:
  - COTS Produkte bei Installation oder Integration
  - Acceptance der usability einer Komponente au Komponentenebene
  - Acceptance test einer neuen Funktionalität auch vor dem Systemtest (ev. mit Prototypen)
- Testumfang abhängig vom speziellen System; höher bei Individual-SW
  Minimum: Installation und einige repräsentative Anwendungsszenarien (use cases)
- alpha-Test = Test von Vorabversionen beim Hersteller
- beta-Test = Test von Vorabversionen beim Kunden

Akzeptanztest - objects

- s.o: Komponenten, Subsysteme, Gesamtsystem

Akzeptanztest - objectives (1)

- Test ob der Vertrag erfüllt
  - Akzeptanzkriterien im Vertrag festgehalten
  - Speziell: Auflagen aus übergeordneten Standards(Regierungs, Sicherheits-)
  - Nachweis, dass diese Anforderungen erfüllt sind
  - Wichtig: Review der Akzeptanzkriterien durch Kunden und formale Einigung darauf
  - Meist in der Kunden oder Endnutzer-Umgebung
  - Ansonsten: wie Systemtest
- User acceptance
  - wichtig wenn Kunde nicht Endnutzer ist
  - alle Endnutzergruppen (subset der stakeholder) müssen einbezogen werden
  - Abweisung des Systems auch bei vollständiger funktionaler Korrektheit möglich (vgl Usability Requirements)
  - Empfehlung: Endnutzer mit Prototypen konfrontieren und Feedback frühzeitig im Design berücksichtigen

Akzeptanztest - objectives (2)

- operational acceptance
  - Akzeptanz durch Systemadministratoren
  - auch: Test von
    - backup/restore cycle,
    - user management,
    - maintenance tasks,
    - security
- Feldtest (alpha / beta test)
  - Motivation: System für viele operationelle Umgebungen, hoher Aufwand für alle relevanten Testumgebungen
  - Nach Systemtest
  - Identifizieren und Entfernen der Umgebungseinflüsse auf das System
  - Testen von stabilen Vorabversionen z.T. mit spezifizierten Anwendungsszenarien oder mit Teilfunktionalität
  - Feedback an Hersteller

Akzeptanztest - environment

- Test ob der Vertrag erfüllt
  Meist in der Kunden oder Endnutzer-Umgebung
- User acceptance beim Endnutzer oder in spezieller Testumgebung
- operational acceptance
- in Zielumgebung
- Feldtest (alpha / beta test)
- alpha-Test beim Hersteller z.B. in Systemtest-Umgebung
- beta-Test beim Kunden

Testen im Entwicklungszyklus

Motivation

- Bisher: Entwicklungsbegleitendes Testen
- Nach Auslieferung und Installation wird SW aber gewartet oder weiterentwickelt
- Es entstehen Versionen des Produktes
- Hier: Testen in der Maintenance- und Weiterentwicklungsphase
- Immer notwendig: gutes Konfigurationsmanagement

SW Maintenance und Testen

- Grundannahmen:
  - SW altert nicht
  - Keine Failures durch Materialerschöpfung o.ä., sondern design faults
- $\Rightarrow$ SW Maintenance = adaptive maintenance + corrective maintenance
- Testen bei SW Maintenance:
  - Nachweis dass Änderungen erfolgreich,
  - Rest-Funktionalität erhalten,
  - Fehler behoben
- Probleme dabei
  - alte und neue Spec nicht aktuell oder nicht vorhanden
  - Speziell bei legacy systems
- Typische Ursachen für Änderungen nach Auslieferung
  - System wird unter Bedingungen verwendet, für die es nicht entwickelt wurde (not predictable, not planned)
  - Neue Anforderungen des Kunden
  - Spezialfälle wurden nicht vorhergesehen
  - Crashes, die selten passieren oder nur nach langer Aktivphase des Systems (oft externe Einflüsse)

Teststrategie in der Maintenance

- alles neue oder veränderte muss getestet werden
- Rest des Systems durch Regressionstest absichern - Ausschluss von Seiteneffekten
- Testen auch notwendig, wenn sich nicht das System sondern die Umgebung ändert ! (z.B. migration to new platform)
- Testing for retirement: Test der Datenarchivierung bzw der Datenmigration zum Nachfolgesystem - ev. auch Back-to-Back-Test

New SW Releases und Testen

- Motivation: geplante Veränderung und Erweiterung eines Systems
  auch: Versionen für weitere Plattformen
- Auch: langlebige Produkte mit regelmäßigen neuen Releases
  koordinieren mit Maintenance
- Neue Releases = Durchlauf aller Projektphasen
  (iterative Entwicklung)
- Testen bei Releases
  - wenn möglich: Durchlauf aller Testlevel – analog zu Maintenance Testen

Inkrementelle Entwicklung und Testen

- Inkrementelle Entwicklung = Serie kleiner Entwicklungsschritte inklusive Auslieferung
- Motivation: Absicherung des Systems bzgl Nutzbarkeit für Endnutzer in Schritten
  auch: Kontinuierliches Review der Anforderungen
- Verwendet bei Agiler SW-Entwicklung wie XP, Scrum und RUP, RAD, Spiralmodell
- Testen bei Inkrementeller Entwicklung
  - Continuus integration testing
  - regression testing (immer alle Test alter Inkremente laufen lassen)
- Kann als wiederholtes V-Modell gesehen werden, bei dem beim Testen von Folgeinkrementen die vorherigen Tests wiederverwendet werden

Testen im Entwicklungszyklus

Einordnung

- Bisher: Testebenen - entsprechend SW Entwicklung
- Jetzt andere Perspektive: Testtypen, die in jeder der Testebenen zur Anwendung kommen
  - functional testing
  - nonfunctional testing
  - testing of SW structure
  - testing related to change

Functional Testing

- functional testing - verification of system input/output behaviour
- design of test cases mit black box testing methods, Basis: functional req.
- Functional Req
  - spezifizieren das Systemverhalten - vgl ISO/IEC Q-Kriterien
  - dokumentiert in einem requirements management system (s. 7.1) oder als Textdokument
- pro Requirement mindestens 1 Test, normalerweise mehr
- Review der Testspezifikation bzgl Requirements Closeout
- Durchlauf der Testfälle zu einem Req erfolgreich $\Rightarrow$ Req ist validiert
- Meist für System- und Abnahmetest
- auch als business-process-based testing oder use-case-oriented testing (s. 5.1)
  eher Szenarienbasiert, nicht direkt auf einzelne Anforderungen

Nonfunctional Testing

- Nicht-funktionale Anforderungen (vgl ISO/IEC 9126 Q-Kriterien) beschreiben Attribute des Systems oder der Systemfunktion oder Qualität von Funktionen
- Auch Eigenschaften, die durch dynamische Tests nicht oder nur indirekt geprüft werden können
- Problem bei impliziten nichtfunktionalen Anforderungen - müssen für Test sichtbar gemacht werden
- auch Probleme bei unterspezifizierten Anforderungen (timing, throughput)
- Teststrategie für (einigen) nichtfunktionale Anforderungen:
  - Wiederverwendung und ev. Modifikation funktionaler Test
  - Dabei Messen der nicht-funktionalen Kriterien
- einige Eigenschaften nur durch statische Tests (z.B. Wartbarkeit)

Nonfunctional Testing - Aspekte

- zu berücksichtigen beim Systemtest (nach Myers):
  - Load test: messen des Verhaltens bei steigender Systemlast
  - Performance test: messen der Verarbeitunggeschwindigkeit und Antwortzeiten (auch mit steigender Last)
  - Stress test: Beobachtung des Systemverhaltens bei Überlast (overload)
  - Testing of security: gegen unauthorized access, denial of service attacks
  - Stability / reliability test: Verhalten im Dauerbetrieb - auch mean time between failures, failure rate for user profiles
  - Robustness test: Reaktion auf Bedienfehler (operating error), HW Failure, programming errors (fault injection) sowie Test von exception handling and recovery
  - Testing of compatibility and data conversion: Test in der Einsatzumgebung, speziell import/export von Daten
  - Testing of different configurations: z.B. OS, User interfaces, HW platforms – oft als back-to-back test
  - Usability test: Untersuchung der Erlernbarkeit, intuitiven Verwendung, Verständlichkeit der Ausgaben etc. (s. ISO 9241, ISO/IEC 9126)
  - Checking of documentation: gegen tatsächliches Systemverhalten - speziell user manual und GUI
  - Checking of maintainability: Bewertung der Verständlichkeit von System Dokumentation; Prüfen auf Modularität

Testing the SW structure

- white box test (s. 5.2)
- Testfälle ableiten aus interner code structure oder Architektur oder abstract models
- Ziel: alle Strukturelemente abdecken durch Tests
- Verwendung meist im Komponententest, aber auch Integrations-, System-, Akzeptanztest

Testing related to Changes and Regression Testing

- Nach Änderung, Fehlerbehebung:
  - retest der geänderten Anteile und
  - regression tests des Restsystems zur Vermeidung ungewollter Seiteneffekte
- auf allen Testebenen, für alle Testtypen
- Hier großer Vorteil von Test(durchführungs)-Automatisierung
- Festlegung der Testintensität für Regressionstests notwendig
  - defect retest, confirmation test: rerun all test that detected faults
  - testing altered functionality: Test der veränderten Programmteile
  - testing new functionality: Test der neuen Programmteile
  - complete regression testing: Test des Gesamtsystems
- in der Praxis: selten voller Regressiontest - Strategie zur Testauswahl:
  - nach Priorität im Testplan
  - Weglassen von Spezialfällen der funkionalen Tests
  - Test beschränkt auf wenige Konfigurationen
  - Beschränkung auf relevante Subsysteme oder relevante Testlevel
  - weitere Empfehlungen in der Literatur baiserend auf Erfahrungen

Testen im Entwicklungszyklus

Zusammenfassung Testen im SW Lebenszyklus

- Testlevel basierend auf Entwicklungsstufen des allgemeinen V-Modells
  Identifizierung von Testobjekten, -zielen, -umgebung, -strategie
- Spezielle Aspekte: Testen bei Wartung, Weiterentwicklung, inkrementeller Entwicklung
- Abgrenzung Testebenen von Testtypen (auch bezogen auf Q-Kriterien)

Teil IV

Statisches Testen

Kursübersicht

- Grundlagen des Software Testens (Fundamentals of Software Testing)
- Testen im Entwicklungszyklus (Testing in the Software Life Cycle)
- Statisches Testen (Static Test)
- Dynamisches Testen (Dynamic Analysis - Test Design Techniques)
- Test Management
- Testwerkzeuge (Test Tools)

Überblick Statische Tests

- Strukturierte Gruppenprüfungen
- Statische Analyse
- Zusammenfassung

Motivation

- Static Test = manual checks + static analysis
- ohne Ausführung des Testobjekts
- Ziel: find defects and deviations from spec, standards, project plan
- als Fehlerverhinderung - bevor Fehler Wirkung zeigen
- oft als Exit-Kriterien bei Phasenbasierter Entwicklung
- in diesem Kapitel: verschiedene Arten von Reviews und (automatisierter) statische Analysen

Statisches Testen

Manuelle Prüfungen

- Nutzung der menschlichen Analyse- und Denkfähigkeit - Umgang mit Komplexität
- Vorgehen: intensives Lesen
- Unterschiedliche Ansätze bzgl
  - Intensität (intensity)
  - Formalität (formality)
  - Ressourcen (staff, time)
  - Zielen (objectives)

Definition Review allgemein

- Oberbegriff für (nicht gleichzusetzen mit)
  - Fagan inspection
  - Walkthrough
  - Technical inspection
  - informal review
- Gegenstand: jede Art Dokument
- Unter Beteiligung von Kollegen des Autors (peer review)
- Generelles Ziel: Erhöhung der Qualität
- Notwendig:
  - klares Ziel (goal) des Reviews offen formuliert
  - Auswahl des richtigen Reviewteams, basierend auf Kenntnissen und Fähigkeiten

Vorteile von Reviews (positive effects)

- kostengünstige Fehlerbeseitung (da früh erkannt)
- Verkürzung der Entwicklungszeit
- Reduzierung des Aufwands (Zeit, Kosten) für dynamische Tests
- generelle Kostenreduzierung bei Entwicklung und Betrieb - speziell: frühe Erkennung von inkonsistenen und unpräzisen Kundenwünschen
- reduzierte Fehlerrate im Betrieb
- Wechselseitiges Lernen im Review Team, Verbesserung der Qualität der Zusammenarbeit
- Klare und verständliche Dokumentation von Fakten notwendig
- gemeinsame Verantwortung für die Produktqualität
- systematische reviews können mehr als 70% der Defekte in Dokumenten finden

Potentielle Probleme mit Reviews

- schlecht moderierte Reviews: der Autor kann sich persönlich angegriffen fühlen
- hoher Zeitaufwand ohne sichtbare Produktivität
- hohe Kosten (10-15 % des Entwicklungsaufwands) aber erwartete Einsparung von 14-25%
- unerfahrene Reviewer können

Verlauf eines Reviews allgemein (nach IEEE1028)

- Planung (planning)
- Einführung (overview / kick-off)
- Vorbereitung (preparation)
- Reviewtreffen (review meeting)
- Überarbeitung (rework
- Nachfolgesitzung (follow up) - zweites Review nach Änderung
  auch: identifizieren genereller Probleme im Entwicklungsprozess

Regeln eines Reviewmeetings

- Beschränkung auf 2 Std, ev. Fortsetzung frühestens am nächsten Tag
- Dokumente sind Gegenstand des Reviews, nicht deren Autoren
  - Ausdrucksweise im Griff behalten
  - Autor muss udn soll sich nicht verteidigen
  - Moderator kann und sollte emotionale Reviewmeetings sofort beeenden
- Moderator sollte nicht Reviewer sein
- Keine Diskussion allgemeiner Stilfragen außerhalb der Reviewziele
- Keine Entwicklung von Lösungen
- Jeder Reviewer muss Gelegenheit haben, seine Anmerkungen zu präsentieren
- Protokoll spiegelt Gesamturteil des Teams wieder
- Protokoll nicht als Anweisungsliste an Autor formulieren
- Befunde (issues) sollten gewichtet werden, z.B.
  - critical defect (object not suitable for its purpose; correction necessary for approval)
  - major defect (usability of product is affected, correction necessary)
  - minor defect (small deviation)
  - good (no change necessary)
- Ergebnis: Empfehlung für die Acceptance des review objects, Kategorien
  - accept (without change)
  - accept (with change, no further review)
  - do not accept (further review or checking measure necessary)
- Protokoll muss von allen Reviewteam-Mitgliedern unterschrieben werden

Review Rollen (1)

- Manager
  - wählt Reviewgegenstände aus, sowie Zusatzdokumente
  - stellt Team zusammen
  - sollten nicht am Review teilnehmen
  - entscheidet, welche Konsequenzen aus Reviewergebnissen gezogen werden
- Moderator
  - Verantwortlich für Verwaltungsaufgaben (planning, preparation)
  - sorgt für ordnungsgemäße Durchführung des Reviews
  - muss guter Diskussionsleiter sein
  - soll neutral sein und keine eigene Meinung äußern
  - verteilt ev. unterschiedliche Reviewaufgaben an einzelne Reviewer
- Autor (author)
  - (einer der) Ersteller des Reviewgegenstands
  - stellt sicher, dass das Dokument seine Review-Eingangskriterien erfüllt
  - führt notwendige Änderungen durch (rework)
  - darf sich nicht als Gegenstand der Kritik sehen

Review Rollen (2)

- Gutachter (reviewer, inspector)
  - individuelle Prüfung des Reviewgegenstands (ev. nach vorgegebenen Kriterien)
  - Identifiziert und beschreibt Probleme mit review object
  - u.U. auch: unterschiedliche Reviewer für unterschiedliche Aspekte
  - muss ausreichend für review meeting vorbereitet sein
  - soll nicht nur defects sondern auch gute Teile ausweisen
- Protokollführer (recorder, scribe)
  - dokumentiert Reviewergebnisse (problems, action items, decisions, recommendations)
  - muss online kurz und präzise mitprotokollieren können
  - u.U. sinnvoll: Autor als Protokollführer

Mögliche Schwierigkeiten bei Reviews

- Personen mit erforderlichen Fähigkeiten und Kenntnissen nicht verfügbar - wichtig hierbei: Moderator (psychological skills)
- unpräzise Abschätzung durch Management bzgl Zeitbedarf führt zu unbefriedigenden Reviewergebnissen
- Unzureichende Vorbereitung (wrong reviewer) führt zu unbefriedigenden Ergebnissen
- fehlende oder unzureichende Dokumente
- fehlende Managementunterstützung

Grobcharakterisierung Reviewtypen

Unterscheidung nach

- reviews von technischen Produkten oder Teilprodukten, die während der Entwicklung entstehen
- Reviews von Projektplänen und Entwicklungsprozessen

Hier: erste Gruppe mit Schwerpunkt auf die Unterschiede

- walkthrough
- inspection
- technical review
- informal review

Walkthrough

- informell
- Ziel: finden und lokalisieren von Defekten, Mehrdeutigkeiten, Problemen in Doku
- auch: Diskussion von Alternativen, prüfen von Standard-Konformität
- keine Vorbereitung, nur Meeting ohne Zeitlimit
- Autor stellt Produkt anhand von use cases, Szenarien etc vor
- reviewer (meist andere Entwickler) stellen Fragen
- Team $\leq$ 10 Personen
- Vorbereitungsaufwand gering
- eher für nicht-kritische Dokumente
- kein explizites follow-up
- Autor als Chair - kann zu Problemen führen
- auch möglich: Vorbereitungsphase
- kann formeller oder informeller durchgeführt werden

Inspection

- sehr formal (Fagan Inspection), klar definierte Rollen, formale entry / exit criteria
- andere Namen: design inspection - kann aber für jedes andere Dokument verwendet werden
- klarer Zeitplan, klarer Ablauf
- Protokollierung aller Ergebnisse; am Ende Überprüfung auf Vollständigkeit
- Protokollierung von einstimmigen und abweichenden Ergebnissen
- Protokoll muss Gesamturteil (judgement) für Gesamtreview enthalten
- Datensammlung für Bewertung des Entwicklungs- und des Inspektionsprozesses

Technical Review

- Fokus: Übereinstimmung des Reviewgegenstands mit seiner Spezifikation, Eignung für geplanten Einsatz, Standardkonformität
- Reviewteam aus technischen Experten (aber möglichst nicht aus dem Projekt selber)
- Diskussion nur der wichtigen Anmerkungen der Reviewer (Aufgabe Moderator)
- Hauptaufwand: Vorbereitung
- Reviewmeeting ohne Autor
- Protokoll hält alle relevanten einstimmigen und abweichenden Bewertungen fest
- Grad der Formalität kann variieren

Informal Review

- informal review = review light
- sollte aber lose dem review process folgen
- oft vom Autor einberufen (Unterstützung bei Entscheidungen,Lokalisieren von Fehler)
- häufig: Gegenlesen durch andere Entwickler, Designer etc.
- Varianten:
  - Pair Programming
  - buddy testing
  - code swapping
- hohe Akzeptanz (bei Entwicklern und Management)

Wann welcher Reviewtyp?

- Detaillierte Dokumentation erforderlich? $\Rightarrow$ formales Review
- Schwierigkeiten bei Terminfindung und Zeit? $\Rightarrow$ eher informelles Review oder Walkthrough informell
- Technische Kenntnisse für Review erforderlich? $\Rightarrow$ Technial Review oder informelles Review
- Viele qualifizierte Reviewer benötigt? dann formales Review, aber wenn keine hohe Motivation? dann eher informelles Review
- Zu hoher Vorbereitungsaufwand im Vergleich zu erwarteten Ergebnissen? Dann eher informell
- Reviewgegenstand nicht sehr formal dokumentiert? Dann nicht formal
- Manegement support fehlt? dann nicht formal
- Generell: Typ und Erfolg des Reviews hängt von der Organisation und dem Umfeld ab, in dem sie stattfinden

Erfolgsfaktoren von Reviews

- Befunde müssen klar, neutral und objektiv dokumentiert werden
- der Autor sollte das Review als positive Erfahrung annehmen
- Hintergrund und Kenntnisse der Reviewer und Art der Dokumente erfordern geeigneten Reviewtyp
- Checklisten und Guidelines als Unterstützung
- Geeignetes Training speziell für formale Reviews
- Managementunterstützung bei Ressourcenplanung
- Akzeptanz von continuous learning - auch für review process

Statisches Testen

Statische Analyse allgemein

- Analyse ohne Ausführung des Prüfobjekts – wie bei Reviews
- Ziel: Aufdecken von Defekten
- Unterschied zu Reviews: Werkzeugbasierte Analyse
- Erfordert Dokumente in gewisser formaler Struktur / Sprache
- Typische Gegenstände statischer Analysen:
  - technische Anforderungen
  - SW Architekturmodelle
  - SW Design - z.B. als UML-Klassendiagramme
- in der Praxis: oft nur der Code in geeignetem Formalitätsgrad
- Meist: urchgeführt durch Entwickler bei Komponenten- oder Integrationstest
- auch: statische Analyse als Input für Reviews
- findet nicht alle Typen von Fehlern; z.B. keine Laufzeitprobleme
- Dafür werden Bereiche in Programmen ohne Probleme abgedeckt, die mit dynamischen Tests schwer erreichbar sind
- Eine ganze Reihe von statischen Analysen möglich in Compilern oder anderen Werkzeugen der Entwicklungsumgebung

Statische Analyse - typische Fehler

Fehlerklassen, die durch statische Analysen gut gefunden werden

- syntax violations
- Abweichung von (Coding-)Konventionen und Standards
- Kontrollflussanomalien
- Datenflussanomalien
- auch Security problems (buffer overflow, boundary checks of input data

Compiler as Static Analysis Tool

Zusatzinformationen, die mit speziellen compile options erhältlich sind:

- cross reference lists
- check for strict typing (data usage, variable usage)
- undeclared variables
- unreachable code
- overflow / underflow of field boundaries (static addressing)
- interface consistency
- labels for jump start or target
- bei C compilern: lint als Teil der statischen Analyse im Compiler integriert
- Erweiterung: splint
- auch: weniger statische Werkzeuge wie valgrind

Examination of Compliance with Programming Conventions and Standards

- heute oft Bestandteil der Entwicklungsumgebung z.B. Together Audits, Java Checkstyle als Eclipse Plugin
  Programmiersprachspezifisch!
- Hinweis: nur durch Werkzeuge überprüfbare Regeln verwenden
- Problem: Akzeptanz von Konventionen
- Achtung: häufig ist der Aufwand der Konfiguration nicht zu unterschätzen

Datenflussanomalieanalyse

- Anomalie = Abweichung eines Moduls oder Programms bei bestimmten Eigenschaften
- Nicht alle Datenflussanomalien bewirken Fehler in Programmen
- Sie sind aber Indikatoren für mögliche Fehler
- Datenfluss: entsteht durch die Umsetzung von Eingabewerten in Ausgabewerte über Zwischenergebnisse. Der Datenfluss hängt von den Kontrollstrukturen des Codes ab.
- Im Folgenden: Datenflussanomalien im Beispiel

Basis der Datenflussanalyse

- Verschiedene Vorkommensarten von Variablen (lokale, globale, Parameter)
  - x wird definiert (d): der Variablen x wird ein Wert zugewiesen

$x = 5$

Anmerkung: Deklaration nicht gleich Definition! vgl C

  - x wird referenziert (r): Der Wert der Variablen x wird in einer Berechnung oder Entscheidung gelesen, aber nicht verändert

$y = x + 1$ oder if $(x \geq 0)$ ....

  - x wird undefiniert (u):Der Wert der Variablen wird zerstört.

Zu Beginn eines Programms sind alle Variablen undefiniert
Nach Beenden einer Funktion sind alle lokalen Variablen undefiniert (C)

  - x wird nicht benutzt (empty): x wird von einem Knoten im Kontrollfluss überhaupt nicht benutzt

Datenfluss als Zugriffssequenz

- Fokus auf einer Variablen und deren Zugriffe im Laufe einer Abarbeitung eines Programms oder einer Funktion
- An diesen Folgen können mögliche Problem abgelesen werden

1. u r d r u
2. u d d r d u

- Daraus lassen sich 3 Anomalien ablesen:
  - ur-Anomalie: Referenz einer Variablen auf einen undefinierten, zufälligen Wert
  - dd-Anomalie: zwei aufeinanderfolgende Variablendefinitionen wobei die zweite Definition die erste überschreibt bevor der Wert verwendet wurde
  - du-Anomalie: der definierte Wert wird nicht benutzt bevor der gerde definierte Variablenwert zerstört wird.

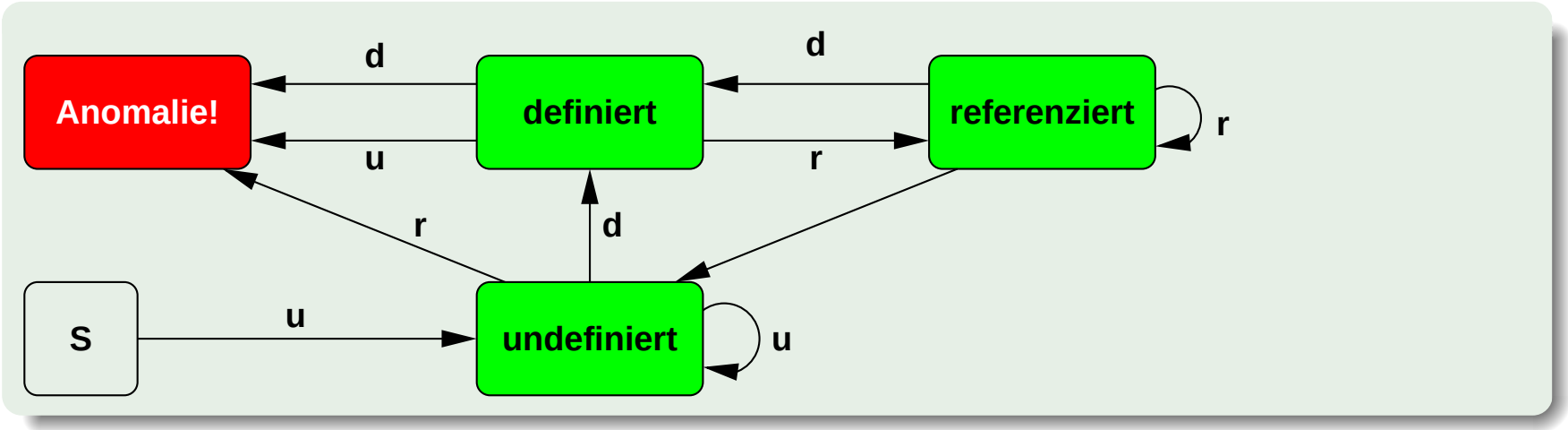
Beispiel: MinMax

```
void MinMax (int& Min, int& Max) {
    int Hilf;
    if (Min > Max) {
        Max = Hilf;
        Max = Min;
        Hilf = Min;
    }
}
```

Welche Anomalien stecken wo in dieser Funktion?

Systematische Vorgehen zur Analyse von Datenflussanomalien

- Verwendung eines Zustandsautomaten:



- pro Variable
- für einen Block von Definition bis Zerstörung
- Probleme beim Erreichen von Anomalie! oder wenn am Ende nicht undefiniert erreicht ist.

Probleme mit Datenfluss-Analyse

- exterm Programmiersprachabhängig
- so wie beschrieben nur für Basic Blocks
- keine expliziten Hinweise zum Umgang mit
  - Parametern
  - Funktionsaufrufen
  - Strukturierten Elementen (z.B. Arrays)

Control Flow Analysis

- Basis: Kontrollflussgraph
- spezieller Augenmerk auf: Verzweigungen, Schleifen
- typische Problemsituationen - als Anomalie, nicht Fehler
  - Sprünge aus Schleifen heraus (oder hinein)
  - Statements mit mehreren Ausgängen
- Notwendig: automatische Erstellung des Kontrollflussgraphen
- Problem: sehr komplexe Graphen - Indikator für potentielles Risiko

SW Metriken

- Quantitative Prüfung Allgemein:

  > *Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.*
  >
  > *Fenton*

  Dabei:
  - Entität = Objekt oder Ereignis der realen Welt
  - Attribut = Eigenschaft oder Aspekt einer Entität
- Grundannahme: von der Quantität kann man auf die Qualität schließen

  > *You can not control what you can not measure.*
  >
  > *De Marco*

- Produkt und Prozessmetriken
- Messbare Eigenschaften bei SW:
  - Größe - Komplexität - Kritikalität - Fehlerrate
- Messbare Eigenschaften des Prozesses (Fortschritt)
  - Mittelabfluss (Zeit, Geld)
  - Abdecken der Anforderungen
  - Erreichen von Planungszielen

Definition von Messprogrammen - 1

- Verlangt nach zB ISO 9001:200, auch ECSS, CENELEC
- Was soll gemessen werden?
  - Was sind die Qualitätsziele, die geprüft werden sollen?
  - Welche Daten werden benötigt um Aussagen machen zu können

  $\Rightarrow$ GQIM = Goal - Question - Indicator - Metrik
  - Anmerkung: es ist nicht sinnvoll alles zu messen, was gemessen werden kann; Daumenregel: max 10 Metriken
  - Achtung: die Detaildefinition von Daten ist u.U. auch abhängig von Programmiersprache
- Wann soll gemessen werden?
  - In welchen Phasen der Entwicklung (für welche Daten)
  - Mit welcher Frequenz (einmal zum Ende der Phase, regelmässig)?

Definition von Messprogrammen - 2

- Wer soll Daten erfassen?
  - Verantwortlicher und Ausführende
  - manuell oder automatisch - abhängig von SW Entwicklungsumgebung
- Wie sollen Daten ausgewertet werden?
  - Oft müssen Verhältnisse zwischen Daten hergestellt werden
  - Verlaufsdaten vs. Einzeldaten
  - Festlegen von Kriterien, die auf Probleme hinweisen (Schwellwerte, kritische Verläufe, auch im Vergleich)
- Wo und von wem werden die Daten archiviert?

Allgemeines Problem: Akzeptanz - Messprogamme können leicht als Kontrollmechanismus missbraucht werden!

Für den Certified Tester relevante Metriken

- Komplexitätsmetrik auf Basis der zyklomatischen Zahl
- gemessen am Kontrollflussgraphen
- Berechnung: Gegeben Graph G mit e Kanten und n Knoten. Dann ist die cyclomatic number v(G) gegeben durch

$$v(G) = e - n + 2$$

  Anmerkung: +2 da ein virtuelle Rücksprungkante vom Endknoten zum Anfang angenommen wird.
- Gibt Information bzgl. Testaufwand (Anzahl unabhängiger Pfade)
- auch relevant für maintainability
- Daumenregeln: $v(G) > 6$ nicht akzeptabel

Statisches Testen

Summary Static Tests

In diesem Kapitel:

- Review als Prozess
- Reviewtypen: walkthrough, inspection, technical review, informal review
- Automatisierte Statische Analysen:
  - was kann der Compiler,
  - Prüfen gegen coding conventions und coding standards
  - Datenflussanalyse
  - Kontrollflussanalyse
  - Metriken

Teil V

Dynamisches Testen

Kursübersicht

- Grundlagen des Software Testens (Fundamentals of Software Testing)
- Testen im Entwicklungszyklus (Testing in the Software Life Cycle)
- Statisches Testen (Static Test)
- Dynamisches Testen (Dynamic Analysis - Test Design Techniques)
- Test Management
- Testwerkzeuge (Test Tools)

Überblick Dynamischer Test

- Blackbox Tests
- White Box Tests
- Intuitive Tests und Erfahrungsbasierte Testfallermittlung
- Zusammenfassung

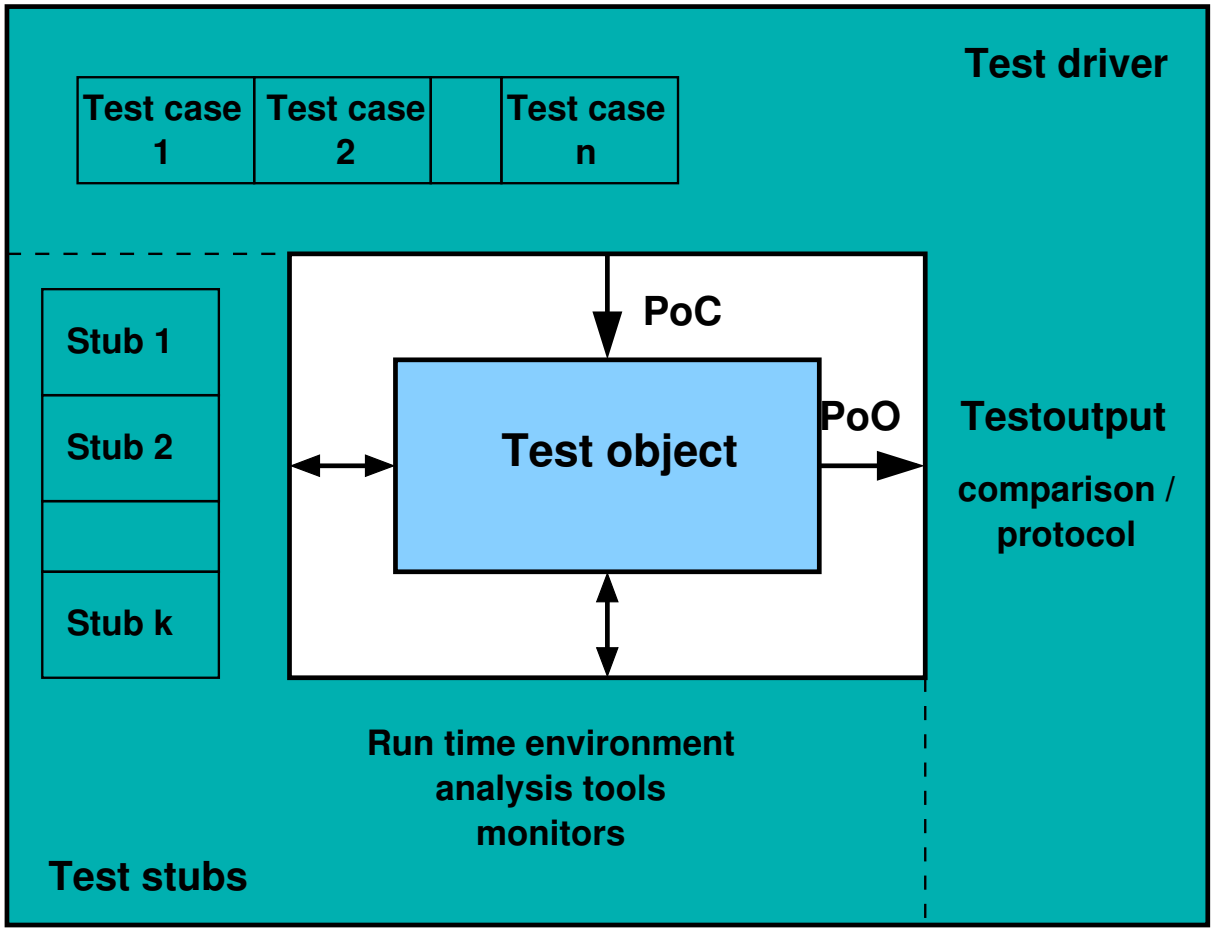
Dynamischer Test: Ansätze allgemein

- dynamischer Test = Prüfung bei/durch Ausführung des Testobjekts
- Ziel: Auswahl geeigneter Eingaben / Ereignisfolgen, die bestimmte Verhalten des Systems / der Komponente erzwingen
- Ansätze:
  - Anforderungsbasiert (überwiegend Systemtests)
  - Partitionstests - Tests von Vertretern einer identifizierten Gruppe von Eingaben / Ereignissen, die sich unterschiedlich verhalten. Bekannteste Variante: Äquivalenzklassentests
  - Strukturelle Tests: basiert auf der Struktur des Programms (meist: Komponententests); Zusammenhang zu Testüberdeckungsforderungen (Pfadüberdeckung, Anweisungsüberdeckung)
- Empfehlung: erst abstraktere Testebenen dann konkretere d.h. Anforderungen - Partitionen - Struktur: zur schrittweisen Verfeinerung der Testfälle
- Empfehlung: Entwurf von high-level Tests sollte früh starten

Aufbau eines Testbetts

(nach Spillner)

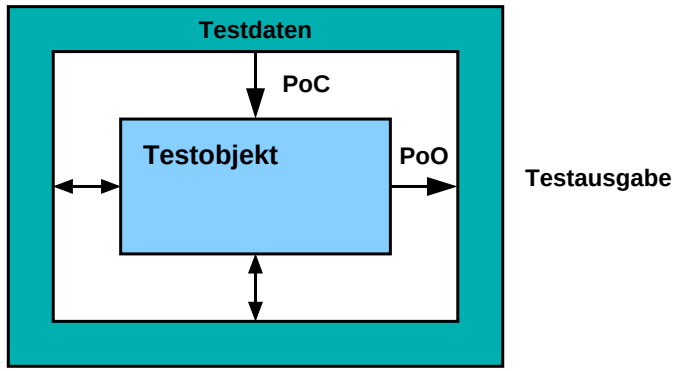
Testbed nach Spillner



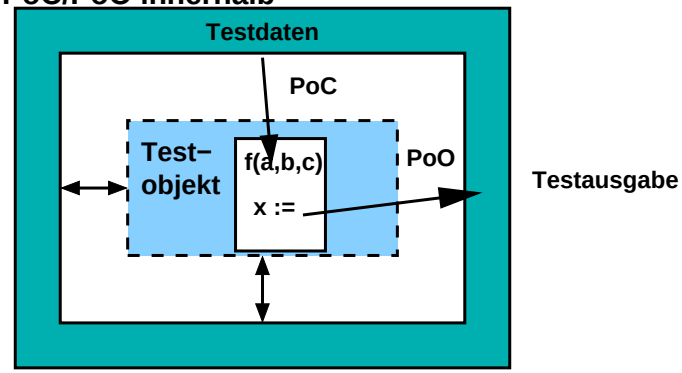
Generelle Methoden für Testfall-Spezifikation

- Black-Box-Verfahren
  keine Verwendung von Wissen über Aufbau des Testlings;
  Point of Control/Point of Observation außerhalb
  meist Test gegen Spezifikation
  - Äquivalenzklassenbildung
  - Grenzwertanalyse
  - Zustandsbezogener Test
- White-Box-Verfahren (Glass-Box)
  strukturelle Testverfahren (basierend auf Komponentenhierarchie, Kontrollfluss, Datenfluss)
  eher Modultests
  - Anweisungsüberdeckung
  - Zweigüberdeckung
  - Test der Bedingungen
  - Pfadüberdeckung
- Intuitive Testfallermittlung

Blackbox: PoC/PoO ausserhalb



Whitebox: PoC/PoO innerhalb



PoC: Point of Control
PoO: Point of Observation

Generelles Vorgehen für die Identifikation von Testfällen

- Ziel: so wenig wie möglich Testfälle, aber so viele wie nötig (zum Aufdecken relevanter Fehlerzustände)
- systematisches Vorgehen - nicht aus dem Bauch heraus
- Vorgehen in Schritten:
  - Identifiziere conditions und preconditions für einen Test und Testziele (goals)
  - Spezifiziere einzelne Testfälle (test cases)
    inkl. input data, expected values
  - Lege fest, wie diese Testfälle auszuführen sind (chains, suites)
    auch: test procedure
    auch: Prioritäten, Abhängigkeiten, Zeitplanung
    oft: Testskripte (test script)
- dokumentiere diese Schritte formal oder informell

Dynamisches Testen

Black-Box: Äquivalenzklassenbildung (1)

- Menge der möglichen konkreten Eingabewerte wird in Äquivalenzklassen eingeteilt; je Klasse wird mindestens ein Repräsentant getestet
- Äquivalenzklasse: Menge von Werten die ein gleichartiges Sollverhalten zeigt (Eingabe- oder Ausgabewerte des Testobjekts)
- Systematisches Herleitung von Testfällen mit Äquivalenzklassen
  - ermittle für jede Eingabevariable den Definitionsbereich $\Rightarrow$ zulässige Werte; diese müssen gemäß der Spezifikation verarbeitet werden.
  - Verfeinerung der Äquivalenzklassen auf Basis der Spezifikation; Festlegen der erwarteten Ergebnisse und eventuell Vorbedingungen für den Testlauf
  - Auch: Zerlegung des Ausgabebereichs - aber hier ist die Ermittlung geeigneter Startdaten schwieriger
  - Speziell: unzulässige Eingaben und Ausgaben
  - Aussichtsreich: Grenzwerte der Äquivalenzklassen - decken oft Missverständnisse oder Ungenauigkeiten auf

Black-Box: Äquivalenzklassenbildung (2)

Tipps für die Ermittlung von Äquivalenzklassen:

- Ermitteln von spezifizierten Einschränkungen und Bedingungen für Eingaben und Ausgaben aus der Spezifikation; für jede eine Äquivalenzklasse
- Zusammenhängender Wertebereich: ein gültiger, zwei ungültige Äquivalenzklassen
- Anzahl von Werten einzugeben: Klasse mit richtiger Anzahl richtiger Werte, je eine Klasse für Unter- und Überschreitung der gültigen Anzahl
- Menge von unterschiedlich zu behandelnden Werten führen zu je einer gültigen und einer ungültigen Äquivalenzklasse
- Situationen durch zwingend zu erfüllende Einschränkung / Bedingung beschrieben: je eine gültige und eine ungültige Äquivalenzklasse
- Zweifel an Gleichbehandlung der Werte: weitere Teilung der Äquivalenzklasse

Black-Box: Äquivalenzklassenbildung (3)

Definition von Testfällen

- Äquivalenzklassen: Fälle für einzelne Eingabeparameter
- Testfälle als Kombination der jeweiligen Repräsentanten:
  - gültige Testfälle: Kombinationen von allen Äquivalenzklassen mit gültigen Werten
  - Negativtestfall: Kombination eines ungültigen Repräsentanten für einen Parameter mit gültigen Repräsentanten der anderen
- Explosion der gültigen Testfälle durch Kreuzprodukt - Regeln zur Reduzierung:
  - Testfälle nach Nutzerprofilen priorisieren
  - Testfälle bevorzugen, die Grenzwerte verwenden
  - Nur zwei Parameter permutieren (paarweise Kombination statt vollständiger Kombination)
  - Repräsentanten ungültiger Äquivalenzklassen nicht mit anderen Repräsentanten ungültiger Klassen kombinieren (Gefahr der Fehlermaskierung!)

Black-Box: Äquivalenzklassenbildung (4)

Festlegen von Testkriterien

- Endekriterium - Wann ist genug getestet?
- zB. als Verhältnis von Anzahl durchgeführter Tests zu Gesamtzahl von Äquivalenzklassen:

  ÄK-Überdeckung = (Anzahl getesteter ÄK /Gesamtzahl ÄK) * 100%

  EC-Coverage = number of tested EC / total number of EC) * 100%
- Achtung: Fehlen Äquivalenzklassen, so kann ein hoher Überdeckungsgrad irreführen!

Bewertung:

- Sorgfältige Bestimmung der Klassen liefert systematisch relevante Tests; unnütze Tests werden vermieden
- Keine Berücksichtigung von Abhängigkeiten und Wechselwirkungen zwischen Parametern
- In Kombination mit fehlerorientierten Verfahren wie der Grenzwertanalyse wirkungsvoll

Black-Box: Grenzwertanalyse (1)

- Beobachtung: Fehler treten häufig an den Grenzbereichen der Äquivalenzklassen auf
- Nur möglich bei geordneten Datenmengen, deren Grenzen identifizierbar sind

Beispiel für geordnete Mengen mit Grenzwerten:

- INT mit MinInt und MaxInt
- Integer aus [-5, 5]

Beispiele für Mengen ohne Grenzwerte

- reelle Zahlen aus (-5, 5)
- {ROT, GELB, GRÜN}

- Testfälle:
  - Grenzwerte (gültig und ungültig) zB. der Äquivalenzklassen

Black-Box: Grenzwertanalyse (2)

Tipps zur Testfallerstellung

- Datenbereich (Eingabe und Ausgabe): Grenzwerte und benachbarte Werte ausserhalb des Bereichs
- Grenzwerte bei Anzahl (Eingabe / Ausgabe): Min, Max, Min - 1, Max + 1
- geordnete Mengen: erstes und letztes Element
- komplexe Datenstrukturen: leere Liste, Nullmatrix, etc.
- Numerische Berechnungen: eng zusammenliegende Werte und weit auseinanderliegende Werte
- ungültige Äquivalenzklassen: nur sinnvoll, wenn Ausnahmebehandlung mit Grenzen assoziiert.

Festlegen der Testendekriterien:

- Analog zu Äquivalenzklassenüberdeckung:

  GW-Überdeckung = (Anzahl getesteter GW /Gesamtzahl GW) * 100%

  BV-Coverage = (number of tested BV /total number BV) * 100%

Bewertung:

- In Verbindung mit Äquivalenzklassen sinnvoll
- erfordert hohe Kreativität, speziell bei komplexen Daten

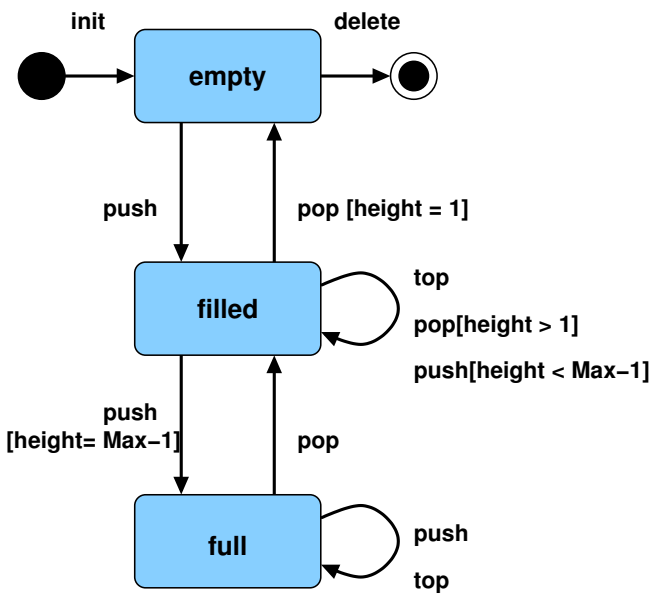
Black-Box: Zustandsbasierte Tests (1)

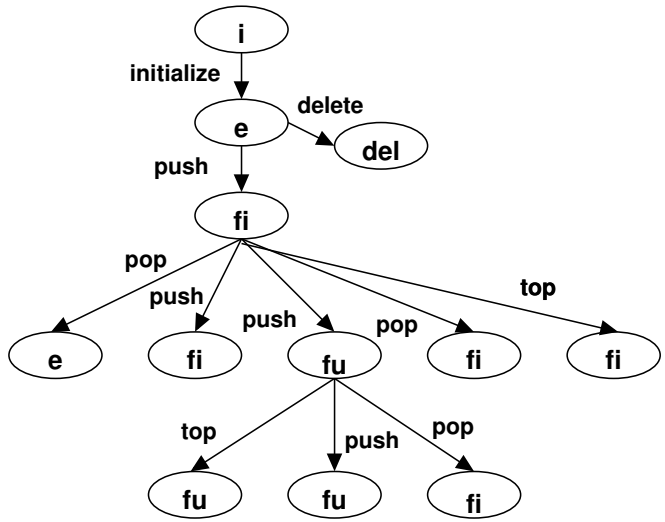
- Basierend auf Zustandsmodellen (Automaten) für die Historie des Testlings
- Startzustand - Folgezustände - Zustandsübergänge
- Verhalten hängt ab vom Zustand $\Rightarrow$ testen des Verhaltens in den jeweiligen Zuständen, die gezielt angesteuert werden müssen.
- Testkriterien: Basierend auf Übergangsbaum
  - Wurzelknoten: Anfangszustand
  - Berechne Knoten als Folgezustände, Kanten markieren mit Übergangsfunktion
  - Zweig terminiert wenn Knoten auf dem Weg von der Wurzel schon vorhanden oder zugehöriger Zustand ein Endzustand ist (inkl. Fehlerzustände!)
- Zwei Testziele
  - Konformitätstest (test of specified function)
  - Robustnesstest (test for robustness)

Black-Box: Zustandsbasierte Tests - Example

Stack mit 3 Einträgen

Zustände: leer, gefüllt, voll
Operationen: push, pop, top, delete Transitionen: leer nach gefüllt mit push; etc. Testfälle = ?





Transition tree for compliance test

Ex test sequence: init, push, pop, delete

Black-Box: Zustandsbasierte Tests (2)

Testfälle definieren:

- Ein Testfall ist definiert durch Anfangszustand, Eingabenfolge, erwartete Ausgaben/Aktionen, Endzustand
- Für jeden Zustandsübergang: Vorzustand, auslösendes Ereignis, Reaktion, Folgezustand
- Achtung: Es ist nicht immer offensichtlich in welchen Zuständen sich der Testling befindet! Eventuell ist dafür eine begrenzte Sicht auf interne Variablen nötig.
- Bei Spezifikation schon Testbarkeit prüfen: hohe Anzahl von Zuständen und / oder Übergängen ist problematisch - eventuell Vereinfachung vorschlagen
- Komplexe Zustandsdefinitionen sind problematisch - auch hierauf bei der Spezifikation achten

Black-Box: Zustandsbasierte Tests (3)

Testendekriterien - mögliche Varianten:

- jeder Zustand mindestens einmal erreicht.
- jeder Zustandsübergang mindestens einmal durchgeführt.
- alle spezifikationsverletzenden Zustandsübergänge geprüft.
- Erweitert:
  - Alle Kombinationen von Zustandsübergängen
  - alle Kombinationen von Zustandsübergängen in jeder beliebigen Reihenfolge, auch mehrfach hintereinander

  $\rightarrow$ Langzeittests, $\rightarrow$ Modellchecking

Bewertung:

- relevant wo Zustände eine Rolle spielen
- Auch: OO Klassentests - Zustände entsprechen Kombinationen von Attributwerten einer Klasse bzw ihrer Instanz

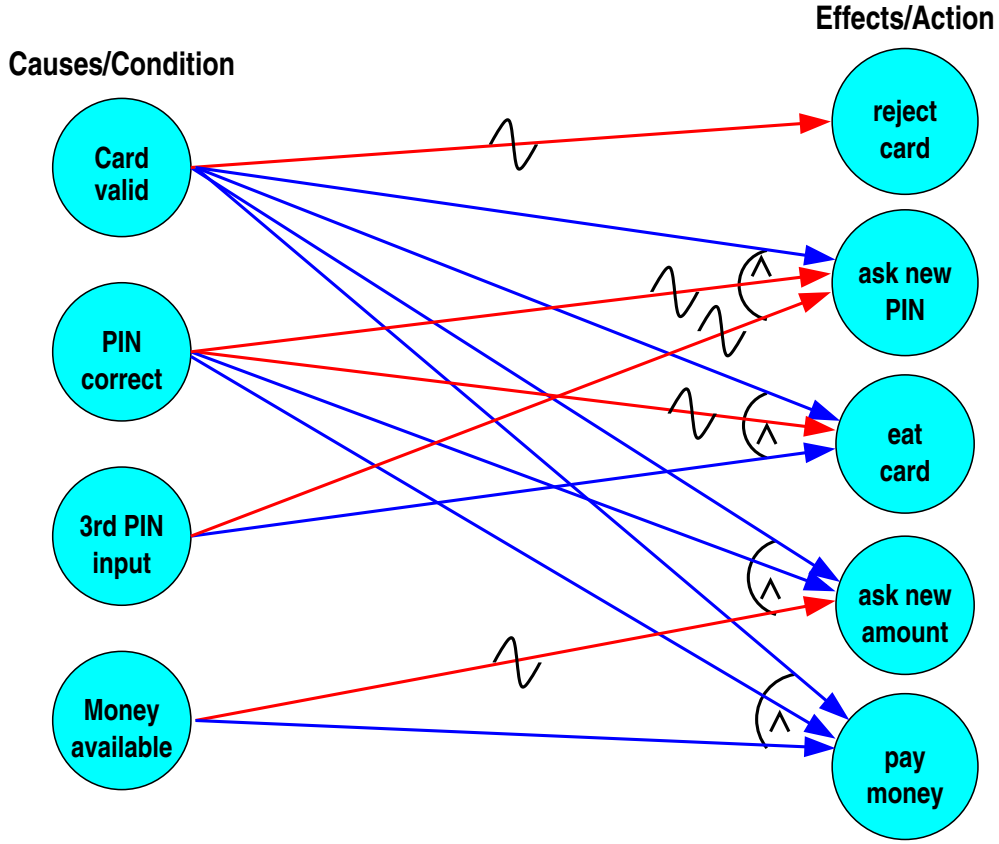
Ursache-Wirkungs-Analyse als Testansatz

- Cause-Effect Graphs (Ursache-Wirkungsgraphen) (Myers)
  - Verwendet Abhängigkeiten (dependencies) zwischen Eingaben (conditions)
    einfache Verknüpfungen: AND, OR, NOT
  - beschreibt Wirkung der Eingabedaten auf Ausgabe/Systemverhalten actions)
- Decision Tables (Entscheidungstabellen)
  - Obere Hälfte: conditions und ihre Kombination
  - Untere Hälfte: actions und Info ob oder ob nicht getriggert.
- Systematische Transformation des Graphen in die Entscheidungstabelle
- Systematische Ableitung von test cases aus Tabelle: jede Spalte ist ein Testfall (logischer Testfall)
- Test Completion Criteria: Minimum jede Spalte mindestens einmal
- Bewertung:
  - systematisches Vorgehen - deckt u.U. Kombinationen auf, die sonst ausgeschlossen würden
  - Explosion bei vielen Bedingungen und vielen Actions

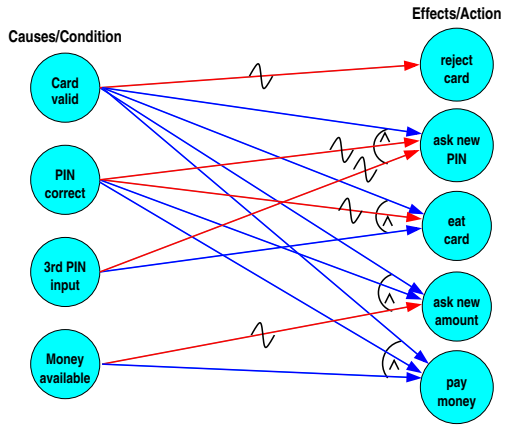
Beispiel Ursache-Wirkungs-Graph (1)

Beispiel: Einfacher Bankautomat

- Conditions
  - bankcard valid
  - PIN entered correctly
  - maximum number of PIN inputs = 3
  - money available in machine and in account
- Actions
  - reject card
  - ask for another PIN
  - eat card
  - ask for different amount
  - pay requested amount



Beispiel Ursache-Wirkungs-Graph (2)



Cause / Condition	1	2	3	4	5
bankcard valid	N	Y	Y	Y	Y
PIN correct	-	N	N	Y	Y
3rd PIN	-	N	Y	-	-
Money available	-	-	-	N	Y
Effect / Action					
reject card	Y	N	N	N	N
ask new PIN	N	Y	N	N	N
eat card	N	N	Y	N	N
ask new amount	N	N	N	Y	N
pay money	N	N	N	N	Y

Use Case Testing

- Motivation: use cases als Requirementspec (vor allem bei OO-Entwicklung)
- auch: UML use case diagram
- halten externe Sicht des Systemverhaltens fest (actor - use case mapping)
  $\Rightarrow$ relevant für System und Abnahmetest
- geringes Maß an Strukturierung:
  - include - für gemeinsame Teilfunktionalität mehrerer Use Cases
  - extend - für Ausnahmefälle
  - generalize - für Spezialisierung resp. Abstraktion
- UC Diagrams taugen nicht wirklich um Anforderungen zu dokumentieren
  - fehlt: Precondition, Postcondition, Systemverhalten (Reaktion / Interaktion)
  - $\Rightarrow$ lieber ergänzen durch UC Testschablonen z.B. mit Haupt- und Alternativverhalten - auch Ausnahmen explizit
- Für Test: jeder UC mindestens einmal - auch: jede Folge von UCs im Diagramm
- Achtung: keine Unterstützung für Ableitung von Eingabedaten oder Sollverhalten! - sehr abstrakte Testfälle
- Wert: für Kunden/Endnutzer gut nachvollziehbar
- Ähnlich, aber oft konkreter: User Stories (XP), Usage Szenarien

Weitere Black-Box-Tests

- Syntaxtest - bei formaler Spezifikation der Syntax zB von Eingabemasken
- Zufallstest (random test)
- Smoke-Test = Ausprobieren des Testobjektes

Black-Box-Tests

Allgemeine Bewertung

- Vorteile Black-Box-Tests
  - Prüfung der Funktionalität unabhängig von Implementierung
- Nachteile Black-Box-Tests
  - Fehlerhafte Spezifikationen werden nicht erkannt
  - Nicht geforderte Funktionalität wird nicht erkannt (auch: toter Code)
- Anwendung bei Abnahmetests, Schnittstellentests - überall dort wo die äußere Funktion relevant ist

Dynamisches Testen

White-Box: Anweisungsüberdeckung (1)

- Testfälle so wählen, dass alle oder eine festgelegte Quote der Anweisungen mindestens einmal ausgeführt werden (C0-Maß)
  Dh. jeder Knoten sollte einmal besucht werden
- Basiert auf Kontrollflussgraph (Darstellung der Kontrollstruktur einer Komponente als Graph mit gerichteten Kanten; Elemente: Sequenz, Verzweigung, Schleifen, Basisanweisungen)
- Testdurchlauf beinhaltet Nachweis, welche Anweisungen ausgeführt wurden.
- Unter Umständen genügt ein Testfall
- Testendekriterium: Quote (analog zu GW, ÄK)
- Bewertung:
  - schwaches Kriterium - muss nicht alle Fälle abdecken
  - identifizierten potentiell unerreichbaren Code
  - erkennt u.U. leere IF-Zweige nicht

White-Box: Zweigüberdeckung

- Testfälle so wählen, dass alle oder eine festgelegte Quote der Bedingungen mindestens einmal ausgeführt werden (C1 - Maß)
- Basiert auf Kontrollflussgraph
  - bei Bedingungen die Auswertung zu TRUE und zu FALSE
  - alle CASE-Fälle
  - Schleifen: ohne Durchlauf, Durchlauf und Wiederholung
- Testdurchlauf beinhaltet
  - Entscheidung über weiteren Testverlauf basierend auf der Auswertung einer Bedingung
  - Nachweis, welche Bedingungen und Anweisungen ausgeführt wurden (Zweige)
  - Berücksichtigung von leeren IF-Zweigen ergibt sich aus Berücksichtigung der Bedingungen
- Testendekriterien über Quote, empfohlen: 100 %
- Bewertung:
  - mehr Testfälle, da alle Möglichkeiten der Bedingungsauswertung zu berücksichtigen sind
  - Festlegen der erwarteten Ergebnisse analog zu Äquivalenzklassentest
  - Unzureichend für OO (nur innerhalb von Klassen möglich)

White-Box: Bedingungsüberdeckung

- Testfälle so wählen, dass alle Möglichkeiten der Bedingungsauswertung berücksichtigt werden ausgeführt werden
- Einfache Bedingungsüberdeckung (Branch Condition Testing) Kriterium: jede atomare Teilbedingung wird mit TRUE und FALSE berücksichtigt
- Mehrfache Bedingungsüberdeckung (Branch Condition Combination Testing) Kriterium: jede Kombination der atomaren Bedingungswerte soll berücksichtigt werden
- Minimale Mehrfachbedingungsüberdeckung (Modified Branch Condition Decision Testing) Kriterium: nur solche Varianten von Kombinationen atomarer Bedingungswerte, die sich unterscheiden
- Testfälle: so auswählen, dass alle Bedingungen erreicht werden, u.U. ist dafür eine Rückwärtsanalyse notwendig
- Testendekriterien: basierend auf Quote
- Bewertung
  - Empfehlung: Minimale Bedingungsüberdeckung
  - Einfache Überdeckung: nicht ausreichend, Mehrfachüberdeckung i.A. zu aufwändig
  - u.U. aufwändige Auswahl von Testfällen

White-Box: Pfadüberdeckung

- Testfälle so wählen, dass alle Pfade inkl. Schleifen / Wiederholungen ausgeführt werden
- Beinhaltet die Kombination aller Pfade durch jeweilige Basisblöcke des Programms; berücksichtigt Abhängigkeiten zwischen Zweigen
- Bewertung:
  - Kombination von anderen White-Box-Verfahren
  - Berücksichtigung aller möglichen Anzahlen von Schleifendurchläufen - i.a. nicht mehr 100 % möglich
  - u.U. sehr aufwändig

White-Box-Tests

Weitere White-Box-Verfahren:

- Linear Code Sequence and Jump (LCASJ): Folgen von Anweisungen im Vordergrund (Basisblöcke), Kombination aller Basisblöcke sind zu berücksichtigen
- Datenbasierte Verfahren: basiert auf lesenden und schreibenden Zugriffen auf Variablen/Parameter, berücksichtigt Abhängigkeiten zwischen Variablen/Parameter; Unterscheidung zwischen Verwendung der Variablen/Parameter in und ausserhalb von Bedingungen
- andere Bedingungsüberdeckungen:
  - Modified Branch/Condition Coverage nach DO-178B

White-Box-Tests

Allgemeine Bewertung

- Grundprinzip: Basis ist der Programmtext - dessen Komplexität bestimmt den Aufwand
- Vorteile
  - geeignet für untere Testebenen (Modul, Komponente)
  - guter Support durch Werkzeuge für Testfallgenerierung und Testreporting
- Nachteile
  - nicht umgesetzte Anteile der Spezifikation werden u.U. nicht erkannt
  - instrumentierter Code (zB um Überdeckung zu messen) verhält sich unter Umständen anders als der Code alleine (speziell: Zeitverhalten)
- Empfehlung: Minimum Zweigüberdeckung

Instrumentation and Tool Support

- wesentlich bei white box tests:
  Nachverfolgbarkeit der ausgeführten Programmanteile
- oft muss dafür der Code instrumentiert werden
- Nachteil: Instrumentierung beeinflusst Systemverhalten (z.B. Laufzeit)
- für die Auswertung der erhobenen Daten: Tool verwenden!
- Auch: Code Coverage tools wie Clover, EclEmma, djUnit
  aber oft sprachabhängig!

Dynamisches Testen

Intuitive Testfallermittlung

- Ergänzung zur systematischen Testfallermittlung (black box, white box) - aber weder black box noch white box Verfahren
- Ansatz: intuitives Verständnis des Testers nutzen
  - zB. um Fehler zu finden
  - z.B. um noch unerreichbare Zweige/Pfade zu erreichen
- Dokumentation von Vor- und Nachbedingungen, erwartete Ausgaben, erwartetes Verhalten ist auch hier notwendig
- Empfehlung: nicht als einzigen Testansatz, nicht als erste Testkampagne
- Testendekriterien: können hier nicht formal festgelegt werden.
- Variante: exploratory testing
  - z.B. wenn Dokumente von schlechter Qualität
  - Testen zur Entwicklung von Verständnis für Testling (dabei entsteht ein "Modell")
  - erste Testfälle zur systematischer Entwicklung weitere Fälle
- oft gute Ergänzung, da weitere kritische Fehler aufgedeckt werden

Dynamisches Testen

Zusammenfassung Dynamische Tests

- Einführung von konkreten Testtechniken
- Blackbox-Techniken - für den Anforderungsbezogenen Test
- Whitebox-Techniken - für den Strukturbezogenen Test
- jeweils passende Testendekriterien (coverage)
- Empfehlungen
  - Blackbox: Äquivalenzklassen mit Grenzwertanalyse
  - Whitebox: Minimum Branch Coverage, loops: Minimum mehr als einmal durchlaufen
- Anmerkung: Techniken überwiegend für Funktionstests!

Dynamischer Test Add-on: Auswahl von Testtechniken (1)

Auswahl von Techniken auf Basis von

- Art des Testobjekts (kind of test object)
- Formale Dokumentation und Verfügbarkeit von Werkzeugen (formal documentation and availability of tools)
- Einhaltung von Standards (Conformance to standards)
- Erfahrung der Tester (tester experience)
- Kundenwunsch (customer wishes)
- Risikoanalyse (risk analysis)
- Weiter Faktoren (further factors):
  - previous experiences
  - Time and budget

Dynamischer Test Add-on: Auswahl von Testtechniken (2)

Einige Erfahrungswerte:

- Wenn verschiedene Zustände oder Wechsel von Zuständen eine Rolle spielen
  $\Rightarrow$ Zustandsbasierter Test
- Wenn Abhängigkeiten zwischen Eingabedaten Einfluss haben
  $\Rightarrow$ Entscheidungstabellen-basierter Test
- Abnahme und Systemtests aus Use Cases ableiten (mit Vorsicht....)
- Bei Komponenten- und Integrations-Test: coverage measures - durch white box tests ergänzen
- Path-coverage ist eher ein theoretisches Verfahren (in der Praxis zu hohe Kosten)
- white box für die unteren Ebenen, black box für die oberen
- intuitive Testfallermittlung nicht ignorieren - liefert wertvolle Ergänzung zu den systematischen Tests

Teil VI

Test Management

Kursübersicht

- Grundlagen des Software Testens (Fundamentals of Software Testing)
- Testen im Entwicklungszyklus (Testing in the Software Life Cycle)
- Statisches Testen (Static Test)
- Dynamisches Testen (Dynamic Analysis - Test Design Techniques)
- Test Management
- Testwerkzeuge (Test Tools)

Überblick SW Test Management

- Testorganisation (Test Organization)
- Testplanung (Test Planning)
- Kosten- und Wirtschaftlichkeitsaspekte (Cost and Economy Aspects)
- Definition von Teststrategien (Definition of Test Strategy)
- Management von Testaktivitäten (Test Activity Management)
- Fehlermanagement(Incident Management)
- Anforderungen an Configuration Management
- Relevante Normen und Standards
- Zusammenfassung

Aspekte des Testmanagement

- Organization von Test Teams,
- relevante Qualifikationen für Tester
- Aufgaben des Testmanagers und anderer Rollen
- Unterstützende Prozesse

Test Management

Test Teams und Unabhängigkeit

- Durchgängigkeit der Testaktivitäten $\Rightarrow$ Koordination und enge Kooperation
- Diskussion unabhängiger Tester:
  - Pro
    - unbhängige Tester sind unvoreingenommen (unbiased), sehen andere Defekte als Entwickler
    - unabhängige Tester verifizieren implizite Annahmen der Entwickler
  - Contra
    - unabhängige Tester werden häufig isoliert, Mangel an Kommunikation
    - unabhängige Tester werden oft zum Bottleneck, wenn die Ressourcen nicht gut geplant sind
    - Entwickler verlieren ihren Sinn für Verantwortung für das Produkt
- oft: multi-disciplinary teams
- auch: spezialisierte Anbieter von Testdienstleistungen :)

Empfehlungen zum Unabhängigen Testen (1)

- Formen unabhängigen Testens - oder auch: Formen der Unabhängigkeit
  1. Entwickler sind verantwortlich für den Test, testen aber wechselseitig Teilprodukte
  2. Tester als unabhängige Gruppe im Entwicklerteam
  3. Tester als separates Team im Projekt, nicht involviert in die Entwicklung;
     können zur Firma gehören, zu den Endnutzern, oder zu IT Operation
  4. Tester mit Spezialisierung auf spezifische Testaufgaben (Security, Usability, Performance)
  5. separate Organisation stellt Tester – wie bei Verified

Empfehlungen zum Unabhängigen Testen (2)

- Wann welcher Grad an Unabhängigkeit?
  - Testberater für die Planung, für Training, Coaching, Automatisierung
  - Komponententest (component test)
    - Enge Bindung an Entwicklungsaktivitäten
    - Entwicklertests vermeiden
    - Modelle 1 und 2 geeignet, aber mit Risiken (erste Prio Entwicklung, zweite Testen)
    - Unterstützung
      vom Projekt- oder Test-Management festgelegte testing standards, rules, schedules, log-formats und
      geschulte Tester zeitweise als Coaches dazunehmen
  - Integrationstest (integration test)
    - Modell 1, 2 wenn das gleiche Team wie beim Komponententest
    - Komponenten aus mehreren Teams: Vertreter aller Entwicklerteams oder unabhängiges Testteam (Modelle 3 oder 5)
  - Systemtest (system test)
    - Test soll aus customer oder end user Perspektive geschehen
    - $\Rightarrow$ Unabhängigkeit ist wesentlich (Modelle 3,4,5)

Rollen im Testteam und ihre Qualifikation

Rollen:

- Testmanager (test leader)
- Test designer (test analyst)
- Test automator
- Test administrator
- Tester (oft generisch für alle Rollen verwendet;) hier: Testausführung und -bericht

Wünschenswerte Qualifikation allgemein:

- Certified Tester Foundation Level und ev. Advanced Level
- Social competence:
  - ability to work in a team, political and diplomatic aptitude
  - scepticism = willingness to question apparent facts
  - persistence and poise (sicheres Auftreten)
  - accuracy and creativity
  - ability to get quickly acquainted with application (areas)

Aufgaben und Qualifikation Test Manager

- Skills
  - Test planning and test control expert
  - knowledge and experience in the fields of SW testing, Q Management, Project Management, Personnel Management
- Tasks
  - writing or reviewing test policy (übergeordnet)
  - writing test strategy and test plan
  - representing test perspective within project
  - procuring new test resources
  - selecting test strategies and methods, introduce / improve related processes
  - initiate and monitor test work (spec, impl, execution) on all levels
  - introduce suitable metrics for tes progress, evaluate Q of testing and product
  - select and introduce test tools, organize training, decide type and extent of test env and automation
  - plan test runs, adapt test plan based on results and progress (project and test)
  - write test reports and communicate these

Aufgaben und Qualifikation Test Designer

- Skills
  - Expert in test methods and test specs
  - knowledge and experience in the fields of SW testing, SW engineerng, (formal) specs
- Tasks
  - Analyse, review, assess user requirements, specs, designs and models for testability and design of test cases
  - create test specs
  - prepare and acquire test data

Aufgaben und Qualifikation Test Automator

- Skills
  - knowledge of testing basics
  - programming experience
  - excellent knowledge of testing tools, scrio languages
- Tasks
  - Automates tests, using available tools and script languages

Aufgaben und Qualifikation Test Administrator

- Skills
  - expert for installing and operating test env (system admin)
- Tasks
  - set up and support for test environment (Koordination mit Sys Admin and Network Admin)

Aufgaben und Qualifikation Tester

- Skills
  - expert in test execution and incident reporting
  - especially:
    - IT basics
    - testing basics
    - applying test tools
    - understanding test object
- Tasks
  - review test plan and test cases
  - use test tools and test monitoring tools (e.g. performance measurement)
  - execute and log tests, evaluate and document results and deviations

Test Management

QA Plan nach IEEE 730

1. Purpose
2. Reference documents
3. Management
4. Documentation
5. Standards, practices, conventions, and metrics
6. Software reviews
7. Test
8. Problem reporting and corrective action
9. Tools, techniques, and methodologies
10. Media control
11. Supplier control
12. Records collection, maintenance, and retention
13. Training
14. Risk Management
15. Glossary
16. SQAP Change Procedure and History

nach Bedarf erweiterbar

erster grober Aufriss für Testplan

Aktivitäten der Testplanung

- define overall approach to and strategy for testing
- decide and document test environment
- define test levels, their cooperation, integration, coordination with other activities
- decide on evaluation of test results
- select metrics for monitoring and control of testwork and test exit criteria
- determine form of test documentation, e.g. templates
- write test plan and define responsibilities, timing, amount of tests
- estimate test effort and test cost - for replanning and reestimation

Test Plan nach IEEE 829

1. Test plan identifier
2. Introduction
3. Test items
4. Features to be tested
5. Approach
6. Item pass/fail criteria (test exit criteria)
7. Suspension criteria and resumption requirements
8. Test deliverables
9. Testing tasks
10. Environmental needs
11. Responsibilities
12. Staffing and training needs
13. Schedule
14. Risk and contingencies
15. Approvals

Priorisierung

- Generell: um Ressourcen sinnvoll zu nutzen
- Priorisierungsregel: bei einer frühzeitigen Beendung der Testkampagne muss sichergestellt sein, dass das bestmögliche Testresultat erzielt wurde
- Definition durch Projekt Manager - jeder Testfall wird eine Prio zugewiesen
- Priorisierungkriterien
  - Nach Verwendungshäufigkeit einer Funktion
  - Nach failure probability in der Funktion
  - Risiko (Risk of failure): Kombination der Wahrscheinlichkeit und der Schwere eines failures
  - Nach Sichtbarkeit eines failures für den Endnutzer (speziell bei interakiven Systemen)
  - Nach Priorität der Anforderung
  - Nach Wichtigkeit der Nichtfunktionalen Anforderungen (Quality Characteristics) für den Kunden
  - Nach Kritikalität von Komponenten - aus Architektursicht
  - Nach Komplexität von Komponenten oder Subsystemen
  - Nach Risikoabschätzung von failures für das Gesamtprojekt
  - Wo ein Fehler ist sind auch noch mehr.... (fault locality)

Übliche Testendekriterien

- für Gesamttest oder einzelne Testlevel festlegen
- Hilft Risiken für das Projekt zu minimieren - muss aber vorab definiert werden
- Einige übliche test exit criteria
  - Test coverage
  - Product quality z.B. Restfehler nach Kritikalitätsstufe, failure rate, reliability
  - Residual risk: nicht ausgeführte Tests, nicht behobene Defekte, unvollständige Coverage bzgl Code oder Requirements
  - Economic consraints: Kostenschranke, Projektrisiko, Liefertermine, Marktchancen

(s. auch nächsten Abschnitt)

Test Management

Kosten durch verbleibende Fehler

- entstehen bei nicht durchgeführten Tests oder reduziertn Testaktivitäten
- Kostenarten
  - Direkte Defektkosten (direct defect costs)
    Kosten, die dem Endnutzer oder Kunden entstehen - für die der Hersteller aber u.U. aufkommen muss
  - Indirekte Defektkosten (indirekt defect costs) Kosten, die dem Hersteller entstehen durch Kundenunzufriedenheit etc.
  - Kosten für Defektbehebung (cost for defect correction)
    Kosten, die beim Hersteller anfallen für Fehleranalyse, Korrekturen, Retests etc
- auch: Fehlerrisiko abschätzen - abhängig vom Produkt
- Generelle Erfahrungen:
  - je früher Fehler gefunden werden, desto geringer die Kosten
    denn: je später gerunden, desto mehr abhängige Korrekturen notwendig
  - Fehler, die früh ins System eingebaut werden sind tendenziell teurer
    denn: sie wirken sich auf folgende Entwicklungsphasen aus

Was kostet Testen? Faktoren

Kosten abhängig von einer Reihe von Faktoren:

- Reife (maturity) des Entwicklungsprozesses
- Qualität und Testbarkeit (quality and testability) der SW
- Test infrastructure
- Qualität der Mitarbeiter (quality of employees)
- Quality requirements
- Test strategy

Einflussmöglichkeiten des Testmanager:

- Maturity des Entwicklungsprozesses - nur langfristig
- Testability der SW - abhängig von der Reife des Entwicklungsprozesses - eher langfristig
- Test infrastructure - bei Planung oder auch im laufenden Prozess
- quality of employees - durch Planung und Forderung nach geeigneten Schulungen
- Quality requirements - vorgegeben durch Kunden durch TM nur als Berater für PM/QA
- Test strategy - direkt in der Hand des TM

Vorgehen zur Aufwandsabschätzung bei Tests

- Grob zwei Ansätze:
  - Alle Testtasks identifizieren, task owner oder Experte schätzt Aufwand
  - Abschätzung auf Basis ähnlicher Testkampagnen und daraus abgeleiteter Erfahrungswerte
- Daumenregel: für typische business applications: ca 50% der Gesamtprojektressourcen

Test Management

22 Testorganisation (Test Organization)

23 Testplanung (Test Planning)

24 Kosten- und Wirtschaftlichkeitsaspekte (Cost and Economy Aspects)

25 Definition der Test Strategie (Definition of Test Strategy)
- Präventiver vs. Reaktiver Ansatz (Preventative vs Reactive Approach)
- Analytischer vs. Heuristischer Ansatz (Analytical vs. Heuristic Approach
- Test und Risiko (Testing and Risk)

26 Management von Testaktivitäten (Test Activity Management)

27 Fehlermanagement (Incident Management)

28 Anforderungen an Configuration Management

29 Relevante Standards

Teststrategie allgemein

- definiert die Testziele des Projekts
- und die Mittel zu deren Erreichung
- wichtige Grundlage für Planung, Aufwandsschätzung etc.
- Grundsatz: Kosten für das Testen sollen die Kosten für Defekte und Abweichungen nicht übersteigen
- aber: ganz selten werden geeignete Erfahrungswerte aus früheren Projekten gesammelt
- Unterschiedliche Herangehensweisen:
  - Präventives vs. Reaktives Vorgehen
  - Analytisches vs. heuristisches Vorgehen
- zu berücksichtigen
  - Projekt/Produkt-Risiken

Preventative vs. Reactive approach

- preventative approach
  - Tester werden von Beginn an einbezogen
  - Notwendig (mandatory) für sicherheitskritische Systeme
- reactive approach
  - Testen erst planen und beginnen wenn SW oder System fertiggestellt ist
  - nicht zu empfehlen
  - am ehesten erfolgversprechend: exploratory testing
- Empfehlung:
  - test process should start as early as possible
  - testing should accompany al phases of the project

Analytical vs. Heuristic approach

- Analytical approach
  - test planning on basis of data and their analysis
  - amount and intensity of tests chosen suitable to individual or multiple parameters (cost, time, coverage,...)
- heuristic approach
  - Test planning based on experience of experts or rules of thumb e.g. if no data is available, if know-how is missing
- in der Praxis: oft Mischformen
  - Model-based testing
  - Statistical or stochastic (model-based) testing
  - Risk-based testing
  - Process or standard compliant approaches
  - Reuse-oriented approaches
  - checklist-based (methodical) approaches
  - expert-oriented approaches

Risiko als Steuermechanismus bei Tests

- risk = damage * probability
- Oft schwierig Abzuschätzen - in einigen Standards Grobkategorien bzgl Safety
- Generelle Unterscheidung
  - Projektrisiken = Risiken, dass Produkt zu liefern
    - supplier side risks
    - Resource-related: Personalverfügbarkeit, Probleme bei der Interaktion, politische Querelen
    - technische Probleme: requirements, neue Technologien, schlechte Qualität von Zwischenprodukten
  - Produktrisiken
    - Produkt mit unzureichender Qualität (functional, non-functional)
    - Produkt verursacht Schäden in Einsatzumgebung - auch Safety
- Umgang mit Risiken: Riskmanagement (IEEE 730, 829)
  - Kontinuierliche Bewertung (assessement) der Risiken
  - Priorisierung von identitifizierten Risiken
  - Maßnahmen zur Reduzierung dieser Risiken (mitigation)
- Risk-based Testing

  Info über Risiken geht in die Planung etc. der Testaktivitäten ein

  Bezogen auf Techniken, Umfang, Priorisierung
- Test Priorisierung auf Basis der Risiken

Test Management

Aufgaben des Testmanagement

- Testen in Zyklen - wiederholter Durchlauf des Testprozesses auf unterschiedlichen Ebenen
- Testmanager (einer oder mehrere) verantwortlich für
  - initiating
  - supervising
  - controlling

Planung von Testzyklen

- Basis: Teststrategie im Testplan
- relevante Aspekte für die Planung eines Zyklus
  - Entwicklungsstatus (development status)
  - Testergebnisse (test results)
  - Ressourcen
- Detailplanung: welche test cases wann und durch wen - auch regression tests

Überwachung der Testzyklen

- Messen und Überwachen (measure and monitor) der Testresultate
- Verwendung von zuverlässigen Metriken (reliable, regular, simply measurable)
  - fault-based and failure-based: Anzahl und Kritikalität von gefundenen Problemen
  - test case based: Durchgeführt gegen geplant
  - test object based: z.B. code coverage
  - cost based: Testkosten bisher, erwartete Kosten für Zyklus im Vergleich zu erwarteten Fehlerkosten
- Ergebnisse zusammenfassen in Bericht (test manager) - aber mit exakter Identifizierung von (template s. z.B. IEEE 829)
  - Testobjekten
  - Test progress (planned, run, blocked)
  - incident status (new, open, corrected)
  - risks (new, changed, known
  - outlook (next cycle)
  - assessment (subjective) bezogen auf maturity, release status, trust in product
- Daten auch als Basis für Testendekriterien

Steuern der Testzyklen

- Flexibilität: Änderungen müssen jederzeit möglich sein
- Gründe für Änderungen an Testplan etc.
  - Abweichungen vom Plan
  - kritische Fehler
- Maßnahmen:
  - Neuplanung von Ressourcen (Personen, Equipment,..)
  - Plan ändern - z.B. niedrigpriore Tests weglassen
  - Verlängerung der Testphase (für weitere Tests oder retests)
- Testmanager muss Ursachen und Maßnahmen sauber dokumentieren

Test Management

Fehlermanagement

- Umgang mit Test logs (Testprotokoll)
- Incident Reports (Fehlermeldung)
- Incident Classification (Fehlerklassifizierung)
- Incident Status (Fehlerstatus)

Aufgaben des Testers im Zusammenhang mit test logs

- test log analysis - nach jedem Testlauf, spätestens nach einem Testzyklus
- Sicherstellen, dass gemeldete Abweichungen auch welche sind
- saubere Dokumentation der Fehler
- Nicht: Ursachenanalyse (cause analysis) - Aufgabe des Entwicklers

Incident Reports

- von Testern, Entwicklern und Kunden / Endnutzern
- empfohlen: Datenbank verwenden
- als Basis für Diskussion mit Entwickler
- incident repository als Basis für Testmanager zur Bewertung des Stands
- empfohlen oder erzwungen: einheitliches Format (definiert in Teststrategie)
- wichtig: alle Informationen festhalten, die für Fehlerreproduzierung, Korrektur und retest relevant sind
- Vorgaben z.B. in IEEE 829, IEEE 1044 - oft Anwendungsbereichsspezifisch

Beispielstruktur

- Identification
  - Id
  - Test object
  - version
  - plattform
  - reporting person
  - responsible developer
  - reporting date
- Classification (s. auch Abschnitt Incident Classification)
  - status (aktueller Status und Historie der Fehlermeldung)
  - severity (nach festgelegten Klassen)
  - priority (der Korrektur)
  - requirements (die durch den Fehler verletzt wird)
  - problem source (Dokument oder Ebene in der der Fehlerzustand erzeugt wurde)
- Problem description
  - test case (ganz oder Teile, für Fehlerreproduktion)
  - problem description (vom Tester)
  - comments (von Entwicklern oder anderen)
  - defect correction (Kurzbeschreibung)
  - references (zu relevanten Berichten)

Incident Classification

- Verwendet bei monitoring test process
- auch: als Basis für test process improvement
- severity
  - Abhängig vom Anwendungsbereich / relevanten Standards; oft
    - FATAL: system breakdown, loss of data - kein Release möglich
    - VERY SERIOUS: essential malfuncioning - Verwendbar mit erheblichen Einschränkungen
    - SERIOUS: Abweichungen von der Funktion - Verwendbar mit Einschränkungen
    - MODERATE: kleinere Abweichungen - verwendbar ohne Einschränkungen
    - MILD: kleinere Abweichungen für einige Stakeholder - verwendbar ohne Einschränkung
- fault priority
  - IMMEDIATE
  - NEXT RELEASE
  - ON OCCASION
  - OPEN

Fehlerstatus als Prozess

- oft durchläuft eine Fehlermeldung unterschiedliche Phasen - ev. auch toolunterstützt
- auch: gesteuert durch SW control board oder allgemeiner change control board
- Beispiel: Schema mit Verantwortlichen:
  - NEW durch Tester, Nutzer, etc.
  - OPEN durch Testmanager
  - REJECTED durch Testmanager (nach Analyse oder direkt)
  - ANALYSIS durch Entwickler
  - OBSERVATION durch Entwickler - wenn Fehler nicht reproduzierbar, fehlende Info zu erbringen
  - CORRECTION durch Project Manager
  - TEST durch Entwickler
  - CLOSED durch Tester (und nur Tester!)
  - FAILED durch Tester (zurück zu ANALYSIS)

Test Management

Konfigurationsmanagement und Testen

- configuration management als Baustein für die Reife des Entwicklungsprozesses
- ohne vernünftiges config management:
  - Behinderung der Integration - welche Code-Version wann?
  - Behinderung der Fehlerlokalisierung und -korrektur - Nachvollziehbarkeit von Änderungen nicht möglich, zT auch Fehlerreproduktion schwierig
  - Behinderung von Tests und Testevaluierung - welcher Testversion zu welcher Codeversion? Woher kommen welche Testergebnisse?
- Anforderungen an das Configuration Management:
  - version management
  - configuration identification
  - incident status and change status accounting
  - configuration audits
- Testdaten gehören genauso unter Config Management wie Entwicklungsdaten!

Test Management

Standards für Entwicklung, QA, SW Test

- müssen Testmanagern vertraut sein
- verschiedene Ebenen:
  - company standards
  - best practices
  - quality management standards - z.B. ISO 9000
  - standards for particular industrial sectors - e.g. RTCA-DO 178B
  - SW test standards - z.B. BS 7925-2, IEEE 829, IEEE 1028

Test Management

Zusammenfassung SW Testmanagement

- Rollen im Testprozess und ihre Aufgaben
- Testmanager und Testplanung
- Details zum Testplan, speziell Teststrategie und Testpriorisierung (auch im Prozess)
- Testen und Risiko
- Metriken für Testüberwachung und -steuerung, speziell: Testendekriterien
- Incident Management im Detail
- Testen und Configuration Management
- relevante Standards

Teil VII

Testwerkzeuge

Kursübersicht

- Grundlagen des Software Testens (Fundamentals of Software Testing)
- Testen im Entwicklungszyklus (Testing in the Software Life Cycle)
- Statisches Testen (Static Test)
- Dynamisches Testen (Dynamic Analysis - Test Design Techniques)
- Test Management
- Testwerkzeuge (Test Tools)

Überblick Testwerkzeuge

- Typen von Testwerkzeugen
- Auswahl und Einführung von Testwerkzeugen
- Zusammenfassung

Testwerkzeuge

Typen von Testwerkzeugen

- CAST = Computer-aided SW Testing - Gegenstück zu CASE
- Verschiedene Toolgruppen für verschiedene Phasen im Testprozess
- Klassifikation nach Phase oder Aktivitäten
- Selten: alle Werkzeugtypen in einem Projekt
- Die Kunst: die richtigen sich ergänzenen Werkzeuge für ein Projekt auswählen
- Werkzeugauswahl - abhängig von Plattform, Projekt, Erfahrung
- Problem: immer neue Werkzeuge (Bsp. Eclipse-Plugins) - aber welches taugt?

Tools for Test Management and Control

- Unterstützung für
  - capturing, cataloging, administration of test cases and priorities
  - status tracking - minimum: passed/fail
  - z.T. resource / schedule planning
- Fortgeschritten: Enge Bindung an Requirementsmanagement (Querverweise auf test cases und abgeleiteter Validierungsstatus)
- spezieller: incident management / change management
- gemeinsam mit Entwicklung: configuration management
- Erstrebenswert: hoher Grad an Integration und Durchgängigkeit zwischen Werkzeugen
- auch erstrebenswert: automatisierte Erzeugung von Berichten und Dokumentation
- Beispiele:
  - Imbus TestBench
  - auch: Sammlung von Eclipse-Plugins
- generell oft Plattform- oder Entwicklungsumgebungsspezifisch

Tools for Test Specification

- Spezifikation von precondition, postcondition, test input data, expected results
- Test (data) generator - abhängig von der Testbasis:
  - database-based - Ableitung der Testfälle aus Datenbankschemata oder auch Datenformaten in speziellen Dateien
  - code-based - Testfälle aus Basis der Analyse des source codes - meist aber ohne Testorakel (expected values) – vgl auch white box tests
  - interface-based - Analyse des Interfaces (parameter) und Verwendung von Äquivalenzklassenbildung - speziell: API, GUI - auch ohne expected results
  - specification-based - Voraussetzung: spec formal genug - Bsp.: UML Seq charts, state diag
    Stichwort: model-based testing (MBT)
- Anmerkung: ersetzt nicht die Kreativität des Testers, eher als Ergänzung zu sehen

Tools for static tests

- entsprechend den Kategorien statischer Tests: review support und static analysis
- review support tools
  - Unterstützung für Planung, Durchführung, Auswertung (plan, execute, evaluate)
  - Information und Dokumentation von Reviewsitzungen etc.
  - auch: Online-Checklisten
- static analysis tools
  - Metriktools (oft: Komplexität)
  - Abweichungen von standards, norms, guidelines
  - auch hier: ev. unter Verwendung von checklists
- weitere Gruppe: model-checking tools
  - Voraussetzung: passende formale Modelle

Tools for dynamic tests

- Plattform / Sprachabhängig
- unterschiedliche Ansätze auch abhängig von Testobjekten/-zielen und ihren Schnittstellen
  - Debugger
  - Testtreiber (test driver)
  - simulator
  - test robots – z.B. capture/replay test execution
  - Vergleicher (comparator)
  - dynamic analysis
  - coverage analysis

Tools for nonfunctional tests

- load and performance tests
- monitors
- Spezialwerkzeuge für Securitytests (z.B. virus scanner)

Testwerkzeuge

Toolauswahl allgemein (1)

- zunächst: bewusst machen, welche Werkzeuge man schon verwendet - z.B. inkl. Unixwerkzeuge
- ideal: Werkzeuge, die von Entwicklern verwendet werden und direkt für den Tester Daten lefern können
- Aber: Grenzen von Werkzeugen erkennen
- Potentielle Fallstricke der Automatisierung:
  - "automating chaos just gives faster chaos" (Fewster)
    Die Verbesserung durch Automatisierung hängt ganz stark von der Projektumgebung und Reife des Entwicklungs-/Test-Prozesses ab
  - Automatisierung ist keine Lösung für schlechte Planung
  - späte ad-hoc Automatisierung kann keine Projekte retten

Toolauswahl allgemein (2)

- Empfohlene Reihenfolge von Werkzeugeinführung:
  - incident management
  - configuration management
  - test planning
  - test execution
  - test specification
- Generell berücksichtigen: Lernkurve (learning curve), nicht Wunschproduktivität
- Bei der Werkzeugeinführung berücksichtigen:
  - Kosten
  - Zeitschiene
  - Aufwand gegen Nutzen

Cost Effectiveness (1)

- cost-benefit analysis: bei Vorüberlegungen Kosten und Nutzen abwägen
- Dabei beachten:
  - Kosten: für Auswahl, für Anschaffung (acquisition), Wartung und Administration
  - weitere Kosten für zusätzliche HW, Training
  - auch abhängig von Werkzeugkomplexität!
  - der Gesamtanteil der automatisierbaren (mechanical) Aktivitäten im Lebenszyklus eines test cases hängt stark davon ab, wie oft der Test wiederholt wird! (test execution automation)
  - Berücksichtigen, dass sich das Testvorgehen mit Automatisierung verändert - u.U. werden auch mehr Tests gemacht, als manuell gemacht würden
  - Auch berücksichtigen: einige Tests sind nur automatisiert möglich (z.B. Performance)
  - in welchem Grad ist Verbesserung der SW-Qualität zu erwarten? Wechselwirkung: mehr Kosten für Test kann Kosten für Entwicklung reduzieren

Cost Effectiveness (2)

- Beobachtungen:
  - kreative Testaufgaben können mit Tools unterstützt werden ⇒ Verbesserung der Testqualität
  - mechanische Testausführung reduziert Testaufwand oder erlaubt mehr Tests
    Aber: mehr Tests bedeutet nicht zwangsläufig bessere Tests
  - Beides führt ohne gute Testprozeduren oder systematische Verwendung von Methoden nicht von selber zu geringeren Kosten

Tools Selection - Vorgehen

- 5 Schritte
  1. Anforderungen für die Anwendung der Werkzeuge identifizieren
  2. Marktrecherche (Überblick über geeignete Kandidaten)
  3. Tool Demo mit dem Ziel einer Auswahlliste
  4. Evaluierung der Tools auf der Vorauswahlliste
  5. Auswertung der Eval-Ergebnisse und Auswahl
- Generelle Auswahlkriterien (für Schritt 1) - ev. mit unterschiedlichen Gewichten
  - Qualität der Interaktion mit Testobjekt
  - Vorkenntnisse der Tester über Tool oder darin verwendete Methoden
  - Integrierbarkeit in bestehende Entwicklungsumgebung
  - Plattform auf der das Tool laufen soll
  - Toolhersteller Service, Zuverlässigkeit, Marktstellung
  - Lizenzbedingungen, Preis, Wartungs/Update-Kosten
- Marktuntersuchung und Vorauswahl - ev. parellel zu Schritt 1
- Evaluierung (ev. auch gemeinsam mit Herstellern) oft an Beispielen
  - Arbeitet das Werkzeugen wie versprochen mit Testobjekt und Entwicklungsumgebung?
  - Sind alle auswahlrelevanten feature und Qualitätsaspekte real vorhanden?
  - Ist der Hersteller Support ausreichend gut - auch bei nicht-standard Anfragen?

Tool Introduction

- Vorher: Auswahl durch geeignetes Kernteam
- Nun: Verifikation der Auswahl durch geeignetes Pilotprojekt
- aber: nicht von den Personen, die die Auswahl getroffen haben!
- Ziel: weitere Erkenntnisse und Detailerfahrungen mit dem Tool (vor Flächeneinführung)
  - notwendige Änderungen an den bisherigen Prozessen
  - Bedarf für Training
  - Identifizierung von neuen Regeln und Konventionen im Umgang mit dem Tool
  - ev. auch: Testbibliotheken (test libraries) für typische Testanteile
  - Notwendig: strong commitment of tool users and stakeholders
- Erfolgsfaktoren (Success factors)
  - Schrittweise Einführung
  - Integration von Tools in die bestehenden Prozesse
  - Einplanen und Durchführen von user training und continuous coaching
  - allgemein zugängliche Regeln und Guidelines für die Werkzeugverwendung
  - Erfahrungen sammeln und für alle sichtbar dokumentieren (tricks, hints, FAQ, HowTo)
  - monitor tool acceptance
  - gather and evaluate cost-benefit data

Erfolgreiche Tool-Einführung in 6 Schritten

1. Execute a pilot project
2. evaluate pilot experiences
3. adapt processes and implement rules for usage
4. train users
5. introduce tool in steps
6. offer accompanying coaching (bindet aber Personal....)

Im Kern: Berücksichtigung der menschlichen Eigenschaft, Neues erst mal abzulehnen

Testwerkzeuge

Zusammenfassung Testwerkzeuge

- Tools für jede Phase im Testprozess zur Verbesserung der Prozesse und mittelbar der Qualität
- Testautomatisierung nur bei reifen Testprozessen erfolgversprechend
- Toolauswahl muss vorsichtig und sorgfältig erfolgen
- Einführung von Werkzeugen nur mit Information, Training, Coaching – Ziel: Akzeptanz