Advanced Software Engineering 2 - FS21

Pascal Brunner - brunnpa7

Inhaltsverzeichnis

Kapitel 1

Grundlagen des Software Testens

1.1 Begriffe und Motivation

Definition Testen: *Der Prozess der aus allen Aktivitäten des Lebenszyklus besteht (sowohl statisch als auch dynamisch), die sich mit der Planung, Vorbereitung und Bewertung eines Softwareproduktes und dazugehöriger Arbeitsergebnisse befassen. Ziel des Prozesses ist sicherzustellen, dass diese allen festelegten Anforderungen genügen, dass sie ihren Zweck erfüllen und etwaige Fehlzustände zu finden*

Fehlerwirkung / Fehlfunktion / äussere Fehler / Ausfall (engl. failure): Ein Ereignis in welchem eine Komponente od

Fehlerzustand (engl. fault, defect, bug): *Defekt (innerer Fehlerzustand) in einer Komponente oder einem System, der eiene geforderte Funktion des Produkts beeinträchtigen kann, z.B. inkorrekte Anweisung oder Datendefinition*

Fehlhandlung (engl. Error): *Die menschliche Handlung, die zu einem falschen Ergebnis führt (nach IEEE 610)*

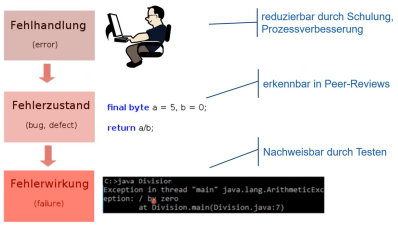

Abbildung 1.1: Abbildung der Fehlerbegriffe

Fehlermaskierung (engl. defect masking): *Ein Umstand, bei dem ein Fehlerzustand die Aufdekcung eines anderen verhindert (nach IEEE 610)*

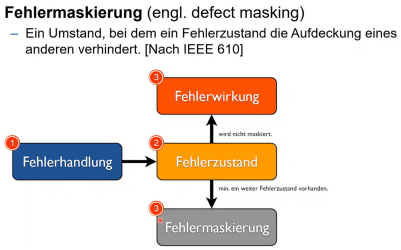

Abbildung 1.2: Abbildung der Fehlermaskierung

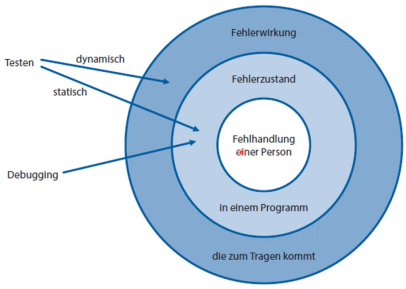

Abbildung 1.3: Zusammenhang der Fehler

False-positive Result: Testergebnis zeigt Fehlerwirkung, obwohl der Fehlerzustand bzw. die Ursache für die Fehlerwirkung nicht im Testobjekt liegt

false-negative Result: Testergebnis zeigt keine Fehlerwirkung, obwohl die Tests diese hätten aufdecken sollen
→ Bei jeder Auswertung von Testergebnissen ist somit zu beachten, ob eine der beiden Möglichkeiten vorliegt

true-positive result: Fehlerwirkung durch den Testfall aufgedeckt

true-negative Result: erwartetes Verhalten bzw. Ergebnis des Testobjekts mit dem Testfall nachgewiesen

1.1.1 Testbegriffe

Um den Defekt zu korrigieren muss der Defekt lokalisiert werden. Bekannt ist die Wirkung aber nicht die genaue Stelle.
Fehlerbereinigung, Fehlerkorrektur (engl. Debugging): Debugging und Testen sind vers. Dinge:

- **Dynamische Tests** können Fehlerwirkung zeigen, die durch Fehlerzustände verursacht werden
- **Debugging** ist eine Entwicklungsaktivtität, die die Ursache (den Fehlerzustand) einer Fehlerwirkung identifiziert, analysiert und entfernt

Ziele des Testens

- Qualititative Bewertung von Arbeitsergebnisse wie Anforderungsspezifikation, User Stories, Design und Programmtext
- Nachweiss, dass alle spez. Anforderung vollständig umgesetzt sind
- Vertrauen in die Qualität
- Höhe des Risikos bei mangelnder Qualität der Software kann durch Aufdeckung von Fehlerwirkungen verringert werden

Unsystematischer Test

- Laufversuch: der Entwickler testet

  Entwickler übersetzt, bindet und startet ein Programm

  Läuft das Programm nicht oder sind Ergebnisse offensichtlich falsch wird Debugging betrieben

  Der Test ist beendet wenn das Programm läuft und Ergebnisse vernünftig aussehen

- Wegwerf-Test: Testen ohne Systematik

  Jemand probiert das Programm mit vers. Eingabedaten aus

  Fallen Ungereimtheiten auf, wird eine Notiz gemacht

  Der Test endet, wenn der Tester findet, es sei genug getestet

1.1.2 Systematischer Test

- Test ist geplant
- Programm wird gemäss Testvorschrifft ausgeführt
- Ist-Resultat wird mit Soll-Test überprüft
- Testergebnisse werden dokumentiert
- Fehlersuche und -behebung erfolgen separat
- Nicht bestandene Test werden wiederholt
- Test endet, wenn vorher definierte Testziele erreicht sind
- Die Testspezfiikation wird laufend aktualisiert

Ziele des systematischen Testens

- Reproduzierbarkeit
- Planbarkeit
- Wirtschaftlichkeit
- Risiko- und Haftungsreduktion

1.2 Testartefakten

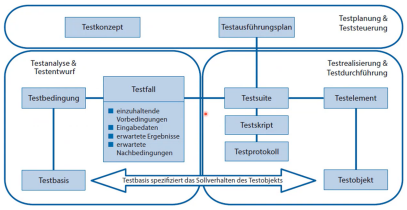

Abbildung 1.4: Uebersicht der Testartefakten und ihre Beziehungen

1.3 Aufwand

- Programm vollständig zu testen ist in der Praxis nicht möglich
- 25-50 Prozent des Entwicklungsaufwands
- Testintensität und umfang in Abhängigkeit mit dem Risiko und Kritikalität
- 2/3 des Testaufwands können auf Komponententests entfallen
- Immer mit beschränkter Ressourcens

1.4 Grundsätze

- Möglichst frühzeitig alle Beteiligten beiziehen
- Beteiligen sich Tester an der Prüfung der Anforderung können Unklarheiten und Fehler in der Arbeitsprodukten aufgedeckt und behoben werden
- Die enge Zusammenarbeit von Testern mit Systemdesiger kann das Verständnis für jede Partei für das Design und des Tests erheblich verbessern
- Arbeiten Entwickler und Tester während der Codeerstellungs zusammen, kann das Verständnis beidseitig verbessert werden
- Wenn Tester die Software vor deren Freigabe verifizieren und validieren können weitere Fehlerzustände erkannt und behoben werden

1. Testen zeigt die Anwesenheit von Fehlerzuständen
2. Vollständies Testen ist nicht möglich
3. Frühes Testen spart Geld und Zeit
4. Häufung von Fehlerzustände → Taucht in einem Modul ein Fehler auf, gibt es eine hohe Wsk dass noch weitere Fehler sich befinden
5. Vorischt vor dem Pestizid-Paradox → nur wiederholen bringen keinen Mehrwert, Testfälle sind zu prüfen, zu aktualisieren und zu modifizieren
6. Testen ist kontextabhängig
7. Trugschluss: Keine Fehler bedeutet ein brauchbares System

Kapitel 2

Testprozess

Ein Testprozess wird in der Regel folgende Aktivitäten umfassen (ISO-Norm 29119-2):

- Testplanung
- Testüberwachung und -steuerung
- Testanalyse
- Testentwurf
- Testrealisierung
- Testdurchführung
- Testabschluss

→ Diese Aktivitäten werden z.T. zeitlich überlappend oder parallel ausgeführt. Der Testprozess ist für jede Teststufe geeignet zu gestalten (Tailoring für ein Projekt)

2.1 Testplanung

- Umfangreiche Aufgabe sollte so früh wie möglich begonnen werden
- Aufgaben und Zielsetzung der tests müssen festelegt werden, genau wie die Ressourcen
- Entsprechende Festlegungen im Testkonzept
  - Testziele
  - Teststrategie
  - Testaktvitäten
  - Ressourcen
  - Testbedingungen und Testbasis
  - Metriken
  - Risiken
- Teststrategie bildet der rote Faden

2.1.1 Testüberwachung und Teststeuerung

- Die Testüberwachung und Teststeuerung umfasst

  fortwährende Beobachtung der aktuell durchgeführten Testaktvitäten im Vergleich zur Planung

  Berichterstattung der ermittelten Abweichungen und die Durchführung der notwendigen Aktivitäten um die Ziele zu erreichen

- Basis für Testüberwachung und -steuerung sind Endekriterien für jeweilige Testaktiväten und -aufgaben

2.1.2 Testanalyse

Bei der Testanalyse geht es darum, zu ermitterln, was genau zu testen ist

- Testbasis prüfen
- Dokumente analysieren
- Testobjekt selbst prüfen
- Berichte heranziehen
- Grundlage ist die Testbasis
- Priorisierung der Testbedingungen
- bidirektionale Rückverfolgung sollte sichergestellt werden (traceability)

2.1.3 Testentwurf

Bei Testentwurf geht es darum, festzulegen, wie getestet wird

- Spezifikation von abstrakten und konkreten Testfällen
- Identifizierung benötigter Testdaten
- Ausgangssituation (Vorbedingung)
- Randbedingungen
- Ergebnisse bzw. welches Verhalten erwartet wird
- Sollergebnis

2.1.4 Testrealisierung

Abschliessende Vorbereitung aller notwendigen Aktivitäten

- Erstellung der Testmittel
- Testrahmen programmiert und Testumgebung installiert
- Abstrtakte Testfälle sind zu konkretisieren
- Testfälle sind zweckmässigerweise zu Testsuiten gruppiert
- automatisierte Testskripts

2.1.5 Testdurchführung

Umfasst die konkrete Ausführung der Tests und deren Protokollierung

- Ausführung von Testabläufen unter Einhaltung des Testplans
- Nachvollziehbarkeit und Reproduzierbarkeit
- Vergleich Ist-Soll
- Fehlerwirkungen oder Abweichungen festhalten

2.1.6 Testabschluss

- letzte Aktivität im Testprozess
- Für Ermittlung von Metriken sollen Testwerkzeuge eingesetzt werden
- unterschiedliche Zeitpunkte für Testabschluss
- Testabschlussbericht ist zu erstellen
  Fasst alle Testaktvitäten und -ergebnisse zusammen
  wird allen Stakeholdern zur Verfügung gestellt
  Testmittel sind zu archivieren

2.2 Psychologie des Testens

- Irren ist menschlich
- Soll der Entwickler sein eigenes Programm testen?
  Blindheit gegenüber eigenen Fehler
  hat hingegen keine Einarbeitungszeit
- Drittperson / Tester
  ist unvoreingenommen
  Einarbeitung notwendig
  Test-Know-how notwendig, bringt ein Tester jedoch mit
- Stufen der Abhängigkeit (von niedrig nach hoch)
  Entwickler selbst
  Kollegen des Entwicklers
  Personen anderer Abteilung
  Person anderer Organisation
- Aufteilung ist produkt- bzw. projektabhängig
- richtige Mischugn und Ausgewogenheit zwischen unabhängiger Tests und Entwicklertests
- Fehlerwirkungen mitteilen
- Reproduzierbarkeit ist wichtig
- Eindeutige Anforderungen, präzise Spezifikation
- Förderlich für die Zusammenarbeit zw. Tester und Entwickler ist die gegenseitige Kenntnis der Aufgaben

Kapitel 3

Testen im Softwareentwicklungslebenszyklus

3.1 Sequentielle Entwicklungsmodelle

Eines der bekanntestens Modellen ist das Wasserfallmodell

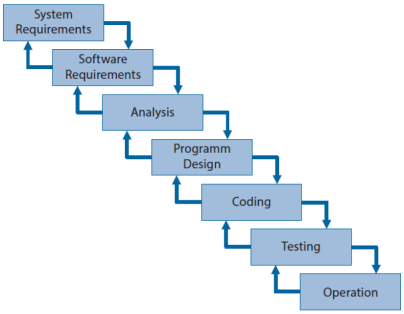

Abbildung 3.1: Das Wasserfallmodell

Ebenfalls zu den sequentiellen Entwicklungsmodellen, gehört das V-Modell

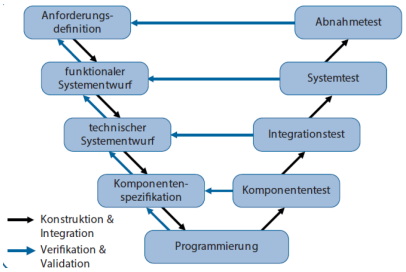

Abbildung 3.2: Das V-Modell

Beim V-Modell trennt man den Konstruktions, mit den Testaktivitäten, sind jedoch gleichwertig aufzufassen (linke und rechte Seite). Dahingehend spricht man von arbeitsteiligen Teststufen, wobei jede Stufe 'gegen' ihre korrespondierende Entwicklungsstufe testet.

Dabei gibt es ein Sprachgebraucht nach ISTQB für das V-Modell

- **Verifikation:** Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind ('Are we doing the thing right?')

  Alternative: Verfikation = formaler Korrektheitsbeweis

- **Validierung:** Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spez. beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind ('Are we doing the right thing?')

Alternative: Validierung = informelle Überprüfung

→ In der Praxis beinhaltet jeder Test beide Aspekte, wobei der validierende Teil mit steigender Teststufe zunimmt

Spricht man von testen innerhalb des Softwareentwicklungslebenszyklus:

- Analyse und Entwurf der Tests während der Entwicklungsaktivtitäten
- Tester sollen im Review Prozess für Requirements oder Architektur als Stakeholder miteingebunden werden

3.2 Iterative und inkrementelle Entwicklungsmodelle

Die wohl bekannteste Form der iterativen Entwicklungsmodellen ist SCRUM.

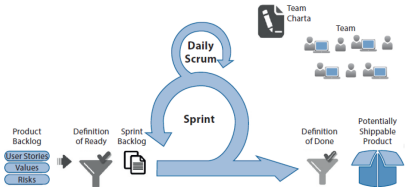

Abbildung 3.3: Scrum

Dabei gibt es die **Testpyramide** nach Cohn und ist eine Hilfe für das Scrum-Team für die Überprüfung, ob die vorhandenen Testfälle angemessen und über sämtliche Teststufen verteilt sind.

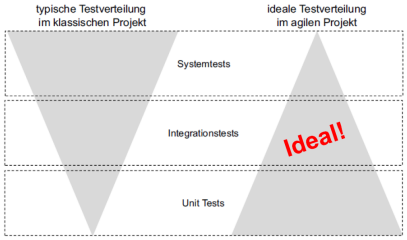

Abbildung 3.4: Die Testpyramide nach Cohn

3.2.1 Testmanagement in Scrum

- Ein Scrum-Team ist ein sich selbst organisierendes, interdisziplinäres Team
- Es gibt keinen Projektleiter, sondern vertraut darauf, dass ein Team sich selbst steuert
- Team ist gemeinsam für alle Arbeiten zuständig
- Programmierung und Testen ist nicht getrennt
- Testmgmt-Aufgaben existieren natürlich
- organisatorische Aufgaben fallen zum Scrum Master
- Leitungsaufgaben im Rahmen der Sprint-Planungspraktiken abgedeckt
- Testaufgaben entweder explizit über eigene Tasks oder implizit mit der Done-Kriterien anderer Aufgaben überwacht
- Testfortschritt und Testergebniss durch Continuous Integration ist hochautomatisiert
- CI kann durch Continuous Delivery erweitert werden → Wenn Tests fehlerfrei durchlaufen, wird es deployed
- klassische Testmanager wird überflüssig, jedoch nicht bei testfachlichen Aufgabenbereichen (Teststrategie und inhaltliches Planen)

- Gemäss Scrum würden diese Aufgaben allensamt mit dem Team durchgeführt
- Ein Teammitglied mit Testexpertise wird benötigt
- Mind. eine Person 'hauptamtlich' für das Testen zuständig sein
- Diese Person berät dann auch der PO bzgl. Produktqualität und Produktfreigabe
- Spricht nichts dagegen diese Person auch Testmanager im Scrum Team zu nennen
- auch exertne Testspezialisten sind öglich
- Auch in Scrum sind grundlegende klassische Testtechniken unverzichtbar
- Alle sollten geschult werden
- Scrum Master oder Testmanager müssten dafür sogar, dass diese Techniken umgeseztt werden

3.3 Softwareentwicklung im Projekt- und Produktkontext

Die Anforderungen an Planung und Nachvollziehbarkeit von Entwicklung undTest sind in unterschiedlichen Kontexten verschieden. Mögliche Punkte können eine Rolle spielen:

- Geschäftsprioritäten
- Art des Produktes
- Markt- und technisches Umfeld
- Identifizierte Produktrisiken
- Organisatorische und kulturelle Aspekte

→ Je nach Einsatzgebiet, sollen die Vorgehensmodelle angepasst werden (Tailoring)

3.4 Teststufen

Beim Testen kann und muss das zu testende System, seine Eigenschaften und sein Verhalten auch auf den versch. Ebenen der Architektur, von den elemanteren Einzelkomponenten bis zum Gesamtsystem, betrachtet und geprüft werden.

Die Testaktivität einer solchen Ebene werden dabei als **Teststufe** bezeichnet. Jede Teststufe ist eine **Instanz des Testprozesses**. Nachfolgend werden die einzelnen Stufen im Detail betrachtet.

3.4.1 Komponententest

Komponenten werden isoliert getestet → Testen im Kleinen

- Das Testobjekt beim Komponententest ist eine Komponente (Unit, Modul, Subsystem)

  Komponente sind ein Teil einer Applikation, bspw. eine Klasse oder Prozedur

  Keine Vorgabe über die Grösse einer Komponente

- Im Komponententest wird die Komponente an den Schnittstellen gegen die Spezifikation und das Softwaredesign getestet.

  es ist eine Komponente-Spezifikation erforderlich!

- typische Testobjekte
  - Komponenten, Klassen(-verbund)
  - Programme
  - Datenumwandlung / Migrationsprogramme
  - Datenbankmodule
- Die Komponente sollte möglichst istoliert getestet werden
- Die Schnittstelle einer Komponente ist i.d.R. eine Programmierschnittstelle
- Wird ein Defekt gefunden, wird das i.d.R. der Komponente zugeordnet.

Testziele

- Test der Funktionalität
  - Berechnungsfehler
- Test auf Robustheit
  - Durch Negativtests
- Test der Effizienz
  - Speicherverbrauch
  - Antwortszeiten
- Test auf Wartbarkeit (mittels statischer Analyse)
  - Code-Kommentare
  - Numerische Konstante

Testbasis:

- Anforderung an die Komponente
- detaillierter Softwaredesign
- Code

Teststrategie

Entwurfskriterien Testbarkeit

- Die Isolierbarkeit einer Komponente ist eine Voraussetzung für Komponententests
- Isolierbare Komponenten entstehen bei Entwurf nicht zwangsläufig - die Isoliebarkeit muss aktiv im Entwurf herbeigeführt werden
- Empfehlung: Testbarkeit sollte bei Entwurfsreviews mit einbezogen werden

Test-First Ansatz

- Zuerst alle Testfälle implementieren und anschliessend produktiver Programmcode
- Stammt aus Extreme Programming (agile Methode)
- Vorteile
  - QS der Anforderung
  - Automatisierung spart Aufwand
  - Test verliert den negativen beigeschmack
  - Testfälle werden dokumentiert und sind reproduzierbar
- → In der Praxis stellt eine unvollständige Komponentenspezifikation ein grosses Problem dar!

3.4.2 Integrationstest

Bildet die Brücke zwischen den Komponenten und dem Systemtest. Wobei als Integration die Verknüpfung der Komponenten zu grösseren Gruppen verstanden wird.

Der dazugehörige Integrationstest hat das Ziel, Fehlerzustände in den Schnittstellen und im Zusammennspiel zwischen integrierten Komponenten aufzudecken

Testbasis

- Softwaredesign
- Systemdesign
- Systemarchitektur
- evtl. Workflows oder Use Cases

Testobjekt

- Einzelbausteine zu grösseren Einheiten zusammenbauen
- Alle beteiligten Komponenten (inkl. Subsysteme, Externe Systeme und Datenbanken)

Testumgebung

- Analog wie Komponententest
- Zusätzliche Monitore
  Mitschreiben von Datenbewegungen
  Standardmonitore für Protokolle

Testziele

- Schnittstellen- und Protokollfehler aufdecken
  inkompatible Schnittstellenformate
  Untersch. Interpretation der übergegeben Daten
  Timing-Problem: Daten werden richtig übergeben aber zum falschen Zeitpunkt

Integrationsstrategien

Top Down

- Der Test beginnt mit der Hauptkomponente
- Vorteil: Keine Testtreiber notwendig
- Nachteil: Noch nicht integrierte Komponenten müssen durch Platzhalter ersetzt werden

Bottom Up

- Der Test beginnt mit der untersten Komponente
- Vorteil: Keine Platzhalter notwendig
- Nachteil: Übergeordnete Komponenten müssen durch Testtreiber simuliert werden

Ad-Hoc

- Die Komponenten werden bspw. in der Reihenfolge ihrer Integration fertiggestellt
- Vorteil: Fertiggestellte Komponenten können zeitnah integriert werden

- Nachteil: Sowohl Testtreiber als auch Platzhalter sind erforderlich

Backbone-Integration

- Es wird eine Programmskette (Backbone) erstellt, in das schrittweise die zu integrierende Komponente eingehängt werden

  Continuous Integration (CI) kann als moderne Realisierung dieser Integrationsstrategien gesehen werden

- Vorteil: Komponenten können in beliebiger Reihenfolge integriert werden
- Nachteil: Ein unter Umständen aufwendiger Backbone oder eine CI-Umgebung muss erstellt und gewartet werden

Big Bang

- Der Test beginnt mit dem vollintegrierten System
- Vorteil: Es sind keine Testtreiber und Stubs notwendig
- Nachteil: Schwierige Fehlersuche - alle Fehlerwirkungen treten geballt auf

3.4.3 Systemtest

Test des Gesamtsytems → Kundensicht statt Entwicklersicht!

- Test des Zusammenspiels aller integrierten Systemkomponenten
- Test in produktionsnaher Testumgebung
- Achtung: der Begriff System ist keinenfalls eindeutig definiertt

Testbasis

- Alle DOkumenten die das Testobjekt auf Systemebene beschreiben wie Anforderungsdokumente, Spezifikation, Benutzungshandbücher etc.

Testobjekt und Testumgebung

- Mit abgeschlossenem Integrationsstest liegt das komplett zusammengebaute Softwaresystem vor
- Systemgrenzen definieren

  Vor Systemtestbgeinng (besser vor Projektbeginn) ist diese Fragen genaustens zu kläre

Testziele

- Validieren, ob und wie gut das fertige System die gestellten funktionalen und nicht-funktionalen Anforderungen erfüllt
- Fehler und Mängel aufgrund falsch, unvollständig oder im System widersprüchlich umgesetzter Anforderungen aufdecken
- Undokumentierte oder vergessenee Anforderungen identifizieren
- Prüfung der spezifizierten Systemeigenschaften
- Testen der nicht-funktionalen Anforderungen

Probleme

- Testaufbau ist aufwändig
- (leicht vermeidbare) Fehler halten den Systemtest immer wieder auf
- Die Fehlerursache ist aufwändig zu finden
- Fehler können Schaden an der Betriebsumgebung anrichten (bspw. bei angeschlossenen Geräte)

⇒ Um Fehler zu finden ist der Systemtest der ungeeigeste Test. Man sollte die Fehler früher finden.

3.4.4 Abnahmetest

Spezieller Systemtest → Abnahme des Gesamt- oder Teilsystems durch Auftraggeber

Mögliche Resultate

- Abnahmebescheinigung
- Nachbesserungsforderungen
- Rückgabe bzw. Minderung

mögliche Formen

- Test auf vertragliche Konformität
- test der Benutzeraktzeptanz
- Akzeptanz durch den Systembetreiber
- Feldtest

vertragliche Prüfung

- Der Kunde führt eine vertragliche Abnahme durch
- Basis der Ergebnisse entscheidet der Kunde ob das bestellte System den vertrg. Anforderungen entspricht
- Testkritieren sind im Entwicklungsvertrag beschriebenen Abnahmekriterien (können auch Normen oder gesetzliche Vorgaben sein)
- Abnahmeumgebung ist normalerweise die produktivumgebung des Kunden

Test auf Benutzerakzeptanz

- User Acceptance Test (UAT)

  Akzeptanz jeder Anwendergruppen sicherzustellen

  Meinungen bzw. Bewertungen der Benutzer einholen

  Resultate sind oft nicht reproduzierbar
- mögliches Vorgehen: iterative-inkrementelle Entwicklung - Prototypen frühzeitig den Anwendern vorstellen

Akzeptanz durch den Systembetreiber

- Tests auf Passung in bestehenden IT-Infrastrukturen

  Backup, Wiederanlauf

  Benutzerverwaltung

  Aspekte der Datensicherheit
- Deployment-Test
- Test der Installierbarkeit

  Die für dieInstallation zugesicherten Installationsumgebung werden getestet

  virtuelle Umbgebung können den Aufwand zur Bereitstellung der nötigen Testumgebung deutlich reduzieren
- Update-Test

  Ähnlich Deployment Test

  Mögliche Ausgangsversionen sind zu berücksichtigen

- Test auf Deinstallierbarkeit

Feldtest

- Bei Standardsoftware werden stabile Vorversion an einen ausgewählten Benutzerkreis ausgeliefert

  Alphatest

  Betatests

  Release Candidate

- Benutzer-Feedback muss organisiert sein

  User to Supplier

  Suppplier to User

3.5 Testarten

Folgende grundlegende Testarten lassen sich unterscheiden

- Funktionale Tests und nicht-funktionale Tests
- Anforderungs- und strukturbasierte Tests

Wobei sich der Fokus und die Ziele je nach Stufe variieren. Dementsprechend kommen untersch. testarten in unterschd. Intesität zur Anwendung

3.5.1 funktionale Tests

- Funktionalität beschreibt 'was' das System leisten soll (SOLL-Zustand)
- Funktionalität wird vom System, von einem Teilsystem oder einer Komponente geliefert
- Testbasis oder Referenz sind: Anforderungsspezifikation, Use Case, User Stories oder auch funktionale Spezfiikation
- funktionaler Test prüft das von aussen sichtbare Verhalten der Software (vgl. Blackbox Verfahren)
- Verwendung von spezifikationsbasierten Testentwurfsverfahren, um Testendekriterien und Testfälle aus der Funktionalität der SOftware /System herzuleiten

3.5.2 nicht funktionale Tests

- nicht-funktionale Test prüft anhand von Software- und Systemmerkmalen 'wie gut' das System arbeitet
- Grundlagen gemäss Qualitätsmodelle (bspw. ISO 25010)

  Lasttest

  Performancetest

  etc.

3.5.3 Anforderungsbezogener und strukturbezogener Test

- Anforderungsbezogenes Testen nutzt als Testbasis Spezfiikationen des extern beobachtbaren Verhaltens der Software

  Vorwiegend im System- und Abnahmetest eingesetzt

- strukturbezogenes Testen nutzt als Testbasis zusätzlich die interne Struktur bzw. Architektur der Software

  Vorwiegend im Komponenten- und INtegrationstest eingesetzt

  manchmal in höheren Stufen als Ergänzung

3.6 Test nach Änderung und Weiterentwicklung

- Jedes Softwaresystem bedarf über die Dauer seiner Nutzung gewisser Korrekturen und Ergänzungen
- In diesem Zusammenhang wird von Softwarewartung und Softwarepflege gesprochen
- Auslöser für die Änderungen eines Sofrwareprodukts sind die Korrektur von Fehlerzuständen oder die geplante Änderungen oder Ergänzungen einer Funktion

3.6.1 Testen nach Softwarewartung und -pflege

- Falls Fehlerwirkung bewiesen, muss dieser beseitigt werden (→ Fehlernachtest)
- Bei bereits getesteten Funktionalitäten muss bewiesen werden, dass kein Fehler vorliegt (→ Regressionstest)

  Sind auch durchzuführen, wenn sich die Softwareumgebung ändert
- Nach- und Regressionstest werden oft mehrfach ausgeführt und müssen wiederholbar sein (Kandidaten für Testautomatisierung)
- Regressionstest umfassen funktionale, nicht-funktionale wie auch strukturelle Tests (auf allen Teststufen)

Kapitel 4

Statischer Test

Testen von Software-Entwicklungsartefakten, ohne diese auszuführen, bspw. durch Reviews oder statische Analyse
Reviews

- Manuelle Prüfungen durch eine oder mehr. Personen
- Menschliche Analyse- und Denkfähigkeit wird genutzt, um komplexe Sachverhalte zu prüfen und zu bewerten
- Kann bei allen Dokumenten durchgeführt werden, die während des Prozesses erstellt oder verwendet werden

Statische (Code-) Analyse

- automatisierte Prüfung durch entsprschende Werkzeuge
- Nur bei Dokumenten mit formaler Struktur (bspw. Programmtext, UML-Diagramme etc.)

4.1 Was kann analysiert und geprüft werden?

- Analyse des Testobjekts (Dokumente, Code etc.) durch intensive Betrachtung
- **Ziel:** Ermittlung von Fehlerzuständen (Defekten) im Dokument bzw. Code

  Verstösse gegen Spezifikationen, oder einzuhaltende Standards, Fehler in Anforderungen / im Design, falsche Schnittstellenspezifikation

  Nachweis der Verletzung von Projektplänen

  Ergebnisse der Untersuchungen werden darüber hinaus dazu benutzt, den Entwicklungsprozess zu optimieren
- **Grundidee:** Prävention

  Fehler(zustände) und Abweichungen sollen so früh wie möglich erkannt werden

4.2 Vorgehen

Bei nicht-Werkzeuggestützten Verfahren → menschliche Analyse- und Denkfähigkeit verwenden. Dies erfolgt durch intensives Lesen und nachvollziehen, wobei Untersch. Verfahren, u.a. Review (von informell bis sehr formal) Entscheidung wann welches zum Tragen kommt, hat unterschiedliche Faktoren.

- Softwareentwicklungslebenszyklus-Modell
- Reife des Entwicklungsprozess
- Komplexität des zu überprüfenden Dokuments
- Gesetzliche oder regulatorische Anforderungen und / oder die Notwendigkeit eines Prüfnachweises (Audit-Trail)

4.3 Reviewprozess

Folgende Hauptaktvitäten gibt es bei einem Review

1. Planung
2. Initiierung, Kick-Off
3. Indiv. Review
4. Diskussion der Befunde
5. Bericht und Fehlerbehebung

4.3.1 Planung

- Mgmt entscheidet welche Dokumente nach welcher Art einem Review unterzogen werden sollen
- entsprechender Aufwand ist einzuplanen
- Eingangs- und Austrittskriterien sind festzulegen
- Manager wählt Person für den Review aus und stellt ein Review-Team zusammen

4.3.2 Initiierung (Kick-Off)

- Versorgung aller benötigten Informationen
- Überprüfung ob Eingangsbedingung erfüllt sind
- Schriftliche Einladung oder sofortiges erstes Treffen des Reviewteams
- Neben dem Arbeitsergebnis, müssen den beteiligten Personen weitere UNterlagen zur Verfügung gestellt werden:

  Dokumente, die herangezogen werden müssen um Review durchzuführen → Auch Basisdokumente (Baseline) bezeichnet

  Prüfkriterien (bspw. mit Checkliste) festlegen

  Sofern Vorlagen zur Protokollierung vorhanden, vorlegen

4.3.3 Individuelles Review (indiv. Vorbereitung)

- beteiligte Personen müssen sich indiv. auf die Sitzung vorbereiten
- Reviewer setzen sich intensiv mit dem zu prüfenden Dokument auseinander
- Erkannte potentielle Fehler(zustände), Empfehlungen, Fragen oder Kommentare werden notiert

Beispiele für versch. indiv. Vorgehensweise:

- Keine Vorgaben: ad-hoc-Vorgehen
- Strukturiertes Lesen: Vorgehen basiert auf Checklisten
- Strukturierte Richtlinie: Vorgehen unter Nutzung von Szenarien und Probeläufen → Dry Runs
- Prüfung aus einer Perspektive: Rollenbasiertes und perspektivisches Vorgehen

4.3.4 Diskussion der Befunde (Reviewsitzung)

- Nach der indiv. Vorbereitung werden die Ergebnisse zusammengetragen und diskutiert
- Form: Reviewsitzung oder firmeninternen Onlineforum
- (potentielle) Abweichungen und Fehler(zustände) werden besprochen und näher analysiert
- Jedes Merkmale und das dazugehörige Ergebnis werden im Rahmen der Befundanalyse dokumentiert
- Reviewteam gibt eine Empfehlung über die Annahme (akzeptiert, Überarbeitung erforderlich, nicht akzeptieren) des Arbeitsergebnis ab
- Best-Practices

  max 2h Sitzung

  Moder hat das Recht, abzusagen oder abzubrechen

  Resultat und nicht Autor im Mittelpunkt

  Moderator darf nicht als Reviewer tätig sein

  Allg. Stilfragen dürfen nicht diskutiert werden

  Diskussion zur Lösung ist nicht Aufgabe des Reviewteams

  Jeder Reviewer muss die Gelegenheit haben, seine Befunde angemessen präsentieren zu können

  Ein Konses der Reviewer ist anzustreben und zu protokollieren

  Befunde nicht in Form von Korrekturanweisungen formulieren

  Befunde sind zu gewichten

4.3.5 Bericht und Fehlerbehebung

Ist die abschliessende Aktivität

- Oftmals beinhaltet das Protokoll der Reviewsitzung alle erforderlichen Informationen
- Normalerweis behebt der Autor die Fehler(zustände)
- Die Reviewergebnisse können in Abhängigkeit von der Reviewart und dem Grad der Formalität stark variieren

4.4 Rollen und Verantwortlichkeiten

4.4.1 Management

- Entscheidet über Durchführung von Reviews und deren Art
- Wählt die Prüfobjekte und die zu verwendenden Dokumente aus
- Verantwortlich für Planung und stellt die notwendigen Ressourcen (Personen, Budget, Zeit) zur Verfügung und das Review-Team zusammen
- Überwachung der Kosteneffizenz und das Treffen von steuernden Entscheidungen im Fall von unzureichenden Ergebnissen des Reviewprozesses

4.4.2 Reviewleiter

- Reviewleiter trägt die Gesamtverantwortung für den Review (Planung, Vorbereitung, Durchführung und Über-/Nachbearbeitung)
- Entscheidet, welche Personen am Review beteiligt sind und organisiert Zeitpunkt und Räumlichkeit für Sitzung

4.4.3 Reviewmoderator

- Leitet die Sitzung
- entscheidende Person für Erfolg
- Muss sehr guter Sitzungsleiter sein
- Muss bei gegensätzlichen Standpunkten vermitteln können und auch auf Untertöne achten
- Darf nicht voreingenommen sein und keine eigene Meinung zum Prüfobjekt äussern

4.4.4 Autor

- Ersteller des Dokuments bzw. des Arbeitsergebnisses, das einem Review unterzogen wird
- Bei mehreren Erstellern sollte ein Hauptverantwortlicher benannt werden, der die Rolle des Autors übernimmt
- Wichtig ist, dass der Autor die geäusserte Kritik nicht als Kritik an seiner Person auffasst, sondern dass es ausschl. um die Verbesserung der Qualität geht

4.4.5 Reviewer (Gutachter, Inspektor)

- Mehrere (max. 5) Fachexperten
- Verantwortung, gute Abschnitte entsprechend zu kennzeichnen
- Mangelhafte Teil zu dokumentieren und nachvollziehbar zu beschreiben
- Reviewer sollten gewählt werden, dass versch. Sichten im Reviewprozess vertreten sind

4.4.6 Protokollant

- Dokumentiert alle Ergebnisse, Probleme und offene Punkte während Sitzung
- Kurz und präzise formulieren
- Schreibt Diskussionsbeiträge verfälschungsfrei auf
- Protokollant und Reviewmoderator sollte nicht dieselbe Person sein
- Manchmal sinnvoll, wenn der Autor auch der Protokollant ist

4.5 Reviewarten

4.5.1 Informelles Review

- Abgeschwächte Form des Reviews
- Autor-Leser-Feedbackzyklus

4.5.2 Walkthrough

- Mittelpunkt eine Sitzung
- kein formaler Ablauf

4.5.3 technische Review (auch fachliches Review)

- Übereinstimmung des Prüflings mit einer Spezifikation, Eignung für den Einsatz, Einhaltung von Standards
- Hinzuziehen von Fachexperten

4.5.4 Inspektion

- Formalste Art
- definierter Ablauf mit genauen Prüfkriterien
- Aufdeckung von Unklarheiten, mögliche Fehler(zustände), Bestimmung der Qualität, Vertrauen in das Arbeitsergebnis schaffen

4.6 Erfolgsfaktoren, Vorteile und Grenzen

- Reviews sind effiziente Mittel zur Sicherung der Qualität
- Idealerweise direkt nach Fertigstellung der Arbeitsergebnisse durchzuführen
- Frühe Finden von Fehler(zustände)n durch Reviews meist kostengünstiger als das spätere Erkennen
- Wissensaustausch unter den beteiligten Personen
- Organisatorische Erfolgsfaktoren
  - Mgmt und PL unterstützen Reviews
  - angemessene Frist geplant
  - genügend Zeit für Vorbereitung
  - Checklisten sind aktuell
  - Umfangreiche Dokumente werden nicht als ganzes geprüft
- Personenbezogene Erfolgsfaktoren
  - passende Personen auswählen
  - dienen zur Qualitätssicherung
  - ausreichend Zeitnehmen und Aufmerksamkeit für Details
  - Kultur des 'ständigen Lernens'

4.7 Unterschied statische vs. dynamische Tests

- Statische und dynamische Tests können das gleiche Ziel verfolgen → Fehler finden
- statische Tests entdecken Fehler direkt in Dokumente
- dynamische Tests weisen Fehlerwirkungen nach - vor allem im Programmcode und nicht in anderen Dokumenten

4.8 Statische (werkzeugbasierte) Analyse

Oftmals Oberbegriffe für alle Prüfverfahren, bei denen keine Ausführung des Testobjekts erfolgt

- Überprüfung, Vermessung und Darstellung eines Dokuments bzw. eines Programms oder einer Kopmonente
- müssen formale Struktur haben
- typischerweise wird Quelltext analyisert
- Prüfung auf Einhaltung von Konventionen und Standards
- Datenflussanalyse
- Kontrollflussanalyse
- Erhebung von Metriken

Als Beispiel ist der Compiler zu nennen

4.8.1 Prüfung der Einhaltung von Konventionen und Standards

- Einhaltung der Programmierkonventionen durch Analysatoren
- Möglichst automatisch überprüfen lassen

4.8.2 Datenflussanalyse

- ur-Anomalie: Ein undefinierter Wert (u) einer Variablen wird auf einem Programmpfad gelesen (r)
- du-Anomalie: Die Variable erhält einen Wert (d), der allerdings ungültig (u) wird, ohne dass er zwischenzeitlich verwendet wurde
- dd-Anomalie: Die Variable erhält auf einem Programmpfad ein zweites Mal einen Wert (d), ohne dass der erste Wert (d) verwendet wurde

Bspw. mit SonarLint oder SpotBugs

4.8.3 Kontrollflussanalyse

- Programmstück kann als Kontrollflussgraph dargestellt werden
- Anweisung + Sequenz von Anweisung ist ein Knoten
- Änderungen werden durch Verzweigungen und Schleifen erreicht
- Durch die Visualisierung der Kontrollflussgraphen lassen sich Anomalien leicht erfassen

  Sprünge aus Schleifen

  Mehrere Ausgänge

4.8.4 Ermittlung von Metriken

- Werkzeuge der statischen Analyse liefern auch Messwerte (Siehe ISO DIN 25010)
- Werkzeuge: SonarQube, Eclipse Metrics

4.8.5 Zyklomatische Zahl

- die Zyklomatische Zahl oder Komplexität misst die sturkturelle Komplexität des Quellcodes
- Grundlage für die Berechnung ist der Kontrollflussgraph
- Die zyklomatische Zahl kann genutzt werden, um Abschätzungen in Bezug auf die Testbarkeit und die Wartbarkeit des Programmteils vorzunehmen
- Zahl höher als 10 ist nicht tolerabel → 3-6 ist akzeptabel
- Zahl gibt Auskunft über den Testaufwands
- Zyklomatische Zahl = Anzahl linear unabhängigen PFade im Programmstück → obere Grenze für die benötigten Testfälle zur Erreichung dieses Kriteriums

Kapitel 5

Dynamische Test

- Programme sind statische Beschreibungen von dynamischen Prozessen (Berechnungen)
- Statistische Tests = Prüfen des Testobjekts
- Dynamische Tests = prüfen durch Interpretation einer Beschreibung die resultierenden Prozesse
- Das Testobjekt wird im dynamischen Test als auf einem Prozessor ausgeführt

5.1 Der Testrahmen

- Testobjekt ruft meist weitere Programmteile über definierte Schnittstellen auf
- Diese Programmteile werden durch Platzhalter (Mock, Stubs) realisiert
- Platzerhalter simulieren das Ein- und Ausgabeverhalten

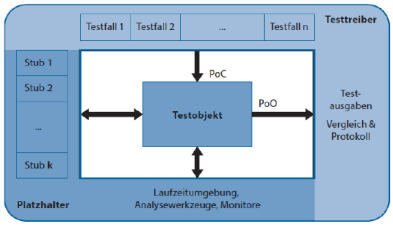

Abbildung 5.1: Abbildung des Zusammenspiels zwischen Testobjekte

5.2 Die Verfahren

Grundsätzlich wird zwischen Blackbox und Whitebox Verfahren unterschieden.

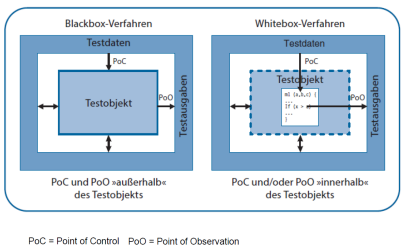

Abbildung 5.2: Abbildung der beiden Testverfahren

5.2.1 Das wichtigste zu Blackbox-Verfahren

- Undurchsichtig
- Keine Informationen über den Programmtext und inneren Aufbau
- Beobachtung von aussen
- Steuerung des Ablaufs nur durch Wahl der Eingabedaten
- Sowohl funktionale, als auch nicht-funktionale Tests
- Spezifikationsorientierte Verfahren (funktionale Testverfahren)

  Modelle bzw. Anforderungen an das Testobjekt, werden zur Spezifikation des zu lösenden Problems herangezogen

  von diesem Modell können systematisch Testfälle abgeleitet werden

5.2.2 Das wichtigste zu Whitebox-Verfahren

- durchsichtig
- Testfälle können aufgrund der Programmstruktur des Testobjektes gewonnen werden
- Informationen zum Aufbau werden für die Ableitung der Testfälle verwendet (bspw. Code und Algorithmus)
- Überdeckungsgrad des Codes kann gemessen werden
- Weitere Testfällen können zur Erhöhung des Überdeckungsgrad systematisch abgeleitet werden
- der innere Ablauf wird analysiert
- Eingriff in den Ablauf im Testobjekt möglich (bspw. für Negativtests)
- für untere Teststufen und Komponenten- und Integrationstest

5.2.3 Erfahrungsbasierte Testfallermittlung

- Nutzen Wissen und die Erfahrung von Menschen zur Ableitung der Testfälle
- Wissen über die Software, ihre Verwendung und ihre Umgebung
- Wissen über wahrs. Fehler und ihre Verteilung

5.3 Blackbox-Verfahren

Dies kann mittels versch. Verfahren geprüft werden

- Äquivalenzklassenbildung
- Grenzwertanalyse
- Zustandsbezogener Test
- Anwendungsfallbasierter Test
- Entscheidungstabellentest

5.3.1 Äquivalenzklassenbildung

- möglichst wenig Testfälle für möglichst wirkungsvolle Tests
- Spezifikation wird nach Eingabegrössen und deren Gültigkeitsbereich durchsucht

  Je Gültigkeitsbereich wird eine eine Klasse definiert
- ist vollständig, wenn alle Repräsentanten getestet sind

Beispiel: Ein Sensor meldet an ein Programm die Motoröltemperatur. Ist die Öltemperatur (T) unter 50 C soll eine blaue Kontrolllampe leuchten, bei T über 150 C soll eine rote Kontrolllampe leuchten. Sonst soll keine der Lampen leuchten.
⇒ Es ergibt folgende Äquivalenzklassen:

1. T $\leq$ 50 C $\rightarrow$ Repräsentat: 17 C
2. 50 C $\leq$ T $\leq$ 150 C $\rightarrow$ Repräsentat: 97 C
3. T $\geq$ 150 C $\rightarrow$ Repräsentat: 159 C

Testfälle mit mehreren Parametern

- Eindeutige Kennzeichnung jeder Äquivalenzklasse
- gÄKn = gültige Äquivalenzklasse n
- uÄKn = ungültige Äquivalenzklasse n
- Pro Parameter mind zwei Äquivalenzklasse

  eine mit gültigen Werten

  eine mit ungültigen Werten
- Bei n Paramter mit m_i Äquivalenzklassen (i = 1 .. n) gibt es $\prod_{i=1..n} m_i$ unterschd. Kombinationen (testfälle)

Testfälle minimieren

- Testfälle aus allen Repräsentanten kombinieren und anschliessend nach Häufigkeit sortieren
- Sicherstellen, dass jeder Repräsentant einer Äquivalenzklasse mit jedem Repräsentant jeder anderen Äquivalenzklasse in einem testfall zur Ausführung kommt (d.h. paarweise Kombination statt vollständige Kombination)
- Minimalkriterium: Mindestens ein Repräsentant jeder Äquivalenzklasse in mind. einem Testfall
- Repräsentanten ungültiger Äquivalenzklasse nicht mit Repräsentanten anderer ungültiger Äquivalenzklassen kombinieren.

Testendkriterium

Äquivalenzklassen-Überdeckung = (Anzahl getestete ÄK / Gesamtzahl ÄK) * 100%
Beispiel:
19 AK aus Anforderungen, 15 wurden abgedeckt mit Testfällen, so gibt es eine AK-Überdeckung von 83.33% *Rightarrow* (15/18)*100=83.33

Pro/Cons

Vorteile

- Anzahl der testfälle kleiner als bei unsystematischer Fehlersuche
- Geeignet für Programme mit vielen versch. Ein- und Ausgabebedingungen

Nachteile:

- Betrachtet Bedingungen für einzelne Ein- / Ausgabeparameter
- Beachtung von Wechselwirkungen und Abhängigkeiten von Bedingungen sehr aufwändig

Empfehlung

- Kombination der ÄK-Bildung mit fehlerorientierter Verfahren bspw. Grenzwertanalyse

5.3.2 Grenzwertanalyse

- Idee: Verzweigungen- und Schleifenbedingungen haben oft Grenzbereiche, für welche die Bedingungen gerade noch zutreffen. Solche Fallunterscheidungne snd fehlerträchtig
- Beste Erfolge bei Kombination mit anderen Verfahren
- Bei Kombination mit AK → Grenzen der ÄK (grösster und kleinster Wert) testen und jeder Rand einer ÄK muss in einer Testdatenkombination vorkommen

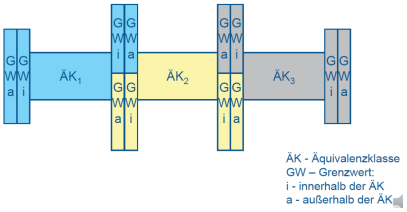

Abbildung 5.3: Abbildung der Grenzwertanalyse

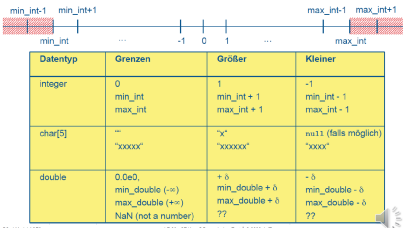

Abbildung 5.4: Beispiel einer Grenzwertanalyse

Testendkriterium

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

Pro/Cons

Vorteil:

- An den Grenzen von AK sind häufiger Fehler zu finden, als innerhalb dieser Klasse
- Die Grenzwertanalyse ist bei richtiger Anwendung eine der nützlichsten Methoden für den Testfallentwurf».
- Effiziente Kombination mit anderen Verfahren, die Freiheitsgrade in der Wahl der Testdaten lassen

Nachteil:

- Rezepte für die Ausawhl von testdaten schwierig anzugeben
- Bestimmung aller relevanten Grenzwerte schwierig
- Kreativität zur Findung von erfolgreicher Testdaten gefordert
- Oft nicht effizient genug angewendet, da sie zu einfach erscheint

5.3.3 Zustandbasierter Test

- Ausgangsbasis bildet die Spezifikatoin des Programms als Zustandsgraph (endlicher Automat)
- Zustände und Zustandsübergänge sind so beschrieben
- Testfall besteht aus folgenden Teilen

  Ausgangszustand

  Ereignis (Eingabedaten)

  Soll-Folge-Zustand (Soll-Zustand)
- Testumfang für einen betrachteten Zustand

  Alle Ereignisse, die zu einem Zustandswechsel führen

  Alle Ereignisse, die auftreten können, aber ignoriert werden

  Alle Ereignisse, die auftreten können und eine Fehlerbehandlung erfordern

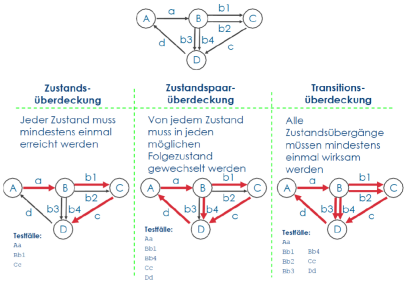

Abbildung 5.5: Abbildung eines Zustandsgraphen

Ein vollständiger zustandsbezogener Testfall umfasst folgende Informationen:

- Anfangszustand des Testobjektes (Komponente oder System)
- Eingaben für das Testobjekt
- Erwartete Ausgaben bzw. das erwartete Verhalten
- Erwarteter Endzustand

Ferner sind für jeden im Testfall erwarteten Zustandsübergang folgende Aspekte festzulegen:

- Zustand vor dem Übergang
- Auslösendes Ereignis, das den Übergang bewirkt
- Erwartete Reaktion, ausgelöst durchden Übergang
- Nächster erwartet Zustand

Roundtrip-Folgen

- Beginnend im Startzustand
- Endet in einem Endzustand oder in einem Zustand, der bereits in dieser oder einer anderne Roundtrip-Folge enthalten war
- (Zustands-) Übergangsbaum

Beispiel zustandsbasierter Test

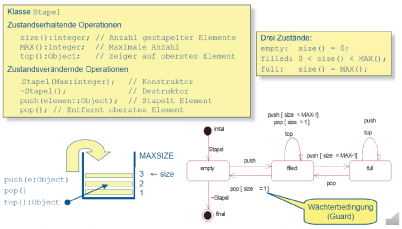

Abbildung 5.6: Beispiel eines zustandbasierten Tests

Arbeitsschritte

1. Analyse des Zustandsdiagramm
2. Prüfung auf Vollständigkeit → Zustandsdiagramm hinsichtlich der Vollständigkeit untersuchen
3. Ableiten des Übergangsbaumes für den Zustands-Konformanztest

   Anfangzustand ist die Wurzel

   Für jeden möglichen Übergang vom Anfangzustand zu einem Folgezustand im Zustandsdiagramm erhält der Übergangsbaum von der Wurzel aus einen Zweig zu einem Knoten, der den Nachfolgezustand repräsentiert. Am Zweig wird das Ereignis (Operation) und ggf. die Wächterbedignung notiert

   Letze Schritt wird für jedes Blatt des Übergangsbaums so lange wiederholt, bis eine der beiden Endbedingungen eintritt:

   [A] Der dem Blatt entsprechende Zustand ist auf einer höheren EBene bereits einmal im Baum enthalten;

   [B] Der dem Blatt entsprechende Zustand ist ein Endzustand und hat somit keine weiteren Übergänge, die zu berücksichtigen wären
4. Erweitern des Übergangsbaumes für den Zustands-Robustheitstest

   Robustheit unter spezifikationsverletzenden Benutzungen prüfen

   Für Botschaften, für die aus dem betrachteten Knoten kein Übergang spezifziert ist, den Übergangsbaum um einen neuen 'Fehler-Zustand' erweitern
5. Generieren der Ereignisssequenzen und Ergänzender Parameter bei Aktoinen bzw. Methodenaufrufen

   Pfade von Wurzel zu Blätter im erweiterten Übergangsbaum als Funktions-Sequenz auffassen

   Stimulierung des Testobjekts mit den entsprechenden Funktionsaufrufen deckt alle Zustände und Zustandsübergänge im Zustandsdiagramm ab

   Ergänzen der Funktions-Parameter
6. Ausführen der Tests und Überdeckungsmessung

Ausführen der Tests

- Testfälle bzw. Ereigbnisfolgen in ein Testskript verkapseln
- Unter Benutzung eines Testtreibers ausführen
- Zustände über zustandserhaltende Operationen ermitteln und protokollieren

Testendekriterien

Minimalkriterium: Jeder Zustand wurde mind. einmal eingenommen
Z-Überdeckung = (Anzahl getestete Z / Gesamtzahl Z) * 100 %

Weitere Kriterien: Jeder Zustandsübergang wurde mind. einmal ausgeführt
ZÜ-Überdeckung = (Anzahl getestete ZÜ / Gesamtzahl ZÜ) *100%

- Alle spezifikationsverletzenden Zustandsübergänge wurden angeregt
- Jede Aktion (Funktion) wurde mind. einmal ausgeführt

Bei hoch kritischen Anwendungen:

- Alle Zustandsübergänge und alle Zyklen im Zustandsdiagramm
- Alle Zustandsübergange in jeder bel. Reihenfolge mit allen möglichen Zuständen, auch mehrfach hintereinander

5.3.4 Entscheidungstabellentest

- Beschreibt sehr anschaulich die (Geschäfts-) Regeln der Art "wenn...dann...sonst"
- typische Anwendungssituation: Abhängig von mehreren logischen Eingabewerten
- untestützen durch ihren Aufbau die Vollständigkeit des Tests
- Jede Tabellenspalte (Regel) entspricht einem Testfall
- Regelüberdeckung: Je Regel wird mind. ein Testfall gewählt
- Das Soll-Resultat ergibt sich durch die entsprechend ausgeführten Aktionen

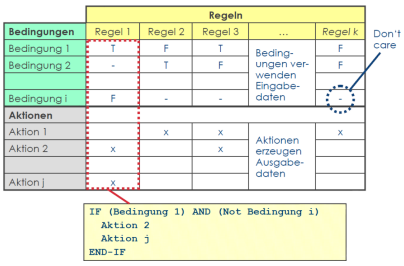

Abbildung 5.7: Abbildung eines Entscheidungstabellentest

Testfälle

- Jede Spalte (Regel) entspricht einem Testfall
  - Voraussetzung pro Tabelle gleich
  - Bedingungen beziehen sich auf Eingaben
  - Aktionen spiegeln erwaretetes Ergebnis wider
- Überdeckungskriterien
  - Alle Bedingungen mind. einmal N und J
  - Alle Aktionen mind. einmal x
  - Alle Spalten (alle Bedingungskombinationen)
- Konkrete Testdaten aus Wertebereichen abeleiten
  - Äquivalenzklassenbildung
  - Grenzwartanalyse

5.4 Whitebox-Testverfahren

- Nutzung von Informationen über das Interna eines Testobjekts
- Gesucht werden 'fehleraufdeckende' Stichproben der möglichen Programmabläuge und Datenverwendung
- Alle Testverfahren, die zur Herleitung oder Auswahl der Testfälle sowie zur Bestimmung der Vollständigkeit der Prüfung (Überdeckungsgrad) Information über die inere Struktur des Testobjekts hernaziehen
- Strukturelle oder strukturbasierte Testverfahren genannt
- codebasierte Testverfahren
- Kann in allen Teststufen genutzt werden, folgende vor allem in der Komponententeststufe eingesetzt
- Gängige Verfahren
  - Anweisungstest
  - Entscheidungstest (Zweigtest)
  - Test der Bedingungen
  - Pfadtest

5.4.1 Anweisungstest und Anweisungsüberdeckung

- einzelne Anweisungen (statements) des Testobjekts stehen im Mittelpunkt
- Testfälle zu identifizeiren, die ein zuvor festgelegte Mindestquote oder auch alle Anweisungen des Testobjekts zur Ausführung bringen
- Verfahren kann sehr gut an einem Kontrollflussgraphen veranschaulicht und erklärt werden
- Kriterium zur Beendigung der Tests
  - Anteil der ausgeführten Anweisungen (Statement coverage) → auch C0-Mass bezeichnet
- dynamisches, kontrollflussbasiertes Testverfahren, das die mindestens einmalige Ausführung aller Anweisungen des Testobjekts fordert
  - Anweisungsüberdeckung = (Anzahl durchlaufener Anweisungen / Gesamtzahl Anweisungen) * 100 %

- Jeder Testfalll wird anhand eines Pfads durch den Kontrollflussgraphen bestimmt

  Bei dem Testfall müssen die auf dem Pfad liegenden Kanten des Graphen durchlaufen werden, d.h. die Anweisungen (Knoten) in der entsprechenden Reihenfolge zur Ausführung kommen

  Häufigkeit der Ausführung spielt keine Rolle, zählt nur ob eine Anweisung ausgeführt wurde

- Wenn der zuvor festgelegte Deckungsgrad erreicht ist, wird der Test als ausreichend angesehen und beendet

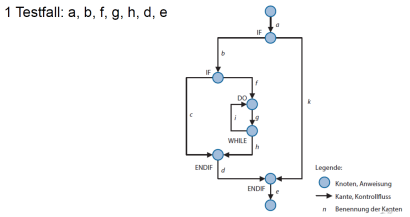

Abbildung 5.8: Beispiel Anweisungsüberdeckung

Diskussion

- Anweisungsüberdeckung ist in der Aussage ein schwaches Kriterium
- 100%ige Überdeckung ist in der Praxis nicht immer erreichbar

  bspw. wenn Ausnahmebedingungen im Programm vorkommen, die während der Testphase nur mit erheblichem Aufwand oder gar nicht herzustellen sind

  Kann auch auf Dead-Code hinweisen, wobei hier eine statische Analyse gemacht werden müsste

5.4.2 Entscheidungstest und Entscheidungsüberdeckung

- weiteres Kriterium ist die Überdeckung der Entscheidungen
- Entscheidungen im Programmtext stehen im Mittelpunkt der Untersuchung
- Nicht die Ausführung wird betrachtet, sondern die Auswertung einer Entscheidung
- Aufgrund des Ergebnisses wird entschieden, welche Anweisung als nächste ausgeführt wird.
- Kriterium zur Beendigung:

  Anteil der ausgeführten Programmzweige (Branch coverage)

  Grundlage bildet in der Regel der Kontrollflussgraph (Kantenüberdeckung) → auch C1-Mass genannt

- kontrollflussbasiertes Testverfahren, fordert die Überdeckung aller Entscheidungen bzw. Zweige eines testobjekts zu allen Fällen
- Zweigüberdeckung zielt auf alle Zweige im Kontrollflussgraphen abzudecken

  Zweigüberdeckung = (Anzahl durchlaufener Zweig / Gesamtzahl Zweige) * 100%

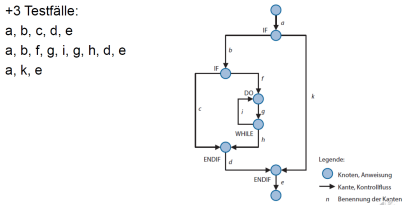

Abbildung 5.9: Beispiel Entscheidungsüberdeckung

Diskussion Entscheidungsüberdeckung

- ist das umfassendere Kriterium
- Entscheidungs- oder Zweigtest braucht mehr Testfälle als beim Anweisungstest

  Anzahl zusätzlicher hängt von der Struktur des testobjekts ab
- Fehlende Anweisungen in leeren IF-Statements können erkannt werden (bei Anweisungstests nicht)
- 100% Entscheidungsüberdeckung garantiert 100% Anweisungsüberdeckung, aber nicht umgekehrt
- Entscheidungen werden unabhängig voneinander betrachet und es werden keine bestimmten Kombinationen der einzelnen Programmteile gefordert

5.4.3 Test der Bedienung

- Wahrheitswerte
- Atomare Teilbedingungen
- Zusammengesetzt Bedingnugen
- Entscheidungen sind zusammengesetzte Bedingungen, die den Programmablauf steuern
- Bei der Zweig-/Entscheidungsüberdeckung wird ausschliesslich der ermittelte Ergebnis-Wahrheitswert einer Bedingung berücksichtigt
- Als Überdeckungskriterien werden Verhältnisse zwischen den bereits erreichten und allen geforderten Wahrheitswerten der (Teil-) Bedingungen gebildet
- Bei Verfahren, welche die Komplexität der Bedingungen im Programmtext des Testobjekts in den Mittelpunkt der Prüfung stellen, ist es sinnvoll, eine vollständige Prüfung (100 Prozentige Überdeckung) anzustreben
- Folgendes wird betrachtet

  Einfache Bedingungsüberdeckung (Condition Testing)

  Mehrfachbedingungsüberdeckung (Multiple Condition Testing)

  Modifizierter bzw. minimaler Bedingungs-/Entscheidungstest (Modified Condition Decision Coverage Testing, MCDC)

Einfache Bedingungsüberdeckung

- kontrollflussbasiertes, dynamisches Testverfahren, das die Überdeckung der atomaren Teilbedingungen einer Entscheidung mit 'wahr' oder 'falsch' fordert
- Bei n atomaren Ausdrücken mindestens 2, höchstens 2^n Testfälle
- Bedingungsüberdeckung = (Anzahl zu wahr und falsch gesteteten atomaren Ausdrücken / Gesamtzahl atomarer Ausdrücke) * 100 %

SLIDE 18