Software Engineering & PM Vorlesung
Qualitätssicherung

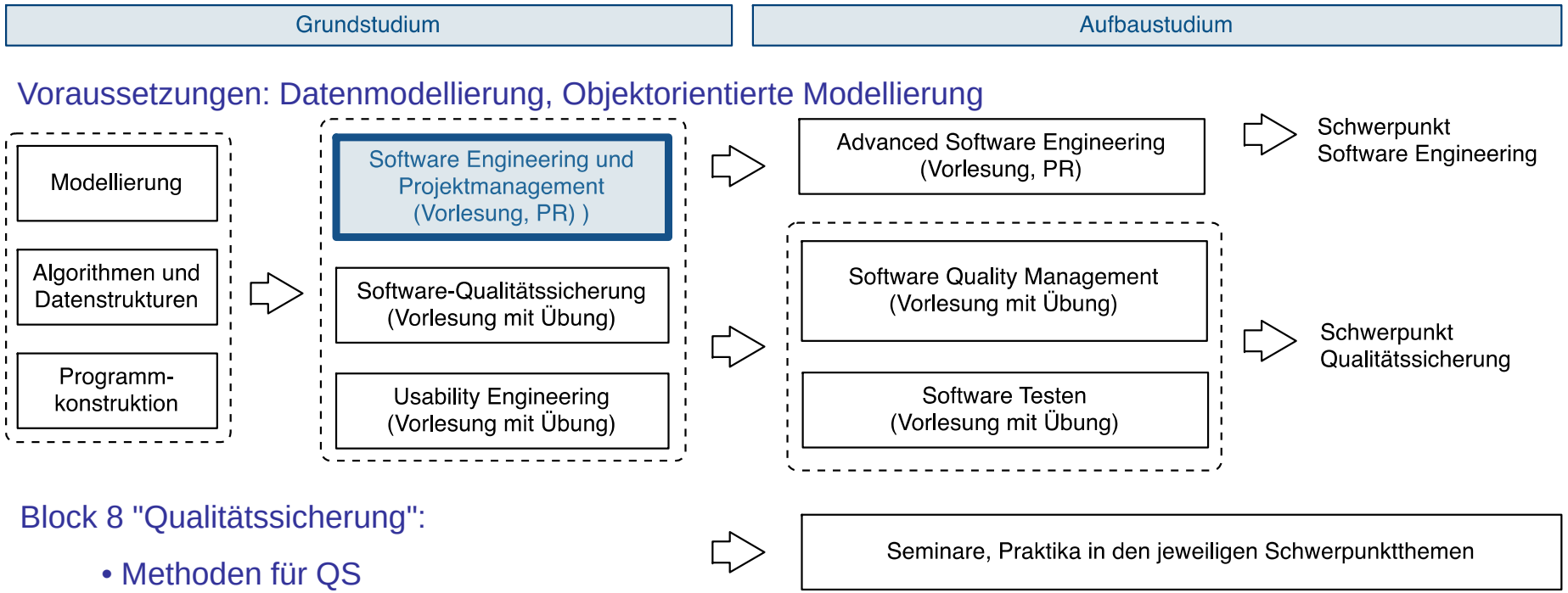
Institut für Information Systems Engineering

Stefan Biffl

Inhalt:

- Qualitätssicherung in SEPM
  - Grundlagen der QS
- SEPM PR: Kernmethoden und deren Anwendung
  - Review: Anforderungen – Modelle (ER, IDEF0, UML)
  - Testen: Erstellen, Beurteilen, Anwenden von Testfällen
  - Test-Driven Development
- SEPM VO: Prüfungsbeispiel "Restaurant„
- BPSE Buch, Kapitel 5

Software Engineering im Studium



Block 8 "Qualitätssicherung":

- Methoden für QS
  - Reviews
  - Test-Driven Development

Überblick Block 8: Qualitätssicherung

1. Qualitätssicherung in SEPM-Projekten

- Grundlagen, Begriffe
- Typische SE-Modelle und deren Verwendung für QS
- Testfälle, Test-Driven Development

2. Statische Methoden der QS in SEPM

- Definition und Arten von Reviews
- Anwendungsbeispiel “Restaurant”
  - Überprüfung der Anforderungen
  - Aus den Anforderungen abgeleitete Modelle
- Reviews von Anforderungen mit ER/UML/IDEF0 Diagrammen

3. Testansätze, Test-Driven Development

- Testen in der Qualitätssicherung und SE Prozessen
- Teststufen und Testarten
- JUnit, Test-Driven Development
- Beurteilung der Güte von Testfällen

Qualitätssicherung in SEPM

Qualitätssicherung in SEPM-Projekten

- Grundlagen, Begriffe
  - Qualität, Qualitätssicherung
  - Verifikation und Validierung
- Typische SE-Modelle und deren Verwendung für QS
  - Anwendungsszenarien, Datenmodelle, Zustandsdiagramme (UML)
  - Daten-/Kontrollflussmodelle (IDEF0, UML)
- Testfälle, Test-Driven Development
  - Herstellen von Testfällen vor der Implementierung
  - Ablauffähige Testfälle für automatischen Re-Test neuer Code-Teile
  - Frühe Review missionskritischer Akzeptanztestfälle

Motivation - Ziel

- Software-Entwicklung erfordert die Herstellung konsistenter Sichten auf Anforderungen und Entwurf eines Systems.
- Modelle helfen den Überblick zu bekommen und zu behalten.
  - Grundlage für effektives und effizientes Arbeiten im Team.
  - Gemeinsame Notation mit konsistenter Bedeutung.

- Reviews überprüfen die Korrektheit und Konsistenz von Artefakten an wohldefinierten Punkten des Entwicklungsprozesses.
  - Review der Anforderungen nach Änderungen
  - Review später hergestellter Dokumente gegen die aktuell gültigen Anforderungen

- Herausforderung: Konsistente Verwendung unterschiedlicher Modelle.
  - Systemstruktur: Subsysteme, Komponenten, Schnittstellen.
  - Verhalten von Komponenten; Interaktion zwischen Komponenten.
- Anforderungen ➔ Modelle ➔ Daten/Komponenten/Testspezifikationen.

- Tests überprüfen das Laufzeitverhalten von Systemteilen
  - Funktionen, Modelle und Qualitätsmerkmale
  - Ausgangsbasis: Anforderungen, Ein-/Ausgangsparameter des Systems, innere Systemstruktur

Qualität und Qualitätssicherung

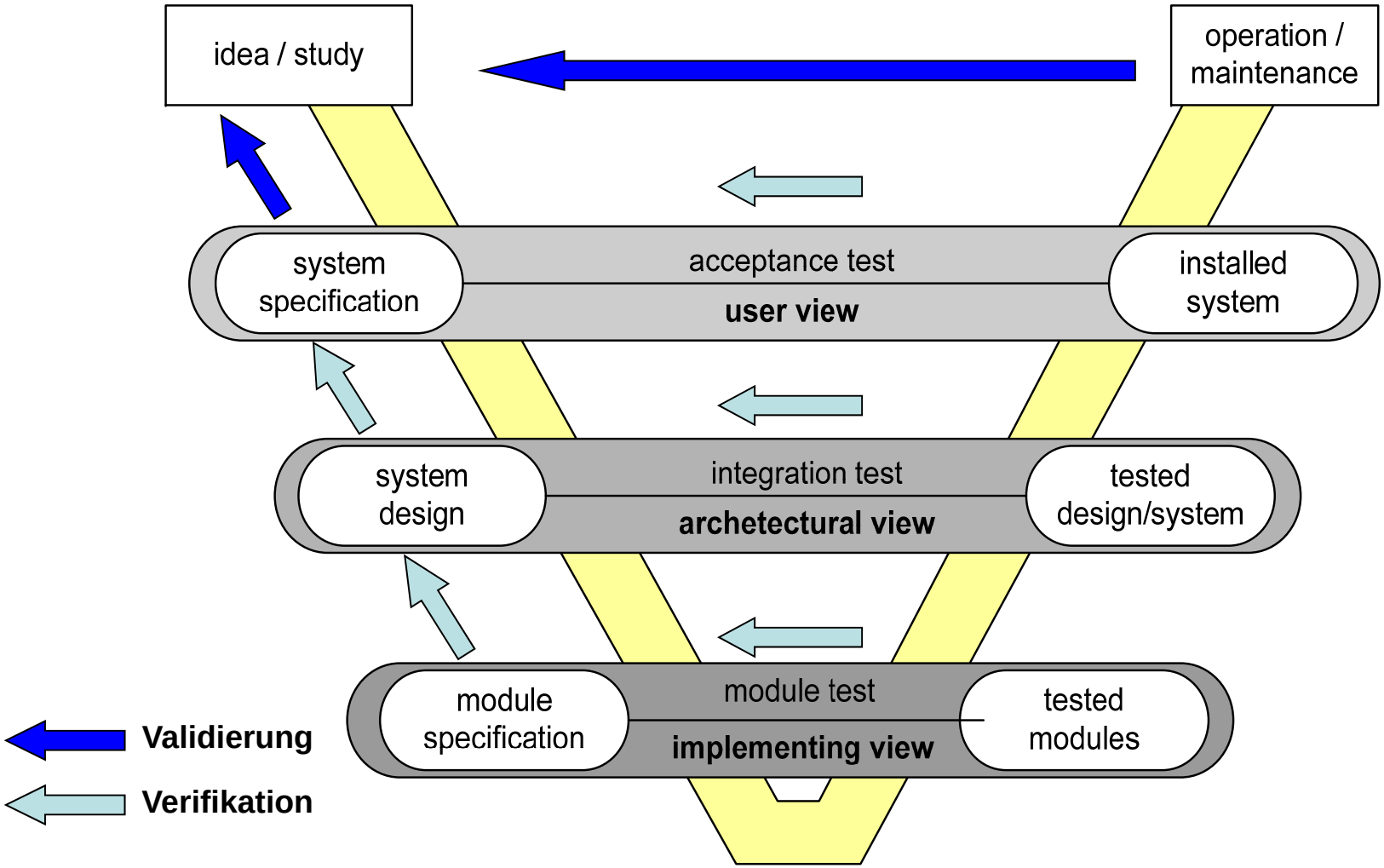
- Qualität ist die Eignung zur Erfüllung vordefinierter Anforderungen.
  - Beispiele für Qualitätsfaktoren (IEEE):
    - Korrektheit, Effizienz, Verwendbarkeit, Testbarkeit, Wartbarkeit
  - Qualität ist keine Eigenschaft, die später hinzugefügt werden kann.
  - Qualität muss während der Entwicklung gesichert werden.

- Qualitative hochwertige Software-Produkte sind
  - termingerecht und im Rahmen des Budgets erstellt.
  - für den Anwender verwendbar.
  - für den Professionisten verständlich und änderbar.
  - für den Betreiber effizient und administrierbar.

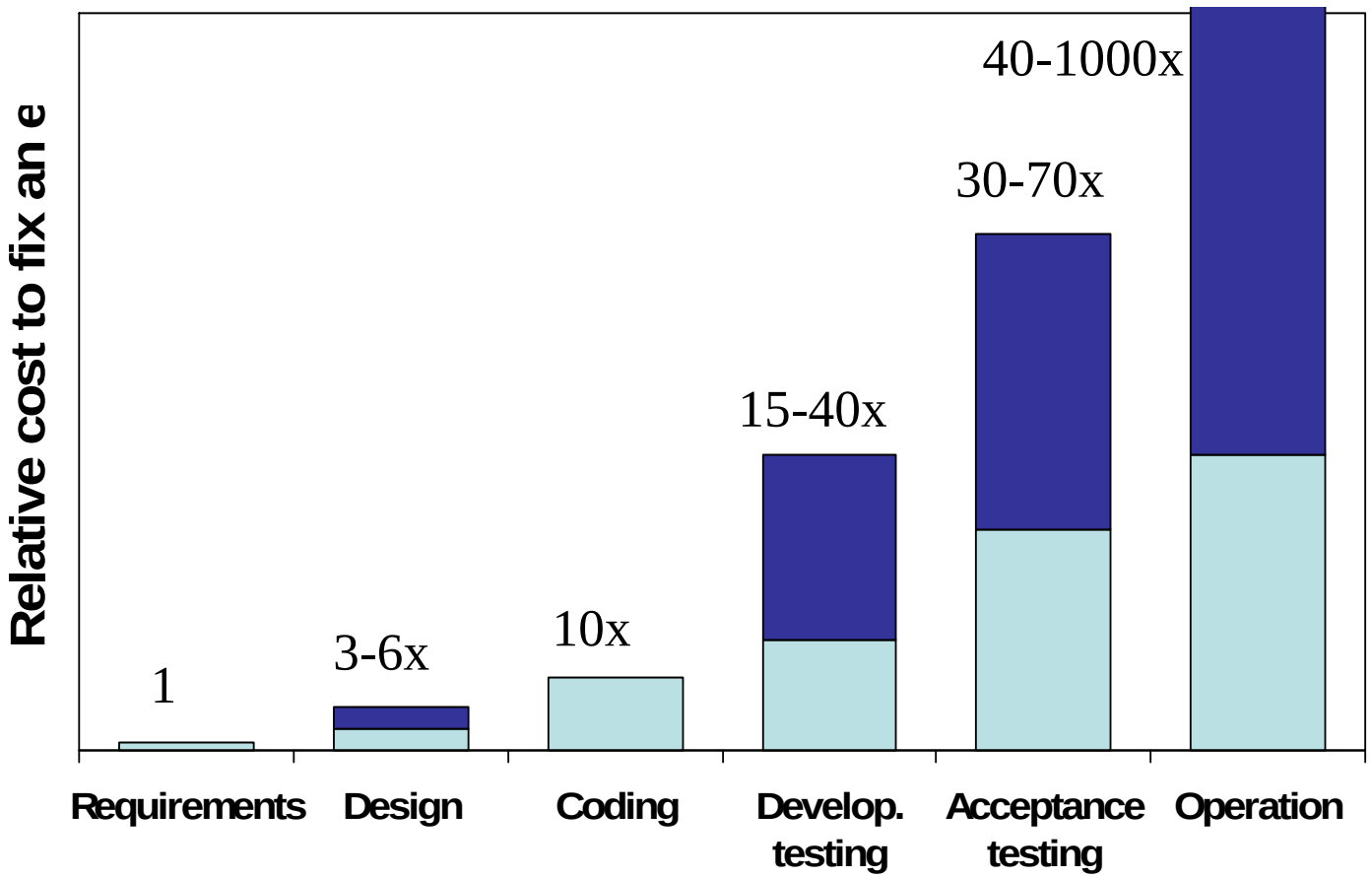
- Qualitätssicherung (QS) besteht in der Durchführung von Verifikation und Validierung in jeder Phase der Software-Herstellung

- Für die Umsetzung in SEPM sind daher notwendig:
  - Testbare Anforderungen
  - Verfolgbare Entwicklung der Anforderungen (mit Modellen) in testbare Produkte.

Testen im V-Modell



Software Development Rework Effort



The cost of fixing errors escalates as we move the project towards field use.
From an analysis of 63 projects cited in Boehm Barry, „Software Engineering Economics“, Prentice-Hall, 1981

Typische Fehler im Software Engineering SEPM PR Projekte

- Datenmodellierung, Reports
  - Schlüsselwerte nicht eindeutig, Fehlende Attribute
  - Fehler bei Beziehungen, z.B. 1:1 Relation im ER sehr selten
- Entwurf: Ergebniswerte von Methoden
  - Typkonversion falsch (z.B. Integer/Real)
- Initialisierung von Variablen
  - Fehlende Zuweisung: nicht eindeutiger Wert
  - Keyword static falsch verwendet: Klasse/Objekt (UML Rollen)
- Bedingungen
  - Logische Fehler; falsche Ausdrücke (und/oder)
- Schleifen
  - Anzahl Durchläufe nicht korrekt; z.B. Endlosschleifen
- Testfälle
  - Unzureichende Überdeckung (Auslassen von Anwendungsfällen, Programmteilen, GUI-Elementen, Datenzuständen)
  - Überflüssige Testfälle (gleiche Äquivalenzklasse)

Zusammenfassung: QS in SE-Projekten

- Für die Umsetzung von Qualität und QS in SEPM sind daher notwendig:
  - Testbare Anforderungen (Definition der Funktionen und Qualitäten)
  - Verfolgbare Entwicklung der Anforderungen (mit Modellen) in testbare Produkte.
- Typische SE-Modelle und deren Verwendung als Basis für die QS
  - Anwendungsszenarien (etwa Anwendungsfälle)
  - Datenmodelle (Relationenmodell(ER), Domänenmodell)
  - Datenflussmodelle (UML, IDEF0)
  - Kontrollflussmodelle (UML, Pseudocode)
  - Zustandsdiagramme, Sequenzdiagramme (UML)
  - Testfallspezifikationen
- Test-Driven Development
  - Herstellen von Testfällen vor der Implementierung
  - Ablauffähige Testfälle für automatischen Re-Test neuer Code-Teile
  - Frühe Review missionskritischer Akzeptanztestfälle

Statische Methoden der QS in SEPM

Statische Methoden der QS in SEPM

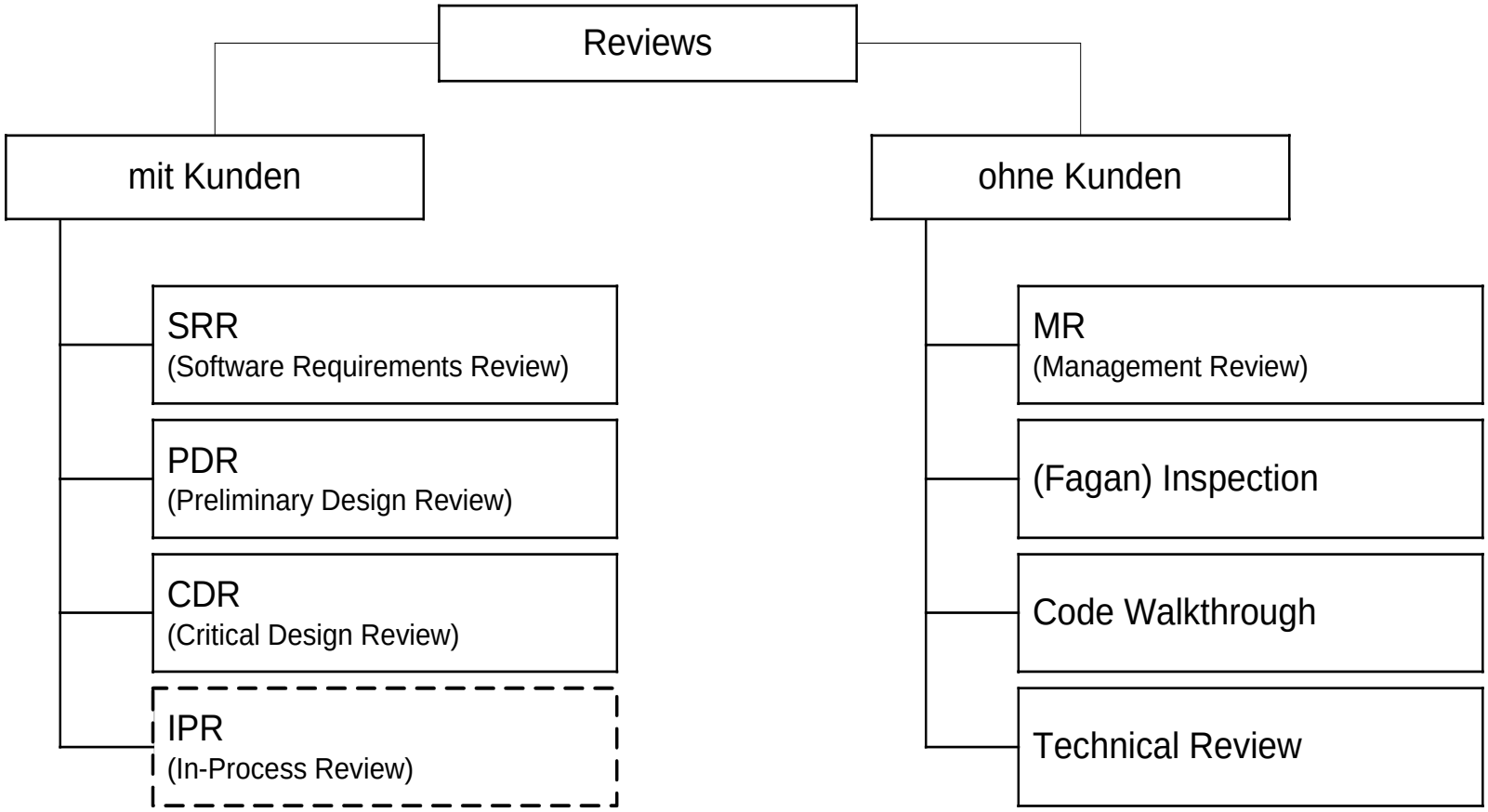
- Definition und Arten von Reviews
  - Rollen bei Reviews
  - Ablauf von Reviews
  - Richtlinien für technische Reviews
- Anwendungsbeispiel “Restaurant”
  - Überprüfung der Anforderungen
  - Aus den Anforderungen abgeleitete Modelle
    - Datenmodelle: Entity Relationship
    - Aktivitätsdiagramme: Datenfluss und Kontrollfluss
    - Zustandsdiagramm: State Machine Diagram
  - Reviews von Anforderungen mit ER/UML/IDEF0 Diagrammen

Reviews

Definition

- „Ein Review ist ein [..] formal geplanter und strukturierter Analyse- und Bewertungsprozess, in dem Projektergebnisse einem Team von Gutachtern präsentiert und von diesen kommentiert oder genehmigt werden.“
  [IEEE Std 610, Wallmüller 2001]
- Reviews dienen vor allem zur qualitativen Beurteilung von Produkten und Prozessen,
  die quantitativ nur schwer oder gar nicht beurteilt werden können (z.B. Modelle, Dokumente)
- Zentral ist die Beurteilung des Produktes und nicht des Autors!
- Unterschiedliche Ausprägungen von Reviews zielen auf unterschiedliche Ziele ab.
- In einen Reviewprozess sind definierte Rollen mit zugeordneten Aufgaben involviert.

Arten von Reviews [Thaller, 2000]



Reviews:
... und verwandte Tätigkeiten

- Inspektion
  - Ziel: Behebung von konkreten Mängeln
  - Bed.: Leser ist nicht Autor
  - Unb.: Alternativen, Stil
- Walkthrough
  - Ziel: Behebung von konkreten Mängeln
  - Bed.: Autor ist Leser & Moderator
  - Bed.: Alternativen, Stil

- Audit
  - Ziel: Überblickende Kontrolle
  - Bed.: Planung durch externe Personen
  - Bed.: Projektmitarbeiter nur als Information

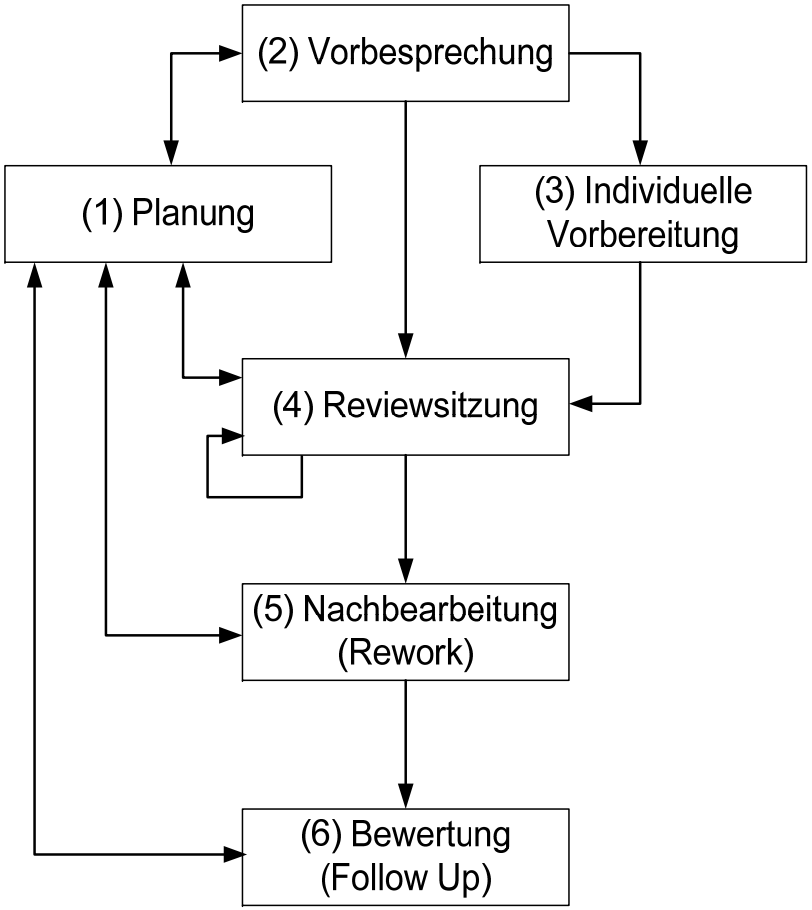
Rollen bei Reviews

- Die typische Größe eines Reviewteams liegt (abhängig von Projektart, -größe, usw.) im Bereich 3 bis 6 Personen, die sich auf folgende Rollen aufteilen:

  - Moderator („keeper of the process“)
    Leiter des Reviews
  - Leser („keeper of focus and pace“)
  - Gutachter („Reviewer“)
    Kommentierung des Reviewobjektes.
  - Schreiber („preserver of knowledge“)
    Protokollschreiber verfasst Protokoll
    (Notizen während der Reviewsitzung)

  - Autor („author“)
    Klärung von offenen Fragen.
    Keine Kommentierung und Rechtfertigung der Lösungen.

Ablauf einer Review



- Planung: Objekt, Prüfziele, Auslösekriterien (Einstiegskriterien), Teilnehmer, Ort, Zeit.
- Vorbesprechung: Vorstellung des Prüfobjekts bei komplexen und neuen Produkten.
- Intensive Einzeldurcharbeitung
- Durchführung: Gemeinsames Lesen, Aufzeichnung von Mängeln; während des Reviews sollen Mängel entdeckt, nicht korrigiert werden.
- In der Nachbearbeitung werden dokumentierte Mängel korrigiert und in der Bewertung überprüft.
- Berichterstattung.
- Wiederholungen von Reviews sind möglich.

- Checklisten unterstützen Reviews.
- Typische Dauer: 2h

Durchführung von Reviews

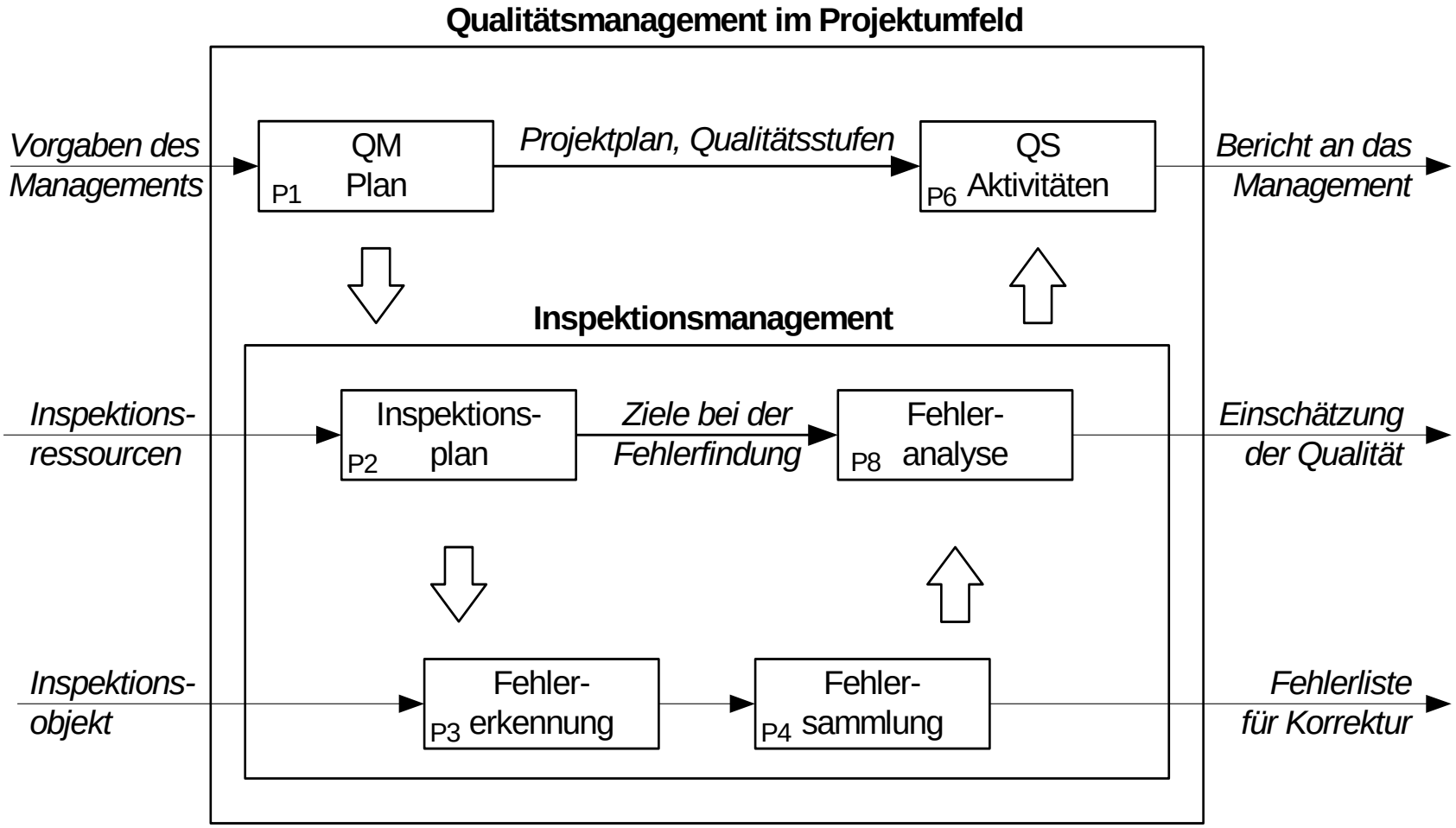
Um brauchbare Ergebnisse zu erzielen, ist der Ablauf einer Review klar definiert und umfasst folgende Phasen:

- Planung („planning phase“)
- Initialisierung („initialisation phase“)
- Vorbereitung („preparation phase“)
- Reviewsitzung („review“)
- Sammlung im Team („meeting“)
- Reviewbericht („reporting“)
- Nacharbeit („re-work“)

Richtlinien für technische Reviews

1. Das Produkt reviewen, nicht den Autor.
2. Verwenden Sie einen Arbeitsplan.
3. Diskussionen sachlich und kurz halten.
4. Problembereiche identifizieren, aber nicht jedes Problem gleich lösen.
5. Schriftliche Aufzeichnungen führen.
6. Anzahl der Teilnehmer begrenzen; gute Vorbereitung aller Teilnehmer.
7. Für jedes Produkt eine passende Checkliste verwenden.
8. Ausreichende Ressourcen und Zeitbudget zur Verfügung stellen.
9. Vorab Training für alle Reviewer.
10. Reviews im Nachhinein beurteilen für Verbesserung der Reviews.

Inspektionsplanung und Kontrolle



Qualitätsdefinition: Beispiel "Restaurant"

- Mögliche Formen der Anforderungen
  - Texte, Modelle, Referenzdokumente (z.B. Standards)
  - Liste mit Aussagen
  - Anwendungsszenarien
  - Text Beschreibung enthält Akteure, Entitäten und Aktivitäten/Operationen.
- Anforderungen beschreiben Funktionen und Qualitäten
  - Überprüfung der operativen Definition der Funktionen und Qualitäten.
- Auflisten von Kandidaten für Funktionen und Qualitäten
- Falls eine Funktion oder Qualität nicht vollständig testbar erscheint, Vorschlag für operativ testbare Variante machen.

Überprüfung der Anforderungen

- Kriterien für brauchbare Anforderungen
  - Notwendige Kriterien
    - Typ (z.B. FURPS)
    - Testbarkeit: Erfüllung einer Anforderung durch ein System muß eindeutig testbar sein (Messvorschrift: was ist wie zu messen? Z.B. Dauer eines Ablaufs).
    - Widerspruchsfrei unter einander
  - Erstrebenswerte Kriterien
    - Klare, knappe Beschreibung
    - Umsetzbarkeit
    - Konsistent mit bekannten Interessen der Stakeholder (richtig, „komplett“, relevant)
- Wichtig: Darstellung, die eine systematische Bearbeitung unterstützt
  - Vor der weiteren Verwendung (z.B. von Prosatext) kann eine Transformation die Verwendung der Anforderungen erleichtern; z.B. Herstellung einer Liste von Anforderungen.

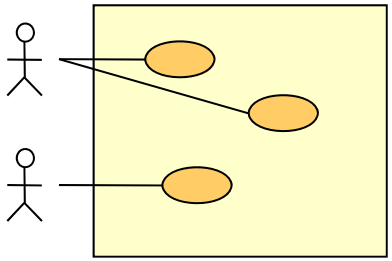
Von Anforderungen abgeleitete Modelle

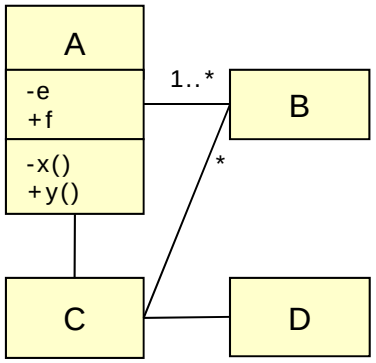
- Anforderungen ➔ Modelle ➔ Datenbank, Business Logic, GUI
  - Datenbanken: Datenmodellierung, Entity Relationship
  - Business Logic: Teilsysteme, Daten- und Kontrollfluss
  - Schnittstellen zwischen Teilsystemen und Benutzern

- Anforderungen ➔ Qualitätsdefinition; Modelle
  -> Qualitätssicherung: Review, Testen
  - Anforderungen beschreiben Funktionen und Qualitäten
  - Zur operativen Definition sind die Qualitäten testbar zu beschreiben.

- Fokus: Test-Driven Development (TDD) Testfälle
  - Herstellen von ablauffähigen Testfällen **vor** der Implementierung
  - TDD überprüft die Spezifikation vor dem detaillierten Entwurf

Überprüfung statischer Aspekte in UML





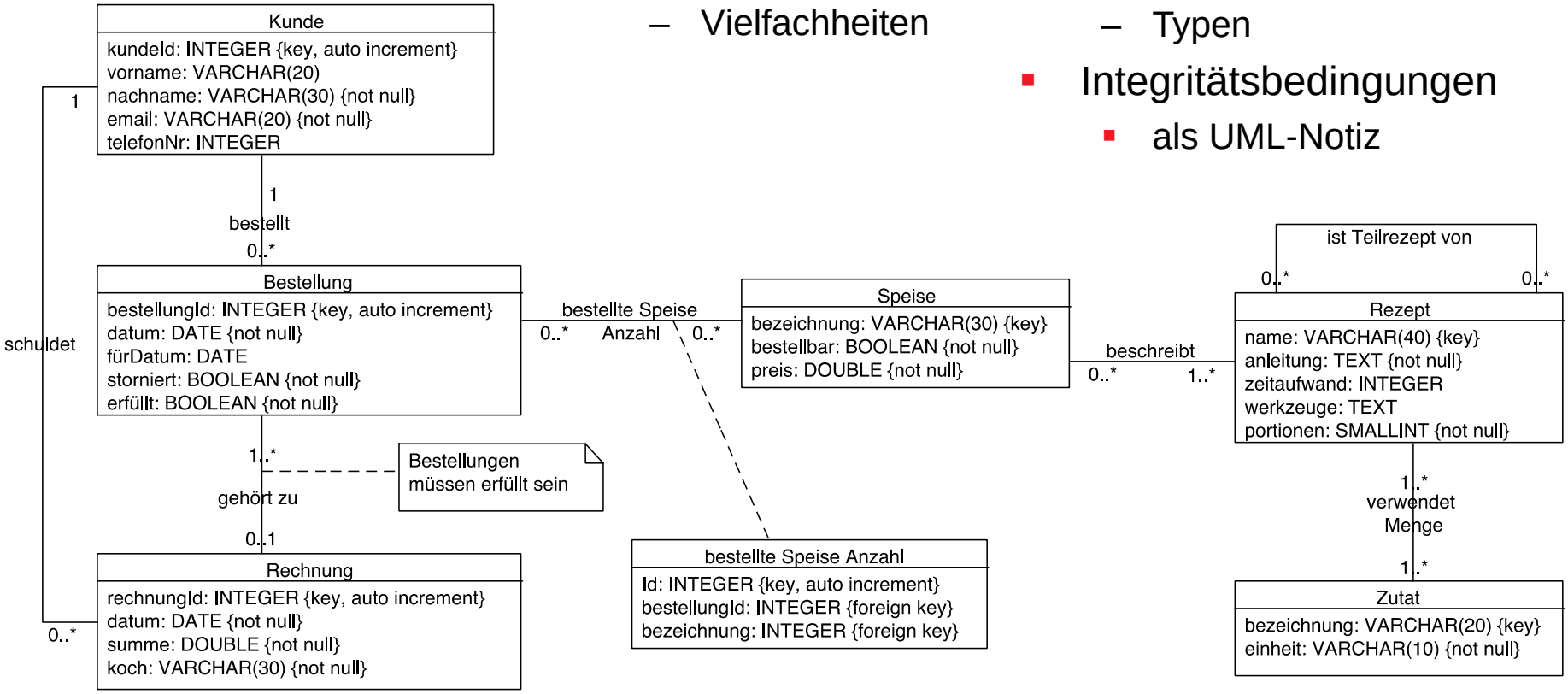
- **Anwendungsfälle**
  - die Benutzer und Nützlichkeit eines Systems
  - definiert durch Akteure und Hauptszenario
- **Klassen**
  - Menge gleichartiger Objekte
  - definiert durch Attribute, Operationen, und Beziehungen zu anderen Klassen
  - auch ein Datenbankschema kann in UML Klassendiagrammnotation modelliert werden.

Überprüfung statischer Aspekte

- Anwendungsfälle
  - Software Requirement Review
  - Jeder Anwendungsfall muss in den Anforderungen beschrieben werden
  - Akteure + Operationen in den Anforderungen führen zu den Anwendungsfällen
- Klassendiagramme
  - Preliminary Design Review
  - Die Daten welche in den Anforderungen zu finden sind werden von Model-Klassen gekapselt
  - Operationen + Entities in den Anforderungen führen zu Klassen
  - Aber auch: GUI Klassen, Support-Klassen für Internationalisierung, ...

Beispiel für UML Entity Relationship (Restaurant, unvollständig)

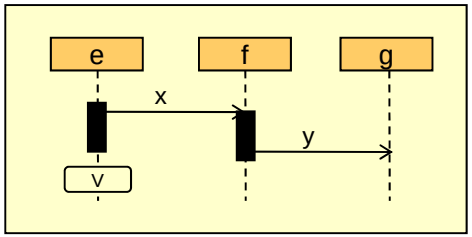
- Entitäten
  - Eine Tabelle pro Entität
- Beziehungen
  - Vielfachheiten
- Attribute
  - Schlüssel
    - Eigenschlüssel
    - Fremdschlüssel
  - Typen
- Integritätsbedingungen
  - als UML-Notiz

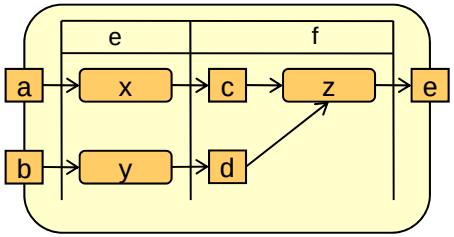


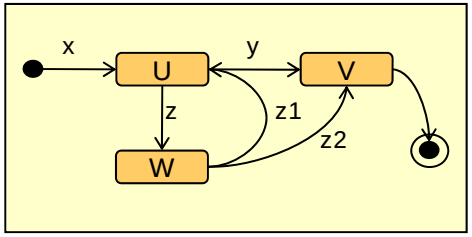
Überprüfung Datenbankrelationen
Review mit ER und Anforderungen

- Anforderungen strukturiert darstellen, z.B. als Liste.
- Anforderungen mit ER-Model reviewen und konsistent machen.
- ER-Modell systematisch durchgehen
  - Datenbanktabellen zu Entitäten und Beziehungen finden (markieren).
  - Schlüsselattribute auf Eindeutigkeit überprüfen
  - Attribute: Typen und Optionen überprüfen
    - Sind weitere Entitäten notwendig? (3. Normalform/Relational)
  - Integritätsbedingungen zu Attributen überprüfen bzw. ergänzen
    - Einzelne Attribute
    - Für Anforderungen relevante Bedingungen über mehrere Attribute
- Datenbanktabellen systematisch durchgehen.
  - Nicht markierte Elemente auf Sinnhaftigkeit überprüfen

Überprüfung dynamischer Aspekte in UML







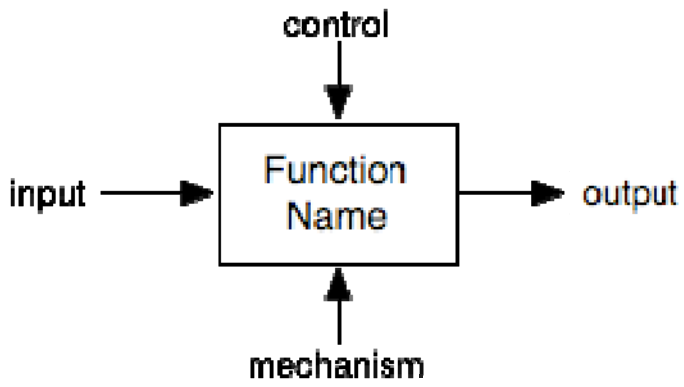
- **Sequenz**
  - ge- bzw. verbotene Interaktionsszenarien
  - definiert durch Folgen von Nachrichtenaustauschen zwischen Interaktionspartnern
- **Aktivität**
  - in sich geschlossenes, pro-aktives Verhalten
  - definiert durch Aktionen, Kontroll- und Datenfluss sowie evtl. Zuständigkeiten
- **Zustandsautomat**
  - reaktives Verhalten
  - definiert durch Zustände, Ereignisse und Transitionen

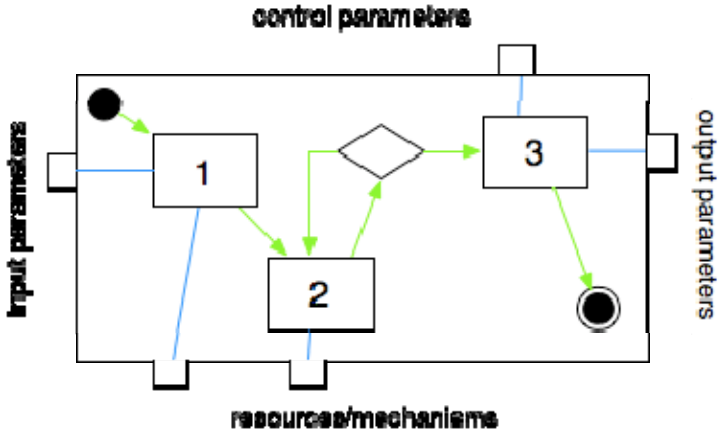
Aus: OO Modellierung, BIG, 2006

Aktivitätsdiagramm: Überprüfbare Aspekte von Daten- und Kontrollflüssen

- Basierend auf Anwendungsszenario
  - Rollen und Aktivitäten
- Detailgrad von Prozessen
  - System und Teilsysteme: Schnittstellen
  - Teilsystem und Aktionen: Detaillierter Kontrollfluss
- Ableitung von Prozessen
  - Datenfluss
    - Eingangs- und Ausgangsparameter
  - Kontrollfluss: Logische Darstellung eines Ablaufes
    - Entscheidungen und Schleifen

Überprüfung von Prozess- und Datenflussanalyse (IDEFØ)



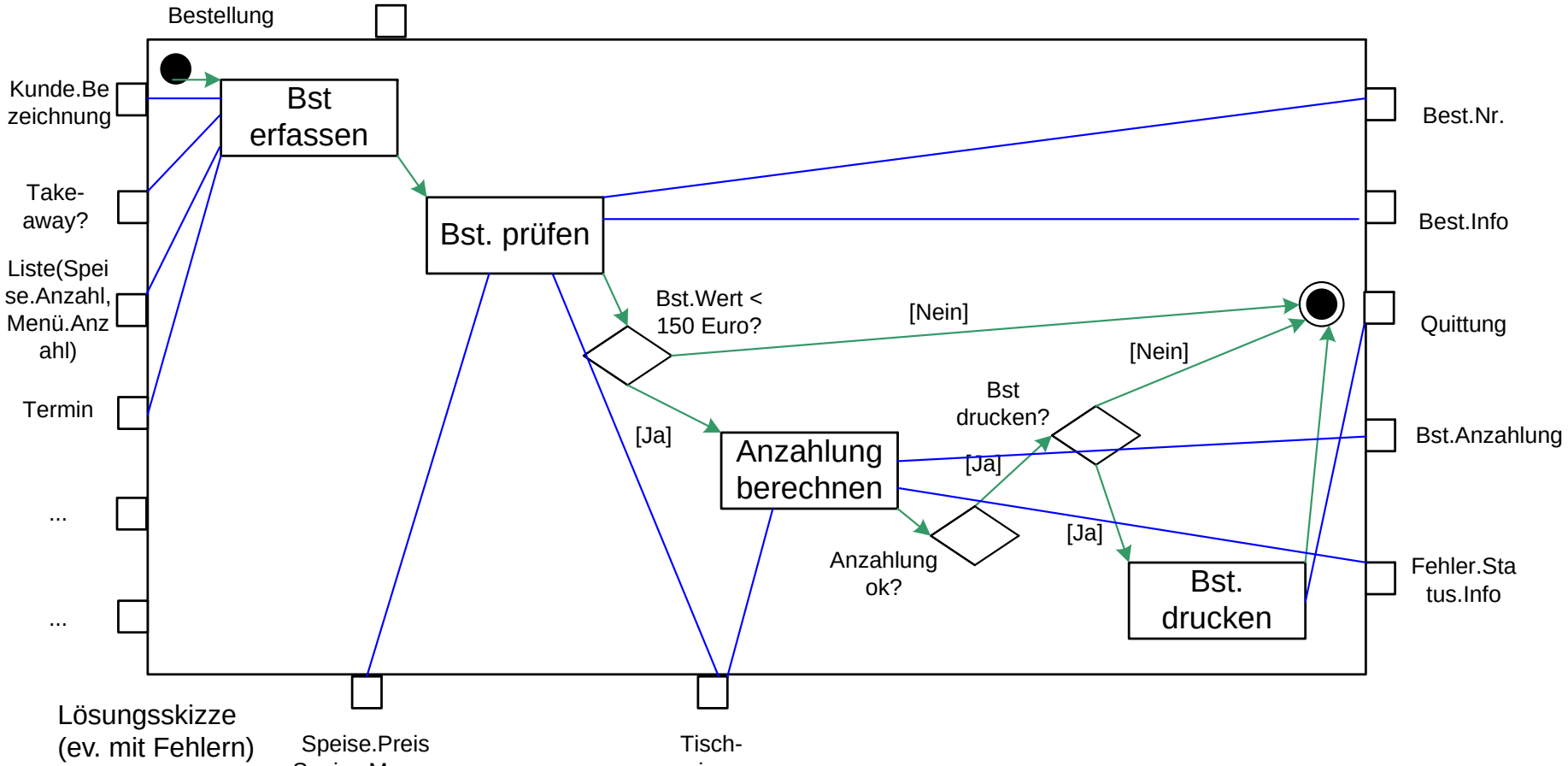


- Top-Down Modell
- Modellierung des Problembereichs
- Hilft neue Aufgabenbereiche zu erfassen und zu strukturieren

- Datenfluss (Daten-Token)
- Kontrollfluss (Kontroll-Token)
  - ergänzt um Start- und Endpunkt

Beispiel: Restaurant.Bestellung

- Datenfluss (Daten-Token)
- Ergänzt um Kontrollfluss (Kontroll-Token), Start, Ende;
- Entscheidungen
- Geltungsbereich von Daten-Token (lokal, global)
- Initialisierung aller Variablen
- Konsistenter Abschluss



Überprüfung Kontrollflussdiagramm
Review mit Anforderungen & Datenfluss

- Anforderungen strukturiert darstellen, z.B. als Liste.
- Anforderungen mit Datenfluss-Modell reviewen und konsistent machen.
  - Im Datenfluss-Modell Elemente markieren, die helfen sollen, die aktuellen Anforderungen zu erfüllen.
- Kontrollfluss-Modell systematisch durchgehen
  - Startpunkt(e) identifizieren
  - Entscheidungspunkte überprüfen: logische Bedingungen zu Entscheidungsvarianten
  - Endpunkt(e) identifizieren
- Überprüfung
  - Alle Alternativen des Anwendungsszenarios (Use Case) zur Aktivität sind abgebildet
  - An Endpunkten müssen Ausgangsparameter gültige Werte haben (entsprechend der Spezifikation der Aktivität)
  - Alle Knoten und Kanten können einer Alternative des Anwendungsszenarios zugeordnete werden.

Zustandsdiagramm
Einführung

- Ein Zustandsdiagramm (State Machine Diagram) beschreibt die möglichen Folgen von Zuständen eines Modell-elements, i.A. eines Objekts einer bestimmten Klasse
  - Während seines Lebenslaufs (Erzeugung bis Destruktion)
  - Während der Ausführung einer Operation oder Interaktion

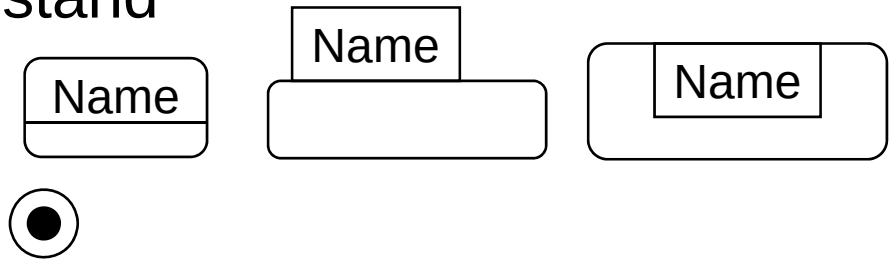
- Modelliert werden
  - Die Zustände, in denen sich die Objekte einer Klasse befinden können
  - Die möglichen Zustandsübergänge (Transitionen) von einem Zustand zum anderen
  - Die Ereignisse, die Transitionen auslösen
  - Aktivitäten, die in Zuständen bzw. im Zuge von Transitionen ausgeführt werden

Aus: OO Modellierung, BIG, 2006

Zustandsdiagramm

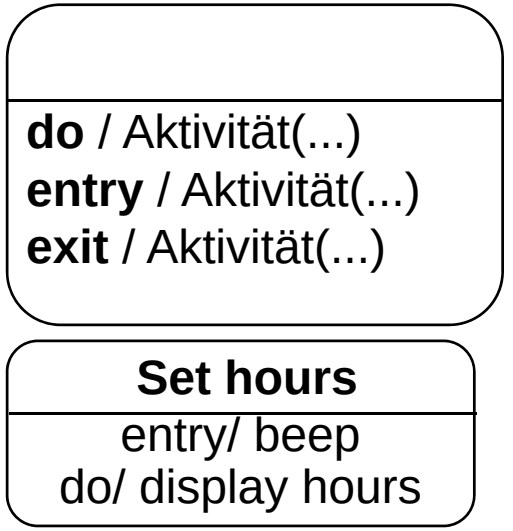
Basiskonzepte – Zustand

- Zustand (state)
  - Zustand i.e.S.
  - Endzustand



- Pseudozustände (weil transient)
  - Initialzustand
  - History-Zustand, Synch-Zustand, Gabelung, Vereinigung, etc.

- Aktivität innerhalb eines Zustands
  - entry / *aktivität*
    Aktivität wird beim Eingang in den Zustand ausgeführt
  - exit / *aktivität*
    Aktivität wird beim Verlassen des Zustands ausgeführt
  - do / *aktivität*
    Aktivität wird ausgeführt, Parameter sind erlaubt
  - *event* / *aktivität*
    Aktivität behandelt Ereignis innerhalb des Zustands



Aus: OO Modellierung, BIG, 2006

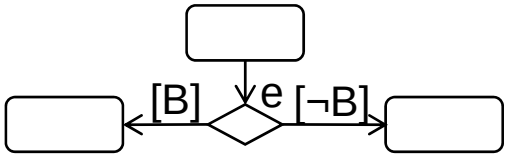
Zustandsdiagramm

Basiskonzepte – Zustandsübergang

- Ein Zustandsübergang (state transition) erfolgt, wenn
  - das Ereignis eintritt – eine evt. noch andauernde Aktivität im Vorzustand wird unterbrochen!
  - und die Bedingung (guard) erfüllt ist – bei Nicht-Erfüllung geht das nicht »konsumierte« Ereignis verloren
- Durch entsprechende Bedingungen können Entscheidungsbäume modelliert werden
- Standardannahmen
  - Fehlendes Ereignis entspricht dem Ereignis »Aktivität ist abgeschlossen«
  - Fehlende Bedingung entspricht der Bedingung [true]
- Aktionen auf Zustandsübergängen möglich
  - Spezielle Aktion: Nachricht an anderes Objekt senden

    send empfänger.nachricht()
  - Beispiel:

    right-mouse-down (loc) [loc in window]
    / obj:= pick-obj (loc); send obj.highlight()

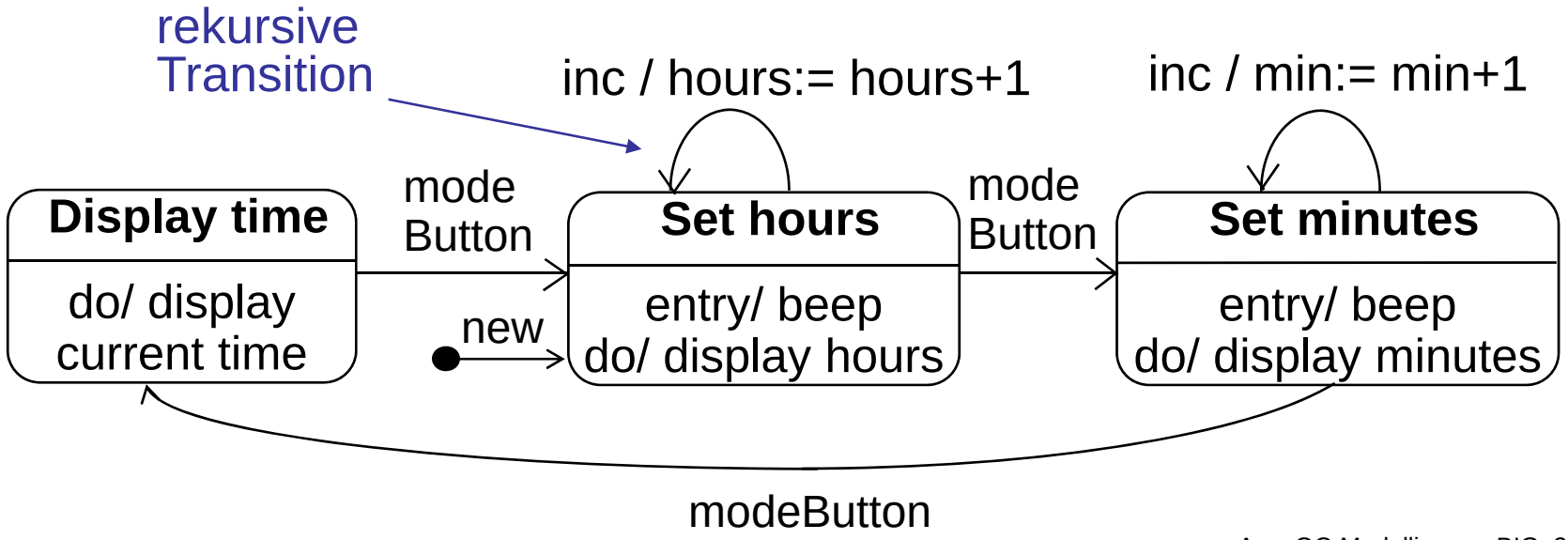


Aus: OO Modellierung, BIG, 2006

Zustandsdiagramm

Basiskonzepte – Beispiel: Klasse DigitalWatch

DigitalWatch
min : Integer = 0 hours : Integer = 0
modeButton() inc()

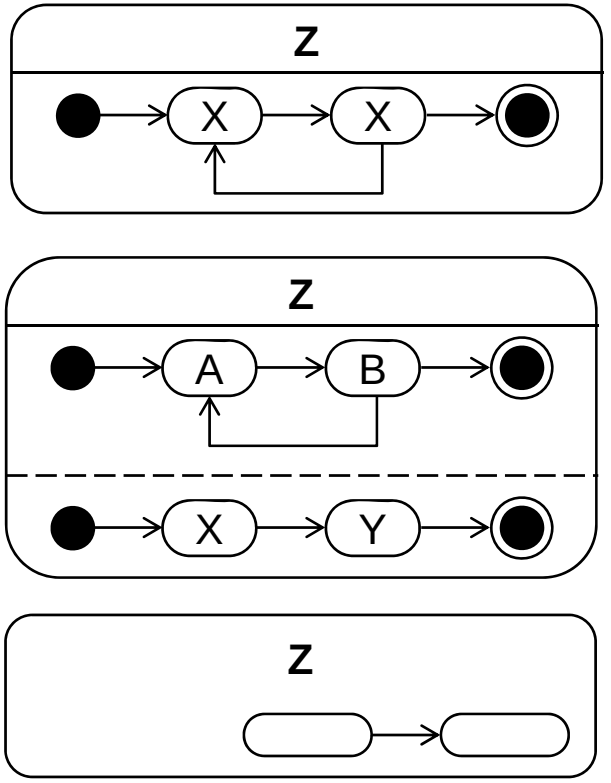


Aus: OO Modellierung, BIG, 2006

Zustandsdiagramm

Strukturierung – ODER- vs. UND-Verfeinerung

- Verfeinerung eines komplexen Zustands (composite state) — geschachteltes Zustandsdiagramm
- ODER-Verfeinerung
  - Disjunkte Sub-Zustände, d.h. genau ein Subzustand ist aktiv, wenn der komplexe Zustand aktiv ist
- UND-Verfeinerung
  - Nebenläufige Sub-Zustände, d.h. alle Subzustände sind aktiv, wenn der Superzustand aktiv ist
  - Die Subzustände werden i.A. ihrerseits Oder-verfeinert
- Hinweis auf ausgeblendete Verfeinerung
  - Diese kann an anderer Stelle dargestellt werden



Aus: OO Modellierung, BIG, 2006

Ablauf Zustandsübergangsdiagramm

- Entitäten und deren Zustände identifizieren
- Parallele Zustände aufteilen

- Für jede Entität
  - Initialzustand bestimmen
- Zustandsübergänge einzeichnen
    - Ereignis(se) bestimmen, die den Übergang anstoßen
  - Wo sinnvoll: Bedingungen für Übergänge bestimmen

- Überprüfung
  - Abläufe des Anwendungsszenarios durchspielen
  - Testfälle durchspielen
  - Überdeckung: Alle Knoten und Kanten abdecken.

Überprüfung Zustandsübergangsdiagramm Review mit Anforderungen

- Anforderungen strukturiert darstellen, z.B. als Liste.
- Anforderungen systematisch durchgehen
  - Im Zustandsübergang-Modell Elemente markieren, die helfen sollen, die aktuelle Anforderung zu erfüllen.
    - Zustände
    - Übergänge

- Zustandsübergang-Modell systematisch durchgehen
  - Nicht markierte Elemente auf Sinnhaftigkeit überprüfen
  - Startpunkt(e) identifizieren
  - Entscheidungspunkte überprüfen: logische Bedingungen zu Entscheidungsvarianten
  - Endpunkt(e) identifizieren

- Überprüfung
  - Alle Alternativen des Anwendungsszenarios (Use Case) zur Aktivität sind abgebildet.
  - Alle Knoten und Kanten können einer Alternative des Anwendungsszenarios zugeordnete werden.

Testansätze, Test-Driven Development

Testansätze, Test-Driven Development

- Testen in der Qualitätssicherung und SE Prozessen
- Teststufen und Testarten
  - Unit, Modul, System Stufen
  - Black-Box, White-Box Testarten
- JUnit, Test-Driven Development
- Beurteilung der Güte von Testfällen
- Anwendungsbeispiel “Restaurant”
  - Ableitung von Testfällen aus Anforderungen, Modellen und Entwürfen

Testen in der Qualitätssicherung

Verschiedene Teststufen:

- Unit-Test
- Modultest (Integrationstest)
- Systemtest
- Alphatest
- Betatest
- ...

Abhängig von der Testebene und dem aktuellen Anwendungsfall verfolgen Tests unterschiedliche Ziele:

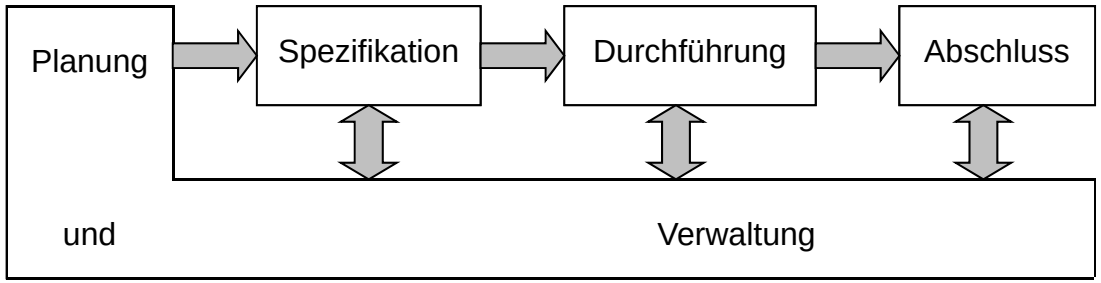
- Funktion (Funktionstest)
- Last (Lasttest)

Je nachdem, ob beim Testen die innere Struktur der Software berücksichtigt wird oder nicht, spricht man von White Box oder Black Box testen

Testen im SE Prozess

- Was tun Sie, wenn Sie die Aufgabe bekommen, ein Programm zu testen?
  - *„Unter Testen versteht man den Prozess des Planens, der Vorbereitung und der Messung, mit dem Ziel, die Eigenschaften eines IT-Systems festzustellen und den Unterschied zwischen dem tatsächlichen und dem erforderlichen Zustand aufzuzeigen.“* [Pol et al., 2000]

- Phasen in der Softwareentwicklung
  - Planung und Verwaltung
  - Spezifikation
  - Durchführung
  - Abschluss



Rollen im Team

- Alle Teammitglieder sind auch Testverantwortliche.
  - Test sind Dokumentation ("living documentation")
  - Artefakt Testplan zeigt wie Testfälle implementiert werden.
  - Eine gute Abstimmung der Rollen und Fähigkeiten im Team wird benötigt.

- Rollenbeitrag zur Qualität der Tests
  - Technische Architekten:
    - Testbarer Code („kleine", eigenständige Programme)
    - Klassendiagramme, Systemarchitektur
  - Tester:
    - In den Anfangsphasen, Black-Box Tests der Interfaces (Modul-/Integrationstests welche Anwendungsszenarien abdecken)
    - Auf Verwendung von Test-Driven Development achten. (Unit-Tests)
    - System-Tests durchführen

Ablauf Testfälle bestimmen

- Ziel aus Testplan bestimmen
  - Z.B. Normal-, Sonder- und Fehlerfälle in einem Anwendungsszenario abdecken
  - Z.B. Alle Knoten und Kanten in einem Kontrollflussgraphen abdecken
- Fälle bestimmen
  - Eingabeparameter
  - Entscheidungen im Ablauf (Wahl der Entscheidung festhalten)
- Erwartetes Ergebnis bestimmen

Äquivalenzklassenzerlegung

- Zerlegung einer Menge von Daten (Input oder Output) in Untermengen (Klassen), die äquivalente Ergebnisse oder Auswirkungen produzieren.
- Jede Klasse(nkombination) sollte mindestens einmal getestet werden.
- Dadurch sinkt die Anzahl der Testfälle und somit der Testaufwand.

Vorgehensweise:

- Finde zu testende Funktionen.
- Finde Eingabe-Vektor (A, B, C, ...) (= Faktoren) je Funktion.
  - Explizite: Parameter.
  - Implizite: globale Variablen, Systemzustand (Objekte, Datenbank), …
- Finde für jeden Faktor seine un-/gültigen Äquivalenzklassen (Klassen A1, A2, ...; B1, B2, ...; ...) und wähle je einen Wert aus jeder Klasse.
- Wahl der Kombinationen: (A1, B1), (A1, B2), ... bestimmt Intensität.

Äquivalenzklassenzerlegung

- Anforderung: 18 < Alter ≤ 65 → drei Äquivalenzklassen
  - A1: Alter ≤ 18 (ungültig)
  - A2: 18 < Alter ≤ 65 (gültig)
  - A3: Alter > 65 (ungültig)

  → Auswahl eines Repräsentanten einer Äquivalenzklasse,

  → Drei Testfälle: x ∈ A1, z.B. 10, x ∈ A2, z.B. 35, x ∈ A3, z.B. 70

Grenzwertanalyse

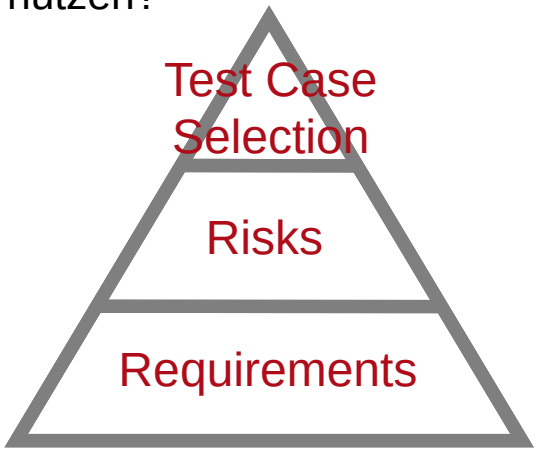
- Erweiterung der Äquivalenzklassenzerlegung für bessere Überdeckung.
- Grenzwerte werden als Klassenrepräsentanten gewählt
  → je Klassengrenze ein Testfall.

- Beispiel: Anforderung: 18 < Alter ≤ 65
  - 18 (ungültig), 19 (gültig), 65 (gültig) and 66 (ungültig)
  - Ev. mehrere Testfälle je Klassengrenze: Alter ≤ 18
  - 17 (gültig), 18 (gültig) and 19 (ungültig)
    Bedenke Tippfehler: Alter = 18!

Wertbeitrag von Software Testen

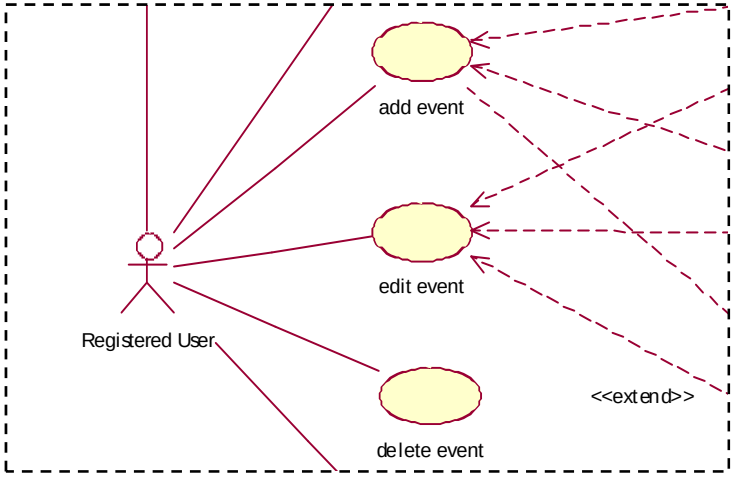
- Zentrale Fragestellung: Ist jeder Testfall, jeder Fehler, jede Anforderung gleich viel „wert“?
- Beispiel: 10 unterschiedliche Bezahlverfahren einer Online-Plattform, wobei 80% des Umsatzes immer über dasselbe Verfahren abgewickelt wird.
- Die Antwort: NEIN => Value-Based Testing.

- Requirements-Based testing
  → Welche Anforderungen bringen dem Kunden den meisten nutzen?
- Risk-Based testing
  → Welche Risiken gefährden diesen Nutzen?
- Test case selection techniques
  → Welche Testfälle addressieren dieses Risiko?

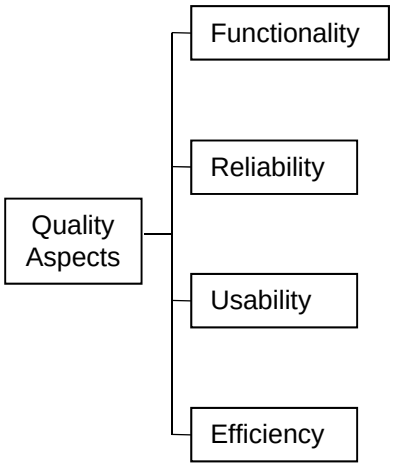


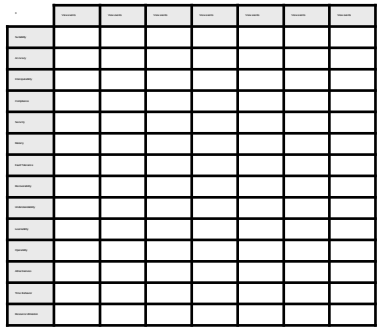
Beispiel: Risk-Based Testing (1)

Features / Anwendungsfälle



Quality Criteria





Risikoeinschätzung:
A … Critical,
B ... Important,
C ... Less important

Security aspect \ Feature	View events	View details	Add event	Edit event	Attach document	Set properties	Delete event
General identity	B	B	C	C	C	C	C
Message content authenticity	B	B	C	C	C	C	C
Message content origin	C	C	B	B	B	A	B
Integrity	C	C	A	A	A	A	
Secrecy and privacy	B	B	C	B	C	C	C
Accountability			B	B	B	B	B

Ramler et al, 2006

Beispiel für Testfallbeschreibung "Bestellung von Menü durchführen"

- Beschreibung ist die Komponente im System und eine Funktion (Ablauf aus Anforderungen)
- Vorbedingungen existieren oder eben nicht
- Eingabewerte sind konkrete Parameterbeschreibungen (Äquivalenzklassen)
- Aktionen sind Aktivitäten zur Eingabe der Werte
- Erwartete Ergebnisse sind Zustände und Ausgabeparameter
- Ergebnisse zeigen Abweichungen zum Tatsächlichen Ergebnis.
- OK/NOK zeigt ob der Test erfolgreich durchläuft.

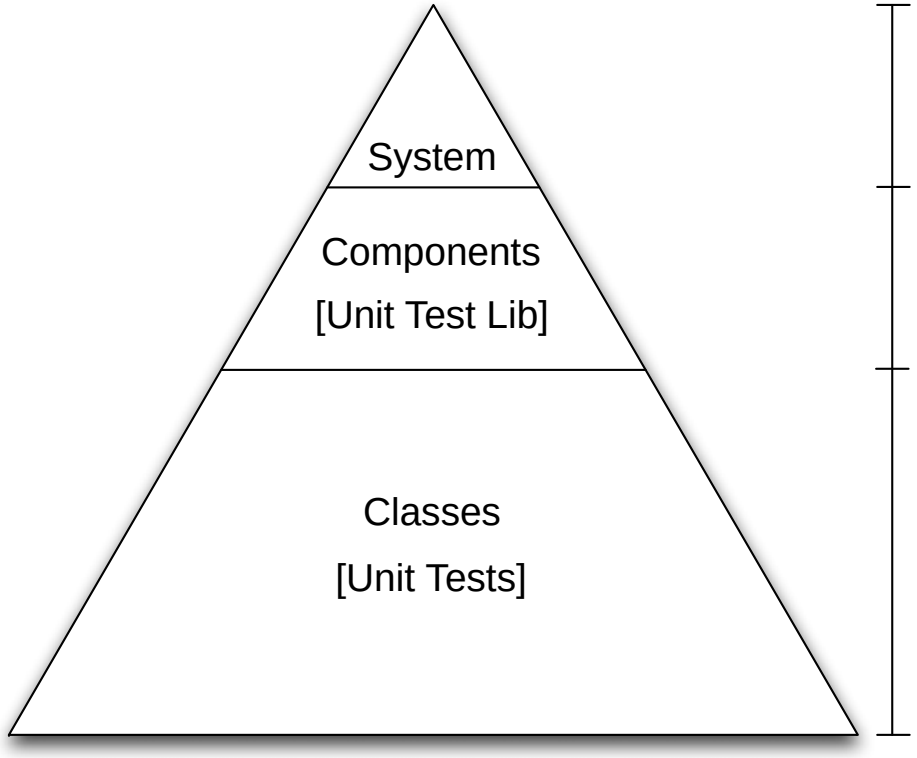
Nr.	Typ	Beschreibung	Vorbedingungen	Eingabewerte	Aktionen	Erwartete Ergebnisse	Ergebnisse	OK / NOK
34	NF	Bestellung von Menü durchführen	(Programm gestartet) (in die Bestellungs-übersicht gewechselt) Min. ein Menü mit preis<250 Euro muss vorhanden sein. Menü muss bestellbar sein.	-Bestellung. take_away -Bestellung. fürDatum -Menu.id -Menü Anzahl = 1	-Neue Bestellung erzeugen. Angabe wann die Bestellung konsumiert wird und ob "take away". -Menü wird aus einer Liste ausgewählt -Anzahl angeben -Bestellung speichern	Bestellung ist in der Datenbank gespeichert und enthält genau ein Menü. Das Attribut Bestellung.storniert ist auf den Booleanwert ‚false' gesetzt, der Primärschlüssel von Bestellung wird von der DB automatisch belegt.	Bestellung ist in der DB gespeichert, jedoch wird bei der Abfrage dessen kein Primärschlüssel mitgeliefert.	NOK

Testfall-Spezifikationsmethoden

- Strukturierte Anweisungen zur Generierung von Testfällen
- Vorteile gegenüber zufälliger Testfalldefinition
  - Höhere Qualität der Testfälle
  - Machen Qualität und Intensität eines Tests transparent
  - Erreichen gewünschte Abdeckung für jeden Teil der Applikation
  - Effektiver für gegebene Fehlertypen
  - Tests werden nachvollziehbar
  - Testprozess wird unabhängig von Personen
  - Testfall-Spezifikationen werden übertragbar und aktualisierbar
  - Testprozess wird besser plan- und steuerbar
- „Nachteile“
  - Fundiertes Wissen der Tester notwendig

Testen - Gesamtsystem

- Die Fläche des Pyramidenabschnitts ist proportional zu der Anzahl der jeweiligen Tests welche bei Auslieferung vorhanden sein sollten.



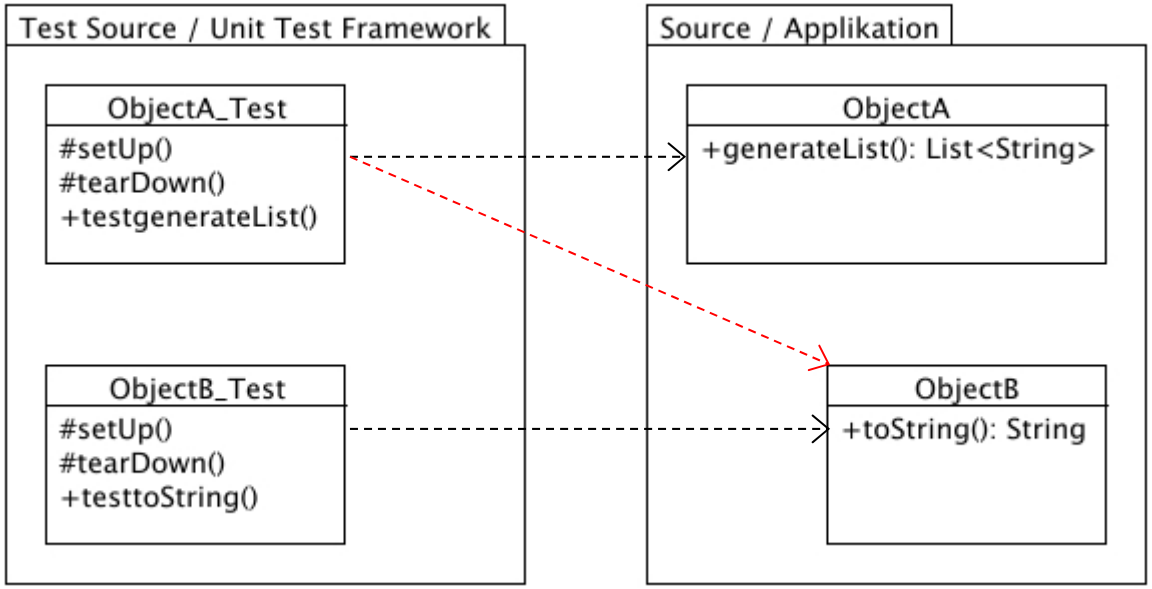
- System Tests
- Integrations/Modul Tests
- Unit Tests

Unit Tests

- Ziel: Dokumentation von Klassen
- Units werden jeweils einzeln gegen ihre Spezifikation getestet: Übergabe von Daten, Prüfung des Outputs.
- Unittests erlauben eine genaue Lokalisierung und frühe Erkennung von Implementierungsfehlern.
- Unittests können bei größeren Systemen sehr aufwendig werden; daher ist bei größeren Tests besonderer Wert auf strukturiertes Vorgehen und Dokumentation des tatsächlichen Vorgehens zu legen.
- Automatisierung essentiell da häufige Regressiontest nach Änderungen notwendig

Unit Testing

- Bei einem Unit-Test in einer OO Sprache liegt der Fokus auf einer Klasse oder eine Methode (Unit == Klasse)
- Alle benötigten Daten passen in den Arbeitsspeicher
- Unit-Tests müssen schnell durchlaufen (Regression-Library)



Module

- Ein Modul hat folgende Eigenschaften
  - Es ist das Werk eines Entwicklers
  - Es hat eine dokumentierte Spezifikation (API)
  - Es ist ein sichtbares, identifizierbares Produkt mit expliziter Integration ins Gesamtsystem (z.B. mit Interfaces)
  - Es kann kompiliert/interpretiert und getrennt von anderen Modulen getestet werden (z.B. mit Mocking)
  - Es ist notwendig

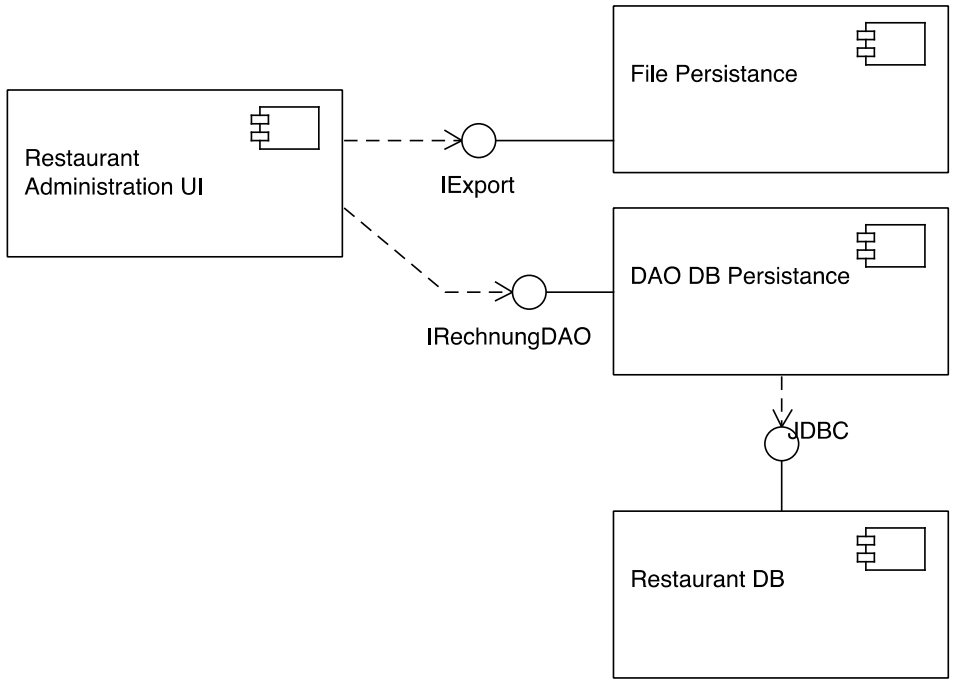
Modultests

- Ziel: Aufspürung von Fehlern in der Implementierung
- Module werden jeweils einzeln gegen ihre Spezifikation getestet: Übergabe von Daten, Prüfung des Outputs
- Modultests erlauben eine genaue Lokalisierung und frühe Erkennung von (Implementierungs-)Fehlern
- Modultests können v.a. bei größeren Systemen sehr aufwendig werden, weshalb sie dann unstrukturiert und undokumentiert gestaltet werden – das ist falsch!

Integrationstests

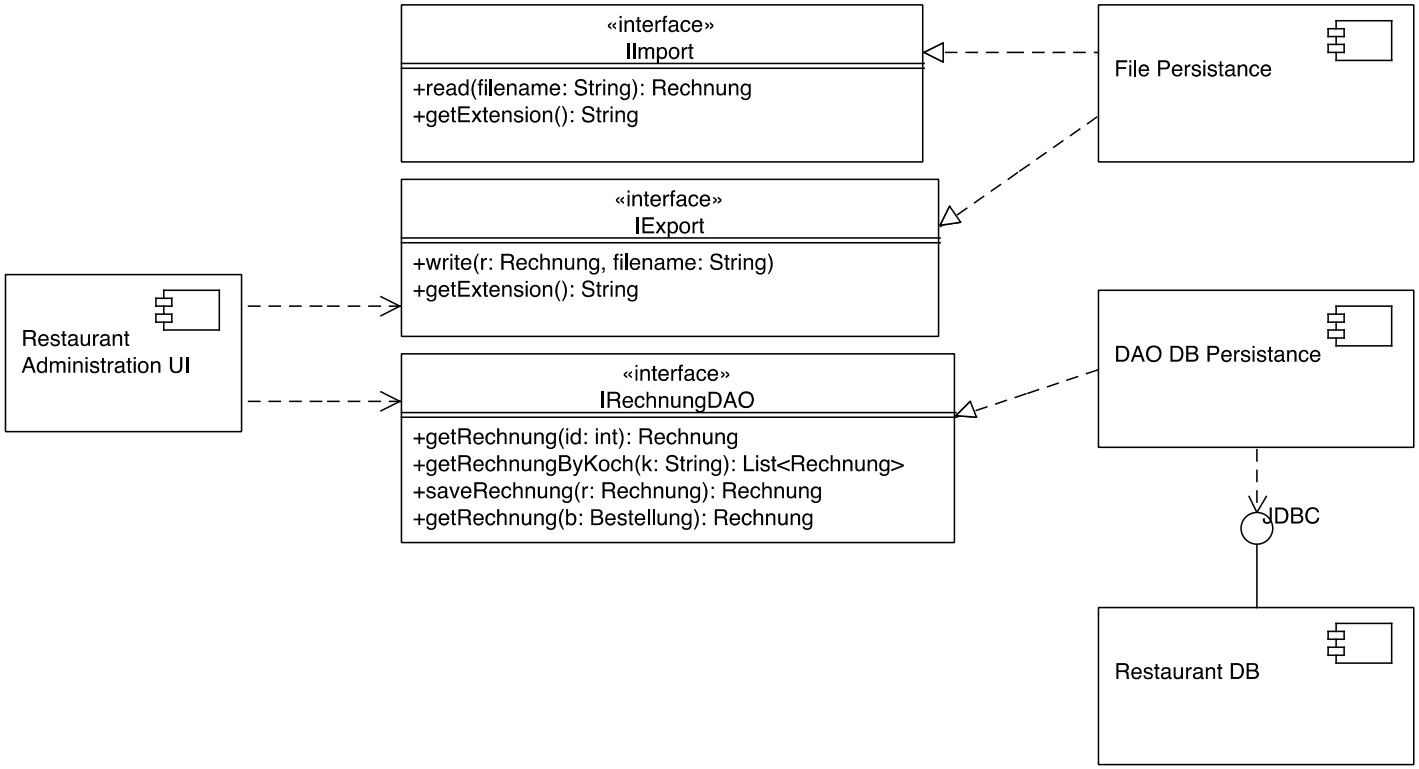
- Ziel: Test der Interaktion zwischen Modulen
- Zusammenführung von Modulen zu größeren Strukturen
  - „Big Bang“
  - Inkrementell: Top-Down oder Bottom-Up
- Häufig: Zusammenführung durch Entwickler, Test durch Tester
- Inkrementelles Testen ist vorzuziehen
  - Weniger Aufwand (weniger Stubs/Treiber nötig)
  - Frühere Erkennung von Schnittstellenfehlern
  - Fehler-Lokation besser eingrenzbar (leichteres Debugging)
  - Weniger Gefahr der Überdeckung von Fehlern in einem Modul durch Fehler in einem anderen Modul.

Beispiel für Module Komponentendiagramm



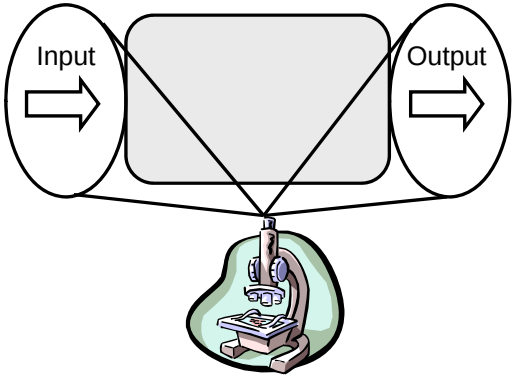
- Komponenten bieten Interfaces an (provides)
- Komponenten erfordern andere Interfaces (requires)
- Dies macht Komponenten zu abstrakten, austauschbaren Modulen

Komponentendiagramm Alternativnotation

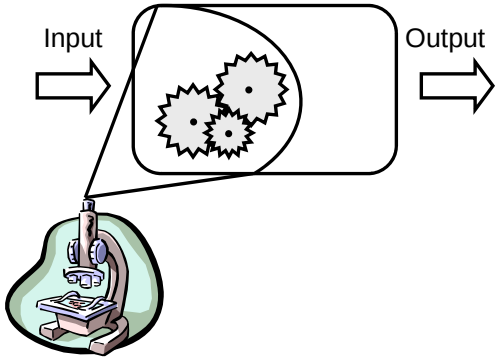


Black Box vs. White Box

- Black Box
  - Basiert auf Spezifikation
  - Wissen über innere Struktur wird ignoriert
  - Daten-getriebene, Input-Output-getriebene Tests
  - Anforderungsüberdeckung
  - Partitionierung der Eingabe-Daten



- White Box (Glass Box)
  - Basiert auf Struktur von Code bzw. SE-Modellen
  - Wissen über innere Struktur notwendig
  - Logik-getriebene Tests
  - Überdeckung von Knoten, Kanten, Pfaden
  - Partitionierung von Daten für Bedingungsauswertung



Teststufen und Testarten in SE

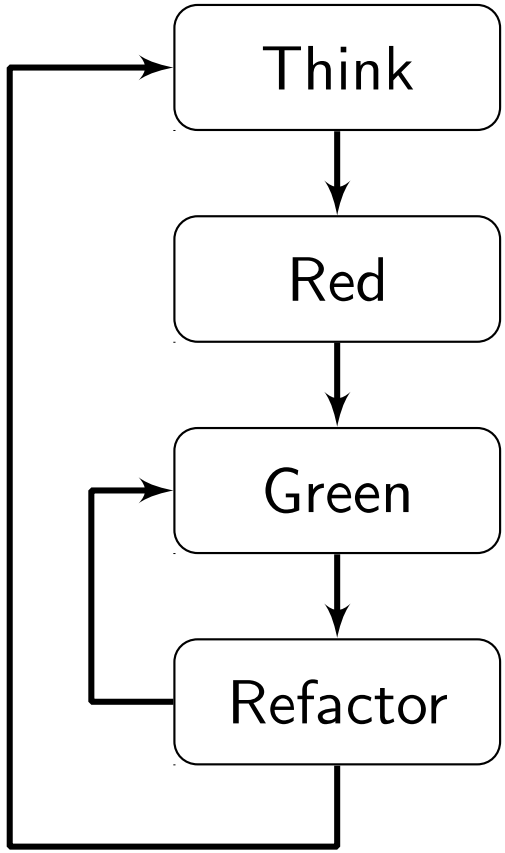
- Teststufe (Unit, Modul, System)
  - Je nach Teststufe ist meist ein anderes Testframework vorzuziehen
  - Jedoch können mit JUnit auch Modul Tests geschrieben werden
  - System Tests (z.B. GUI Tests) sehr mühsam
- Testart (Black-/Whitebox)
  - Unit Tests werden meist als Blackbox ausprogrammiert (TDD)
  - Whiteboxtests sind meist Modul oder Systemtests, sind jedoch visuell kaum von Blackboxtests zu unterscheiden

Test-Driven Development

- Unit Tests: Ablauffähige Testfälle erstellen
- Assertions für „korrekte“ Durchführung des Testfalls ableiten (= erwartetes Ergebnis)
  - Normalfall soll nicht fehlschlagen
  - „Korrekter“ Fehlerfall muss auch wirklich fehlschlagen -> Exception

- Brauchbare Modellierung unterstützt das Herstellen effektiver und effizienter Testfälle
  - Datenmodellierung (Datenbank)
  - Datenfluss und Kontrollfluss (Business Logic)
  - Zustandsübergangsmodelle (GUI)

Test-Driven Development Prozess



1. *Think*: Spezifikation des Tests
2. *Red*: Implementierung des Tests
   - Test schlägt fehl!
3. *Green*: Implementierung der zu-testenden Klasse oder Komponent.
   - Test ist erfolgreich!
4. *Refactor*: Veränderung einer Implementation -> Funktionalität bleibt erhalten -> Test sollte nie wieder fehl schlagen!
   - Lernen aus Test, Veränderung des Prozess

Best-Practices für Testen

- Eine gute Abstimmung der Rollen im Team wird benötigt
- Die Zeit für viele Unit Tests ist wohl investiert da sie die Integrations-Tests wesentlich vereinfachen. (bottom-up testing)

- Technische Architekten:
  - „kleine" Programme, „kleine" Methoden
  - Eigenständig Instanzierbare Klassen
    - Dependency Injection Pattern!
  - Klassendiagramme, Sequenzdiagramme

- Tester:
  - Testet die Objekte so wie sie nachher im Programm verwendet werden

JUnit, .NETUnit etc

- Entwickler erstellt Test-Methoden (Testfälle) während oder sogar vor der Programmierung
- Testfall: Aufruf von Methoden mit bestimmten Parametern, Überprüfung von Ergebniswerten bzw. –zuständen
- Herstellung eines Ausgangszustands innerhalb des Testfalls oder durch andere Testfälle
- Framework erlaubt einfache Erstellung, Ausführung und Auswertung der Testfälle
- somit einfache Wiederholung von Tests z.B. nach Änderungen und Korrekturen möglich (Regressionstests)

Hinweise / Referenzen

- Die Zeit für viele Unit Tests ist wohl investiert, da die Unit Tests die Integrationstests wesentlich vereinfachen (bottom-up testing).
- Test Cases sind „living documents” - sie unterstützen die Kommunikation innerhalb des Teams.
- Formuliert einen wesentlichen Testfall eures eigenen Projektes aus. Wie kann dieser Testfall mit JUnit implementiert werden?

- http://junit.sourceforge.net/doc/testinfected/testing.htm
- http://www.junitdoclet.org/

Beurteilung von Testfällen

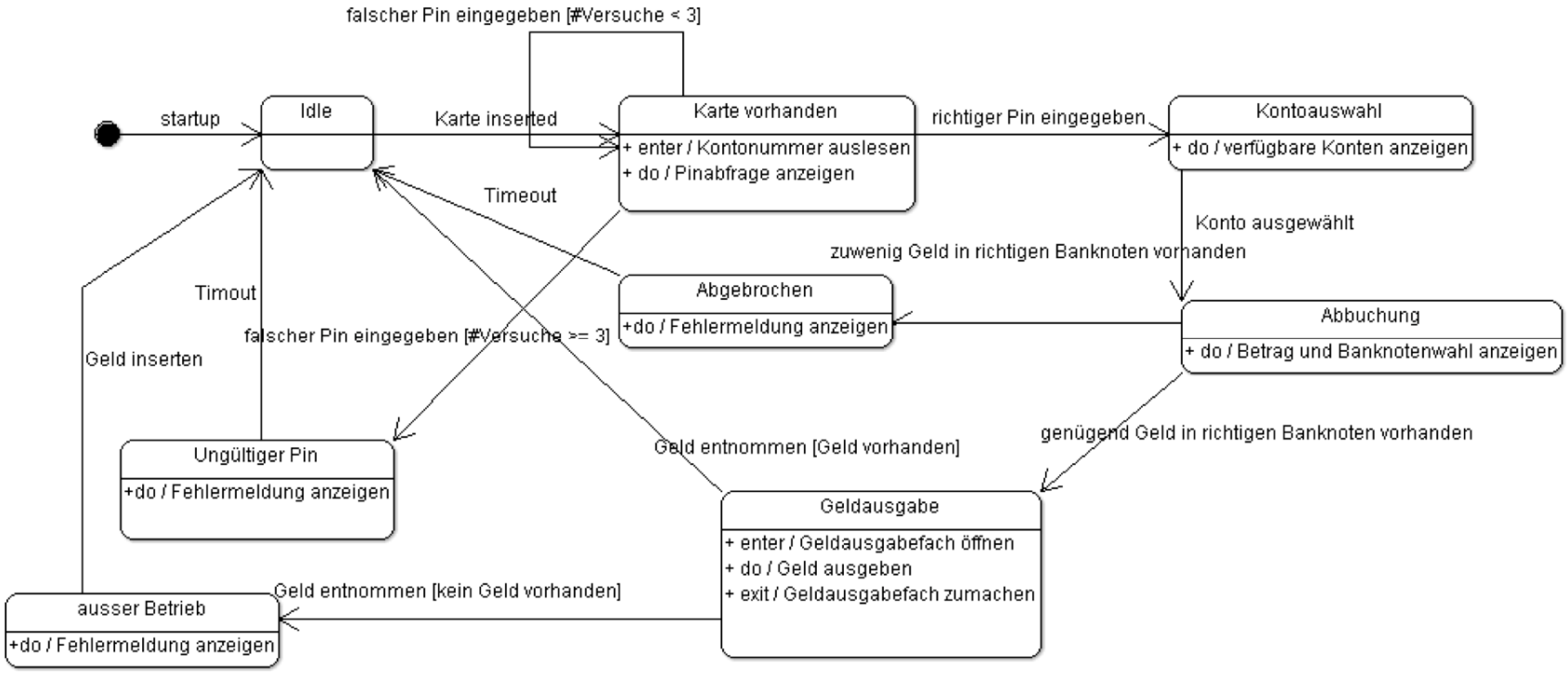
- Ziel aus Testplan bestimmen (Referenz auf Modell)
  - Z.B. Normal-, Sonder- und Fehlerfälle in einem Anwendungsszenario abdecken
  - Z.B. Alle Knoten und Kanten in einem Kontrollflussgraphen abdecken

- Erreichung des Testziels mit Menge der spezifizierten Testfälle überprüfen.

Beispiel Testfälle Cashmachine

- I. Prüfung Abhebewunsch
  - Äquivalenzklassen der Parameter
  - Mögliche Ergebnisse
- II. Berechnung des Geldbetrags

Cashmachine State Chart



Zusammenfassung

- Test-Driven Development als Basis für zielorientierte Software-Entwicklung (Implementierung und QS)
- Brauchbare Modelle unterstützen das Herstellen effektiver und effizienter Testfälle
  - Datenmodellierung (Datenbank)
  - Datenfluss und Kontrollfluss (Business Logic)
  - Zustandsübergangsmodelle (z.B. GUI)
- Reviews helfen, die Anforderungen und Modelle vorab zu überprüfen, um eine solide Basis für Testfälle herzustellen.

Referenzen

Bücher und Skripten

- „Objektorientierte Modellierung“; Unterlagen zur Vorlesung; Business Informatics Group (BIG), TU Wien, 2006.
- „Datenmodellierung“; Unterlagen zur Vorlesung; TU Wien, Inst. f. DB & AI, 2006.
- „ Testen als Entwickler“, Unterlagen zur Vorlesung Qualitätssicherung, Inst. f. Softwaretechnik und Interaktive SystemeM; 2006
- Kemper A., Eickler A.; „Datenbanksysteme“; 6. Auflage, Oldenbourg, 2007.
- Martin Pol, Tim Koomen, Andreas Spillner: “Management und Optimierung des Testprozesses: ein praktischer Leitfaden für erfolgreiches Testen von Software, mit TPI und TMap”, dpunkt.verlag, 2000, ISBN 3-932588-65-7
- [Somm10] Sommerville I.; Software Engineering; 9th ed., Addison Wesley, 2010.
- Spillner et al., "Basiswissen Software-Test", dPunkt, 2003
- Thaller, "Software-Test", dPunkt, 2002
- [Wall11] Wallmüller, E.; Software Quality Engineering; 3. Auflage, Hanser, 2011.

Web Ressourcen

- SEPM Tuwel: https://tuwel.tuwien.ac.at/course/view.php?id=33
  - Prüfungsordner
- Sommerville book companion website: www.pearsoned.co.uk/sommerville
  - Case studies