# Software-Qualitätssicherung VU
# Block 2

Stefan Biffl, Dietmar Winkler

## Qualitätskontrolle und Fehlerreduktion
## Reviews und Inspektionen

# QS VU Zeitplan

| Datum | Thema |
|---|---|
| **VO-Teil:** 13.10. | Einführung Qualität und deren Sicherung; Begriffsdefinitionen; Qualitätsplanung; Messen von Qualität |
| 20.10. | Qualitätskontrolle und Fehlerreduktion; Reviews und Inspektionen |
| 3.11. | Ableiten von Testfällen aus SE-Modellen: Methoden der Verifikation und Validierung |
| 10.11. | Testen: Testprozess als Rahmen; Fokus auf Unit- und Modul-Test |
| 17.11. | Test-Driven Development, Refactoring, Testautomatisierung |
| 1.12. | Testen in agilen Prozessen |

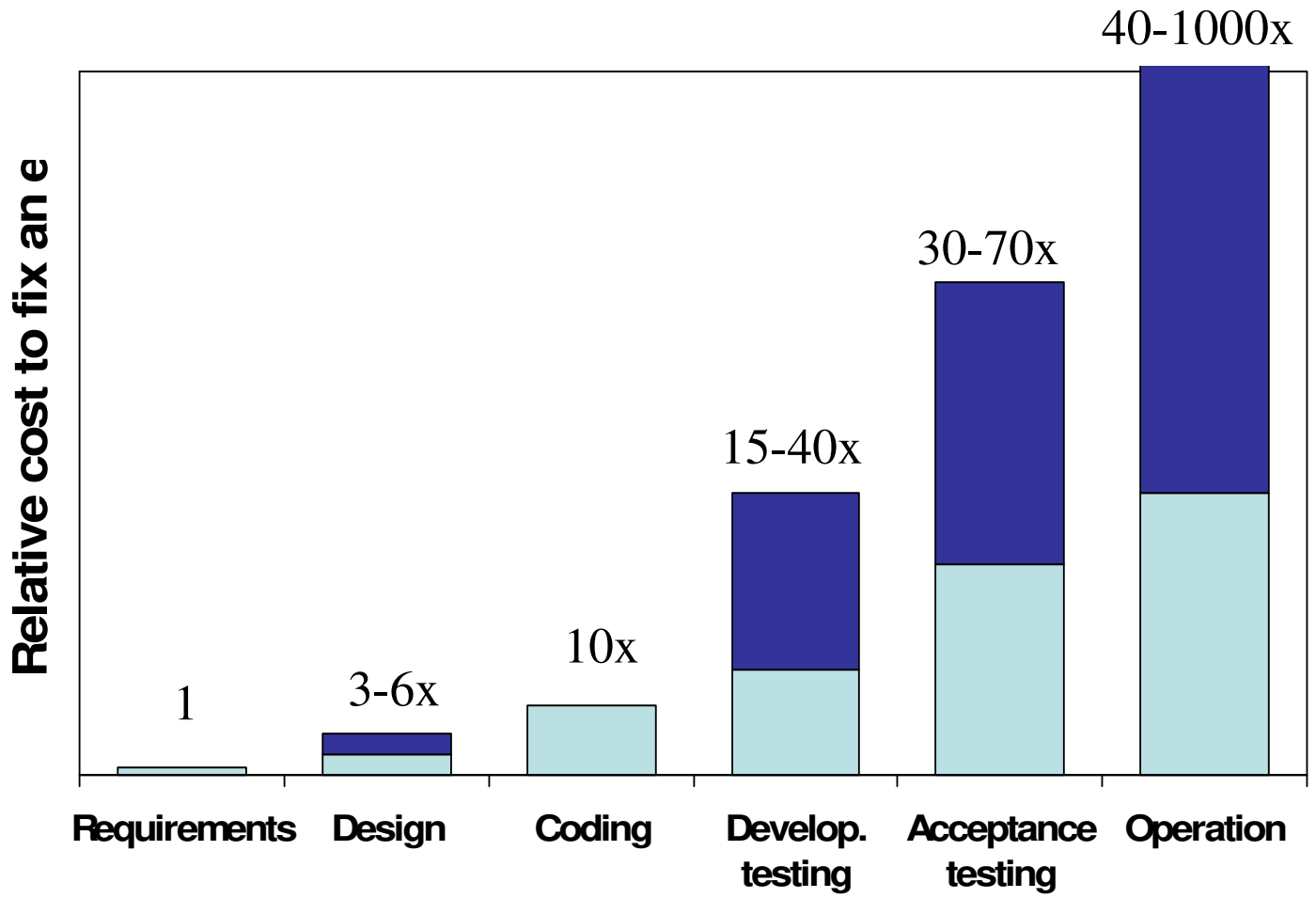
# Qualitätskontrolle & Fehlerreduktion

Reviews & Inspektionen

# Motivation

- Die Korrektur von Fehlern in Softwareprodukten kann hohe Aufwände und Kosten verursachen.
- Je später ein Fehler erkannt wird, umso höher sind diese Aufwände.
- Zielsetzung ist daher, Fehler zu vermeiden oder Fehler frühzeitig zu erkennen und zu beheben.
- Analytische Methoden, wie Reviews, Inspektionen und Audits dienen der Verbesserung der Produkt- und Prozessqualität in allen Phasen der Softwareentwicklung, speziell aber in frühen Phasen der Softwareentwicklung.
- Reviews und Inspektionen erfordern keinen ausführbaren Code, sondern können somit bereits auf z.B. Anforderungsdokumente, Designdokument angewandt werden.

# Korrekturaufwand von Fehlern im Projektverlauf




**The cost of fixing errors escalates as we move the project towards field use.**

From an analysis of sixty-three projects cited in Boehm Barry, „Software Engineering Economics“, Prentice-Hall, 1981

# Defect Reduction Heuristics

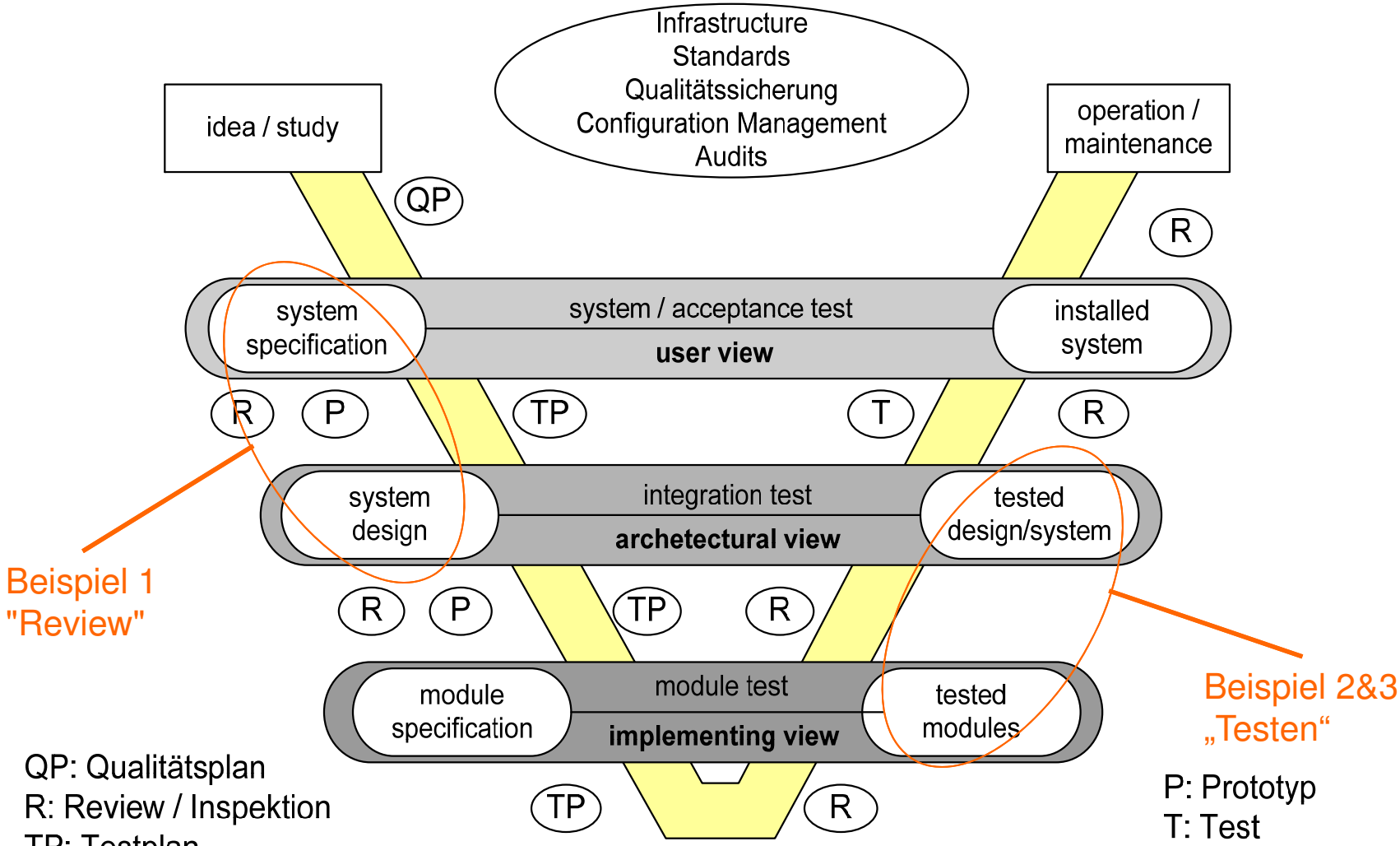
Boehm and Basili; Software defect reduction report [Boehm and Basili, 2001]

- Finding and fixing a software problem after delivery is often 100 times mores expensive than finding and fixing it during the requirements and design phase.
- Current software projects spend about 40 to 50% of their effort on avoidable rework.

- About 80% of avoidable rework comes from 20% of the defects.
- About 80% of the defects come from 20% of the modules, and about half of the modules are defect free.
- About 90% of the downtime comes from, at most, 10% of the defects.

- Peer reviews (i.e., inspections) catch 60% of the defects (on average; however, there is considerable variation of effectiveness).
- Perspective-based reviews catch 35% more defects than non-directed reviews.
- Disciplined personal practices can reduce defect introduction rates by up to 75%.

# QS im Kontext mit dem V-Modell

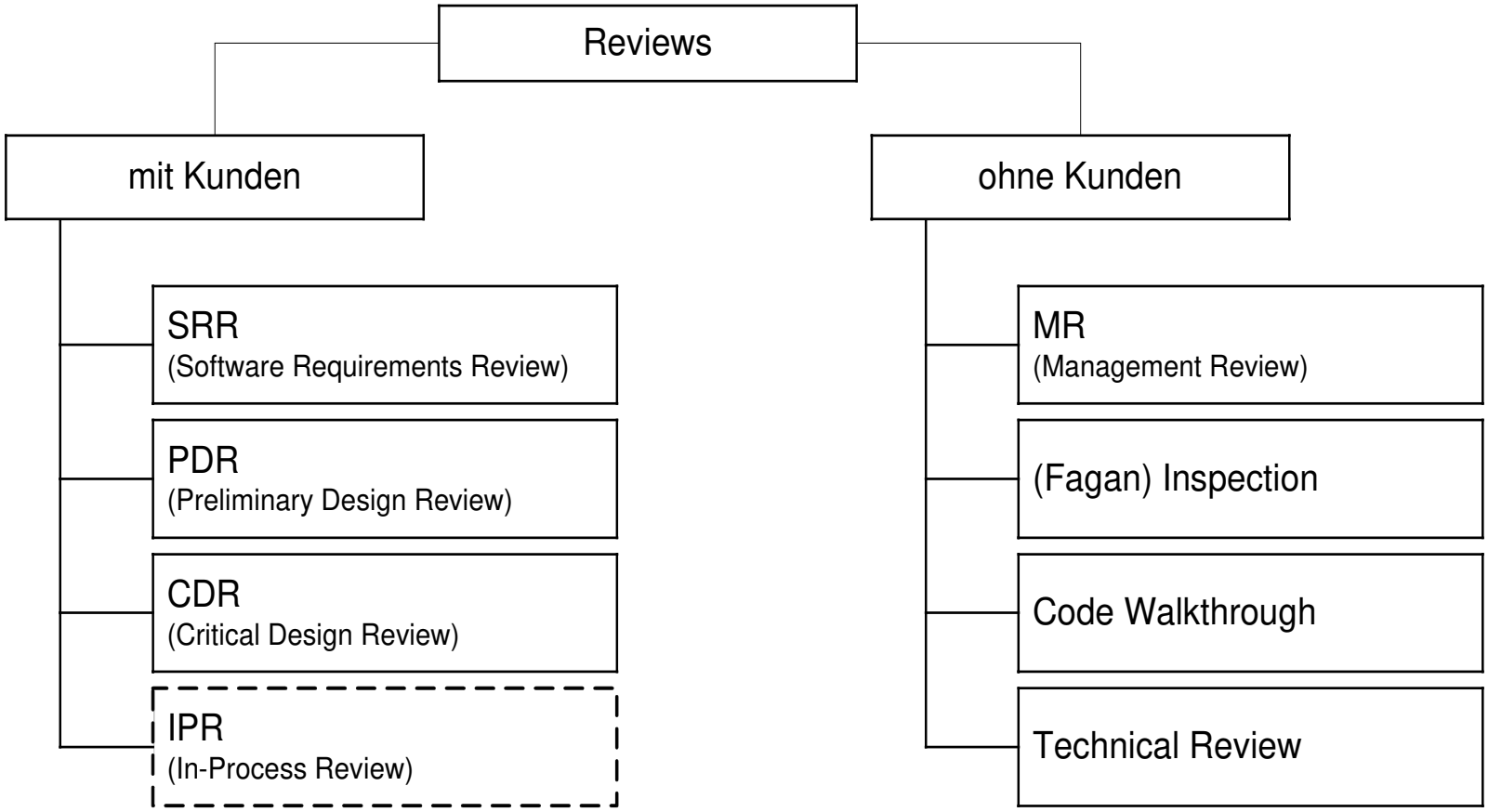


# Reviews

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]



# Vergleich Review, Inspektion, Walkthrough

| (Technisches) Review | Inspektion (formales Review) | Code Walk-Through (abgeschwächtes Review) |
|---|---|---|
| Ziele<br>▪ Stärken und Schwächen des Prüfobjektes identifizieren<br>▪ (Entwicklungsprozess verbessern) | Ziele<br>▪ Schwere Defekte im Prüfobjekt identifizieren<br>▪ Entwicklungsprozess verbessern<br>▪ Metriken ermitteln | Ziele<br>▪ Defekte und Probleme des Prüfobjektes identifizieren<br>▪ Ausbildung von Benutzern und Mitarbeitern |
| Teilnehmer<br>Moderator, Autor, Gutachter, Protokollführer | Teilnehmer<br>Moderator, Autor, Gutachter, Protokollführer, (Vorleser) | Teilnehmer<br>Autor (=Moderator), Gutachter |
| Charakteristika<br>▪ Ausgebildeter Moderator<br>▪ Prüfteam gibt Empfehlung an Manager | Charakteristika<br>▪ Ausgebildeter Moderator<br>▪ Prüfobjekt wird vom Vorleser Absatz für Absatz vorgetragen<br>▪ Moderator gibt Freigabe | Charakteristika<br>▪ Prüfobjekt wird vom Autor ablauforientiert vorgetragen<br>▪ Autor entscheidet |

# Rollen

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

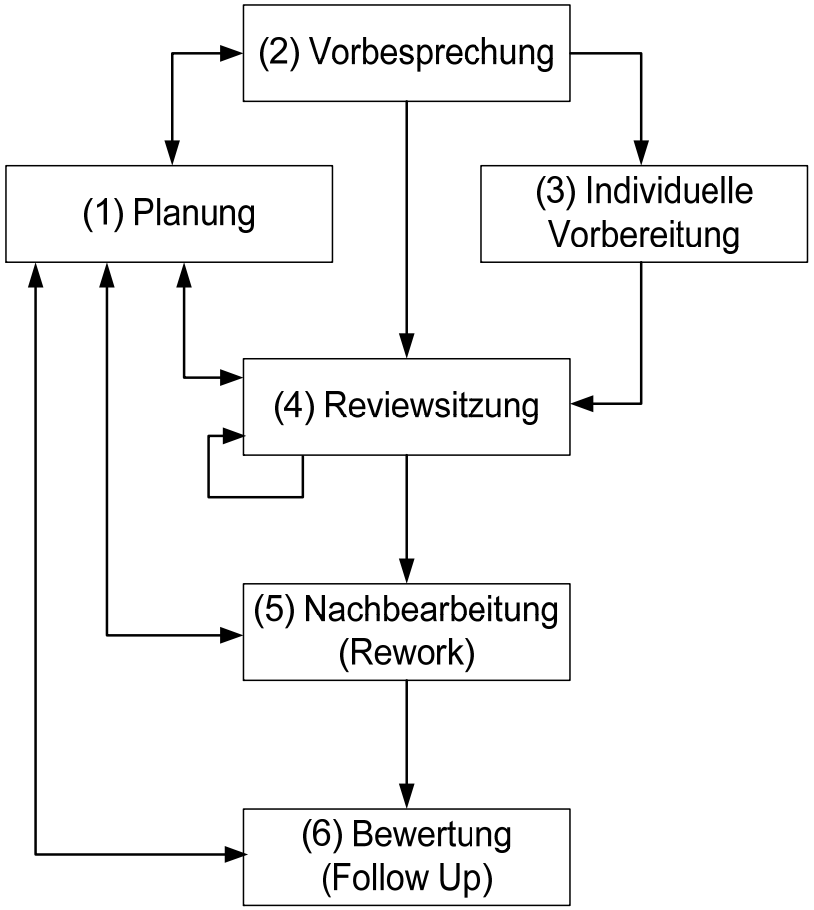
  - Moderator („keeper of the process“)
    Leiter des Reviews
  - Leser („keeper of focus and pace“)
  - Gutachter („Reviewer“)
    Kommentierung des Reviewobjektes.
  - Schreiber („preserver of knowledge“)
    Protokollschreiber verfasst Protokoll
    (Notizen während der Reviewsitzung)

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

# Wer soll an einem Review teilnehmen?

| Personen \ Produkte | Projektplan Analyse | Anforderungen | Analysemodell | Testplan | Projektpl. Realisierung | Entwurf | Implementierung | Installation |
|---|---|---|---|---|---|---|---|---|
| Kunde | ● | ● | | | ● | | | ● |
| Projektleiter | ● | ● | | | ● | | | ● |
| Analytiker | | ● | | ● | | ● | | |
| Integrator | | | | | | ● | ● | ● |
| Entwickler | | | | | | ● | ● | |
| Tester | | | | ● | | | ● | |
| Qualitätsicherer | | | | ● | ● | | | |

# 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

# Reviews in SE&PM

- Welche Arten von Reviews würden in einem SE&PM Projekt Sinn machen?
  - 4 bis 6 Personen
  - 16 Wochen
  - Administrative Aufgabenstellung
- Wie viele Reviews würden Sie einplanen?
  - An welchen Punkten im Projekt?
  - Welche Artefakte wären besonders sinnvoll zu reviewen?
  - Wie viele Personen sollen am Review teilnehmen?
  - Worauf wäre besonders zu achten?
- Welche Effekte erwarten Sie auf das Projekt/Team?

# Richtlinien für technische Reviews

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

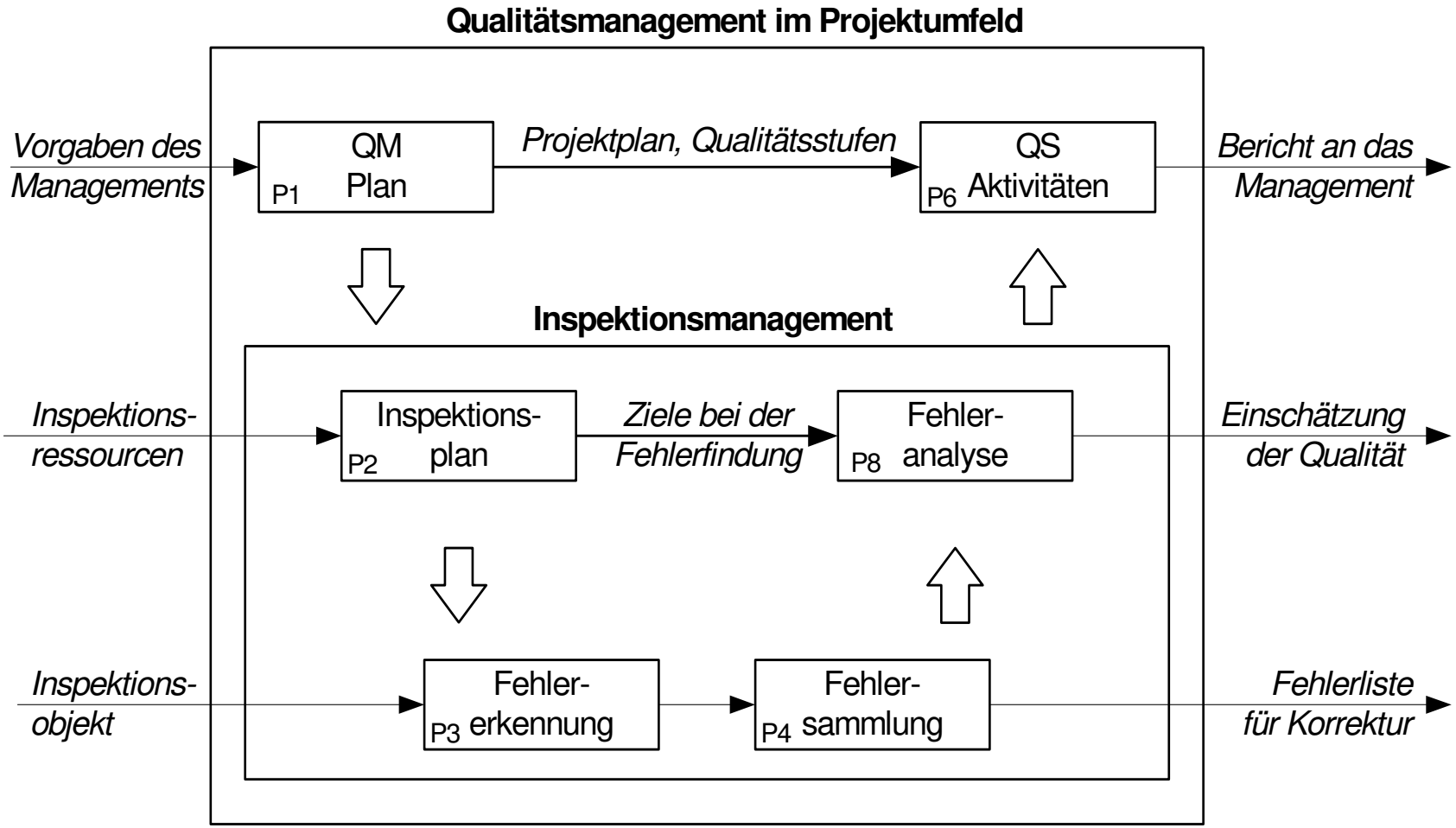
# Inspektionen

- Inspektionen sind spezielle Reviews, die speziell in frühen Phasen der Softwareentwicklung angewandt werden können.
- Inspektionen sind formale, effiziente und wirtschaftliche Methoden um Fehler im Design und Code zu finden.
- Inspektionen ermöglichen eine systematische und strukturierte Fehlererkennung.

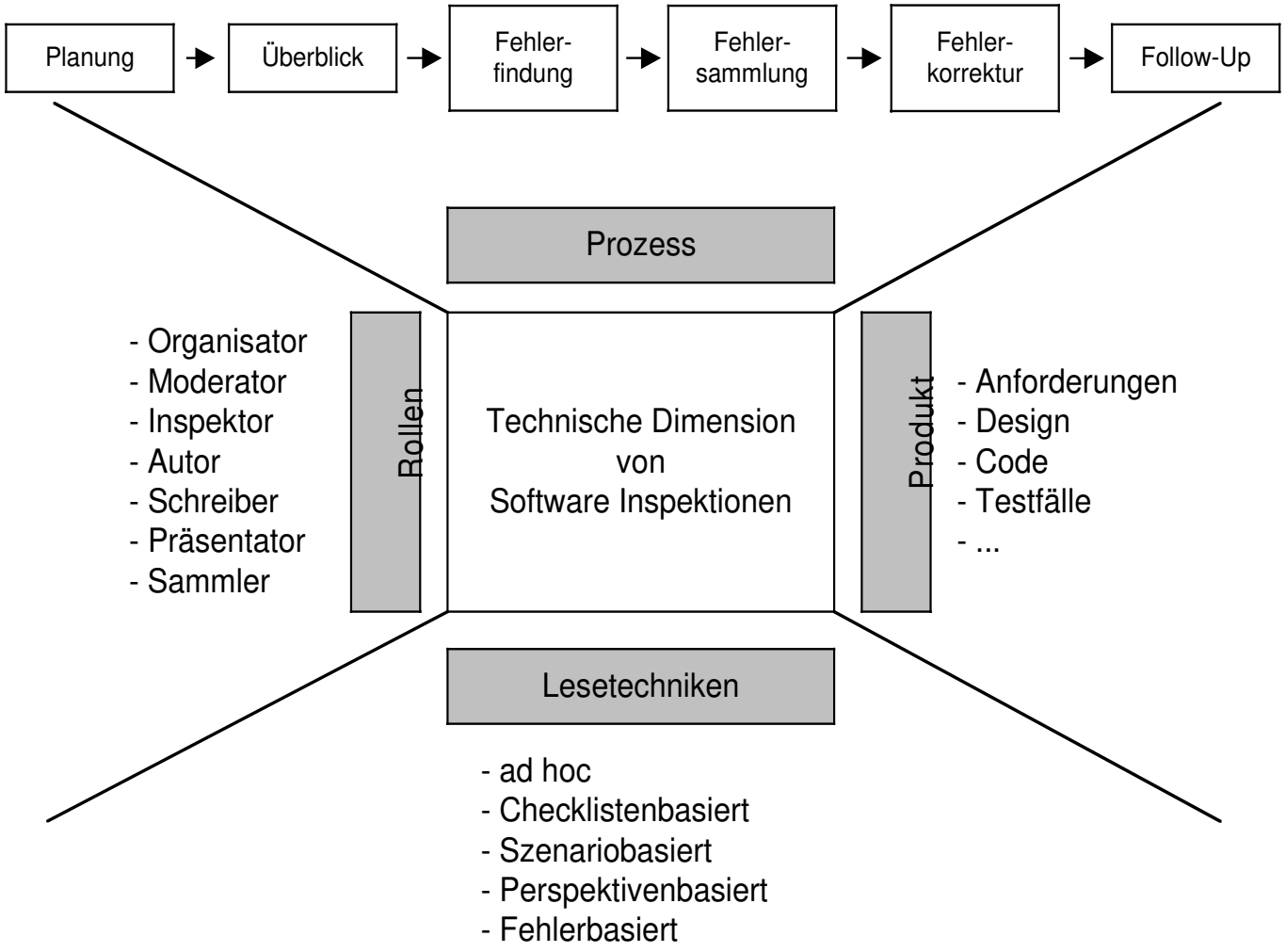
- Lesetechniken unterstützen die Inspektoren durch fokussiertes Lesen (Perspektiven, Rollen, Szenarien, Checklisten, usw.)
- Teammeetings dienen der Sammlung individueller Fehlerlisten (reale vs. nominelle Teams).

- Wichtige Anwendung: Fehlerfindung in Anforderungs- und Designdokumenten.

# Inspektionsplanung und Kontrolle



# Technische Dimension von Inspektionen




[Laitenberger, 2000]

# Lesetechniken zur Fehlerfindung (1/2)

- **Lesetechniken sind strukturierte Ansätze oder Dokumente, die den Fehlerfindungsprozess während einer Inspektion oder einer Review unterstützen.**

- **Arten von Lesetechniken (Auswahl)**
  - **Ad-hoc (unstrukturiertes Lesen)**
  - **Checklistenbasiertes Lesen**
    - Vordefinierte Fragestellungen die sequentiell auf das zu untersuchende Objekt angewandt werden.
    - Die Checklisten sind auf zu findende Fehlertypen, Fehlerorte (Design, Code) usw. abgestimmt.
    - In der Praxis meist generische Checklisten + Projektspezifische Erweiterungen.
    - Einfache Anwendbarkeit durch Inspektoren.

# Lesetechniken zur Fehlerfindung (2/2)

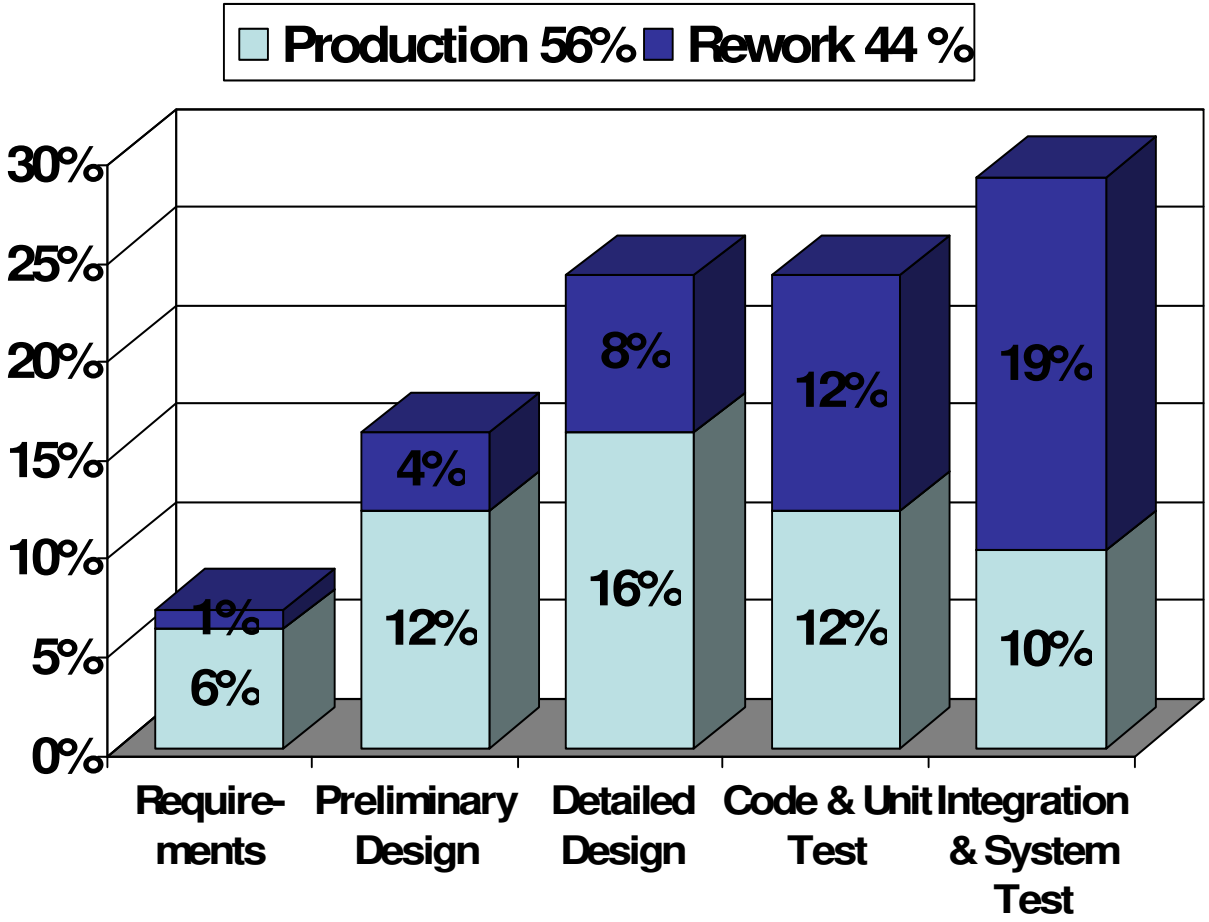
- **Perspektivenbasiertes Lesen**
  - Fehlersuche aus verschiedenen Perspektiven: User, Tester, Entwickler.
  - Zielsetzung: unterschiedliche Fehler aus individuellen Sichten finden.
  - Erfordert entsprechende Fachspezialisten.

- **Usage-Based Reading (UBR) → Best-Practice Inspection**
  - Typischerweise existieren Szenarien (z.B. Use Cases)
  - Priorisierung von Anwendungsfällen durch Fachleute.
  - Zielsetzung: Fehler in wichtigen Anwendungsfällen zuerst finden.
  - Typische Vorgehensweise:
    1. Auswahl des wichtigsten Anwendungsfalls
    2. Überprüfung der relevanten Teile, die diesen Anwendungsfall beschreiben
    3. Reporting von Mängeln und Fehlern
    4. Auswahl des nächsten Anwendungsfalls

# Kosten und Nutzen von Inspektion

- Kosten
  - Aufwand Inspektoren (Personenstunden); erhöht sich bei Teammeetings zusätzlich.
  - Einplanung im Projektplan.
- Nutzen
  - Einsparungen durch frühes Finden und Entfernen von Fehlern.
  - Genauere Informationen über das Produkt und den Entwicklungsprozess zur Planung der Aktivitäten und Ressourcen (PM und QM).
  - Beurteilungsgrundlage für die Produktqualität (z.B. durch Restfehlerschätzungen).
  - Besseres Produktwissen der Inspektoren.
  - Gemeinsame Sicht des Inspektionsteams auf das Produkt.
  - Inspektion als Lerninstrument für neue Teammitglieder.

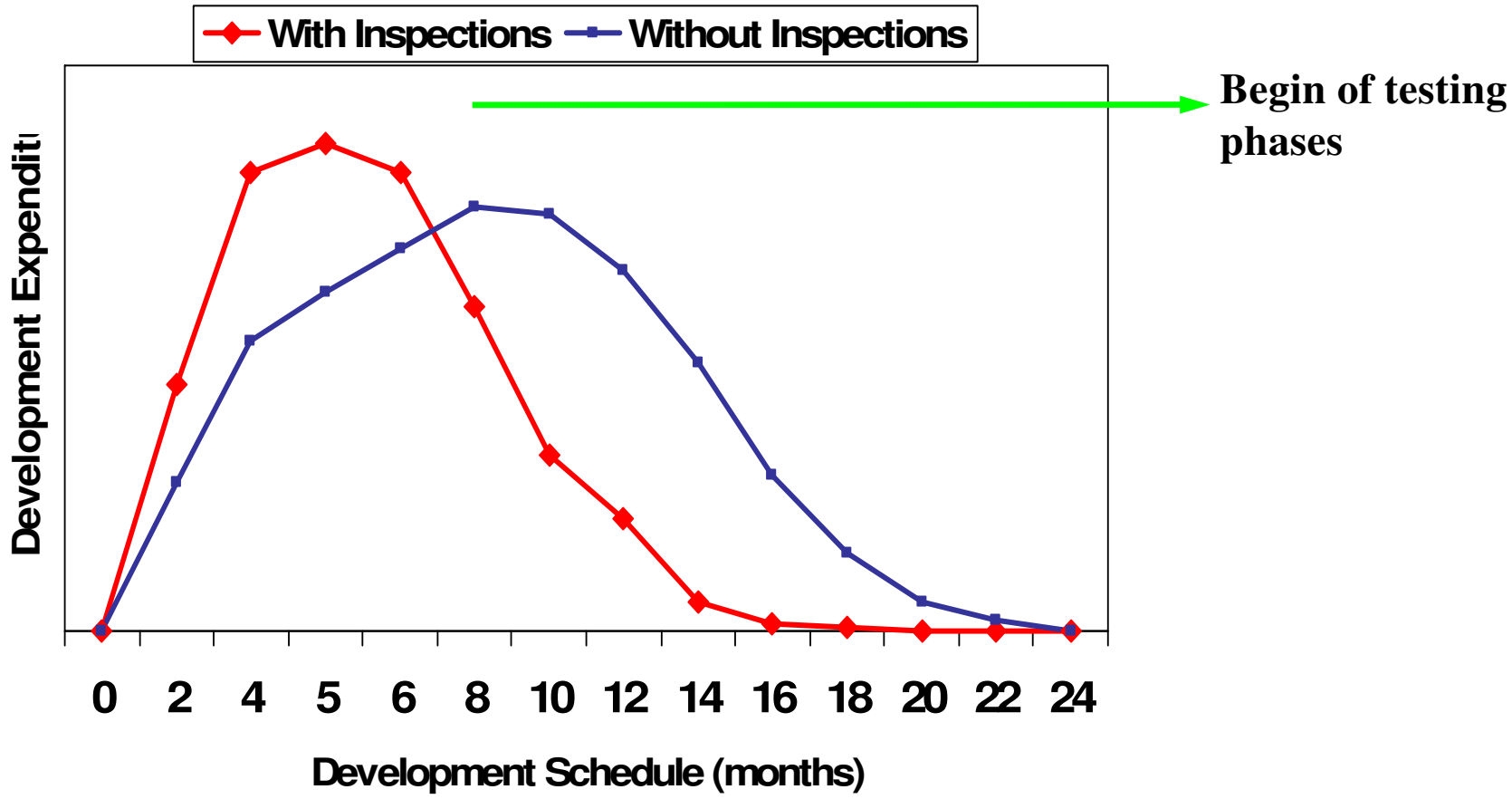
# Software Development - Effort




**Distribution of rework during development**

Wheeler D., Brykczynski B., Meeson R., "Software Inspection An Industry best practice", IEEE Computer Society Press, 1996 Figure 1-5

# Effort along Life cycle




**Software Development Spending Profiles**

Wheeler D., Brykczynski B., Meeson R., "Software Inspection An Industry best practice", IEEE Computer Society Press, 1996, Figure 1-7

# Typische Fehler im Software Engineering (SEPM LU Projekte)

- Datenmodellierung, Reports
  - Schlüsselwerte nicht eindeutig
  - Fehlende Attribute
  - Fehler bei Beziehungen, z.B. Anzahl (1:n; 1:1)
- Entwurf: Ergebniswerte von Funktionen
  - Typkonversion falsch (z.B. Integer/Real)
- Initialisierung von Variablen
  - Fehlende Zuweisung: nicht eindeutiger Wert
- Bedingungen
  - Logische Fehler; falsche Ausdrücke (und/oder)
- Schleifen
  - Anzahl Durchläufe nicht korrekt; z.B. Endlosschleifen
- Testfälle
  - Unzureichende Überdeckung (Auslassen von Use Cases, Programmteilen, GUI-Elementen, Datenzuständen)
  - Überflüssige Testfälle (gleiche Äquivalenzklasse)

# Qualität und QS in SE&PM Projekten

- Für die Umsetzung von Qualität und QS in SEPM sind 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 Use Cases)
  - 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

# Review Ansätze im Übungsteil

- Software Requirement Review
  - Sicherstellen der Anforderung
  - Anforderungen auch angemessen
  - Richtige Spezifikation der Anforderungen
  - Phase: Systemspezifikation

- Preliminary Design Review
  - Prüfen der technischen Angemessenheit mit dem momentanen Design
  - Vergleich momentanes Design mit Software-Design Beschreibung
  - Phase: Systemdesign

# System zur Klassifizierung von Anforderungen: FURPS

- *Functional*
  - Features, Fähigkeiten, Sicherheitsvorrichtungen
- *Usability*
  - Menschliche Faktoren, Hilfe, Dokumentation
- *Reliability*
  - Häufigkeit eines Ausfalls, Vorhersehbarkeit, Wiederherstellbarkeit
- *Performance*
  - Antwortzeiten, Durchsatz, Genauigkeit
- *Supportability*
  - Anpassungsfähigkeiten, Wartung, Konfigurationen

[Grady, 92]

# Anforderungen: Typen

- Funktionale Anforderungen
  - Beschreiben **was** das System tut, um Mehrwert für die Stakeholder zu liefern.
  - Haben typischerweise folgende Form:
    - Das System kann Funktion X ausführen.
- Nichtfunktionale Anforderungen
  - "Qualitätsanforderungen" (**wie „gut "**)
  - Erwünschte Charakteristika des Systems
- Constraints
  - Festgelegte Einschränkungen durch verwendete Frameworks bzw. Architekturentscheidungen
- Klassifizierungen
  - FURPS oder FURPS+
  - Zur Laufzeit des Systems messbar
    - Performanz, Sicherheit, Verfügbarkeit, Verwendbarkeit
  - Nicht zur Laufzeit messbar
    - Modifizierbarkeit, Portabilität, Wiederverwendbarkeit

# Beispiele für Funktionale und Nichtfunktionale Anforderungen

- Funktionale Anforderungen
  - Das System kann gespeicherte Rechnungen als ausdruckbare Datei exportieren.
  - Der Administrator kann Bestellungen stornieren
- Nichtfunktionale Anforderungen
  - Wenn ein Fehler eintritt, informiert das System den Benutzer und arbeitet in einem verschlechterten(degradierten) Zustand weiter.
  - Falls das System die Anzahl der Bestellungen nicht verarbeiten kann, trennt es einige bestehende Verbindungen zu Clients. Der Administrator wird informiert und das System arbeitet in einem verschlechterten(degradierten) Zustand weiter.

# Review von Anforderungen

- Reviews überprüfen die Korrektheit und Konsistenz von Artefakten an wohldefinierten Punkten des Entwicklungsprozesses.
  - Review der Anforderungen nach Änderungen gegen die Projektbeschreibung
  - Review später hergestellter Dokumente gegen die aktuell gültigen Anforderungen
  - Rückverfolgbarkeit der Anforderungen
- Anforderungen nach Kriterien einteilen
  - Funktional, nichtfunktional, testbar, verständlich
  - Erstrebenswert ist eine klare, knappe Beschreibung, welche umsetzbar ist
- Erstrebenswerte Kriterien aufzeichnen
  - Konsistenz, Vollständigkeit
  - Beispiel Review Issue:
    - "Spezifikation des Fehlerfalls ist notwendig um der Projektbeschreibung zu genügen"

# Review von Anforderungen und SE-Modellen

- 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.
- 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

# Von Anforderungen abgeleitete Modelle

- Anforderungen -> Modelle -> Datenbank, Business Logik, GUI
  - Modelle: Anwendungsfälle, Domänenmodelle
  - Datenbankmodelle: Datenmodellierung, EER & Relationen
  - Business Logik: Zustandsdiagramme, 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-First, Test-Driven
  - Tests überprüfen die Spezifikation schon vor dem detaillierten Entwurf
  - TDD: Herstellen von ablauffähigen Testfällen vor der Implementierung
  - Anforderungen müssen auch ohne Software

# Übungsteil: Beispiel Restaurant

- Bestandteile der Aufgabenstellung
  - Projektbeschreibung (Text)
  - Konkrete Anforderungen (Liste)
  - Anwendungsfälle (UML 2)
    - Anwendungsfallbeschreibung
    - Anwendungsfalldiagramm
  - **Domänenmodell (UML 2)**
  - Missionskritische Testfälle
- Nächster Schritt:
  - Review von Software Engineering Modellen (Domänenmodell) mit den Anforderungen und der Projektbeschreibung

# UML Sichten auf Softwarestrukturen (UML Klassendiagramm)

- Konzeptionelle Sicht
  - Gibt eine Übersicht über den zu untersuchenden Problembereich, z.B. Domänenmodell.
- Spezifizierende Sicht
  - Hier betrachten wir Software-Schnittstellen (keine Implementierungen, z.B. Komponentendiagramm).
- Implementierende Sicht
  - Bei dieser Sichtweise beobachten wir Klassen und legen die Grundlage für die Implementierung. Dies ist vermutlich die am häufigsten eingenommene Betrachtungsweise, z.B. Klassendiagramm.

nach Martin Fowler und Kendall Scott: „UML konzentriert“

# Überprüfung des Domänenmodells mit den Anforderungen

- Von den Anforderungen ausgehen
  - Daten in den Anforderungen werden von Domänenklassen gekapselt
  - **Operationen** + **Entities** in den Anforderungen führen zu den Domänenklassen

- Domänenklassen systematisch durchgehen
  - Ist jede Klasse notwendig, plausibel, korrekt und vollständig modelliert?
  - Passen die Elemente der Klassen zu den Anforderungen? (Abweichungen sind zu dokumentieren)
  - Sind die Assoziationen zwischen den Klassen ausreichend? – Multiplizitäten überprüfen.
  - Mögliche Ergebnisse:
    - Klasse ist unvollständig: AttributX und AttributY fehlt.

# Review des Domänenmodells mit den Anforderungen 1/3

- Beispiel Anforderung:

  27. Neue Speisen werden vom Koch eingefügt, dabei sind auch das *Rezept* der Speise sowie die Attribute *Bezeichnung* und *Verkaufspreis* einzugeben.
  - Eine Speise kann mit verschiedenen Rezepten zubereitet werden.
  - Jede Speise hat ein „aktuelles Rezept“, mit dem die Speise bei Bestellung zubereitet wird.

- Zugehöriges Domänenmodell (unvollständig):

# Review des Domänenmodells mit den Anforderungen 2/3

- Domänenmodell:


- Review:

| Review des Domänenmodells mit den Anforderungen | |
|---|---|
| **Klassenname** | **Issues** |
| Speise | - Assoziation falsch: Speise muss mindestens ein Rezept haben!<br>- Attribute Name, Bezeichnung fehlen!<br>- Neues Attribut: "main_recipe"<br>-... |

# Review des Domänenmodells mit den Anforderungen 3/3

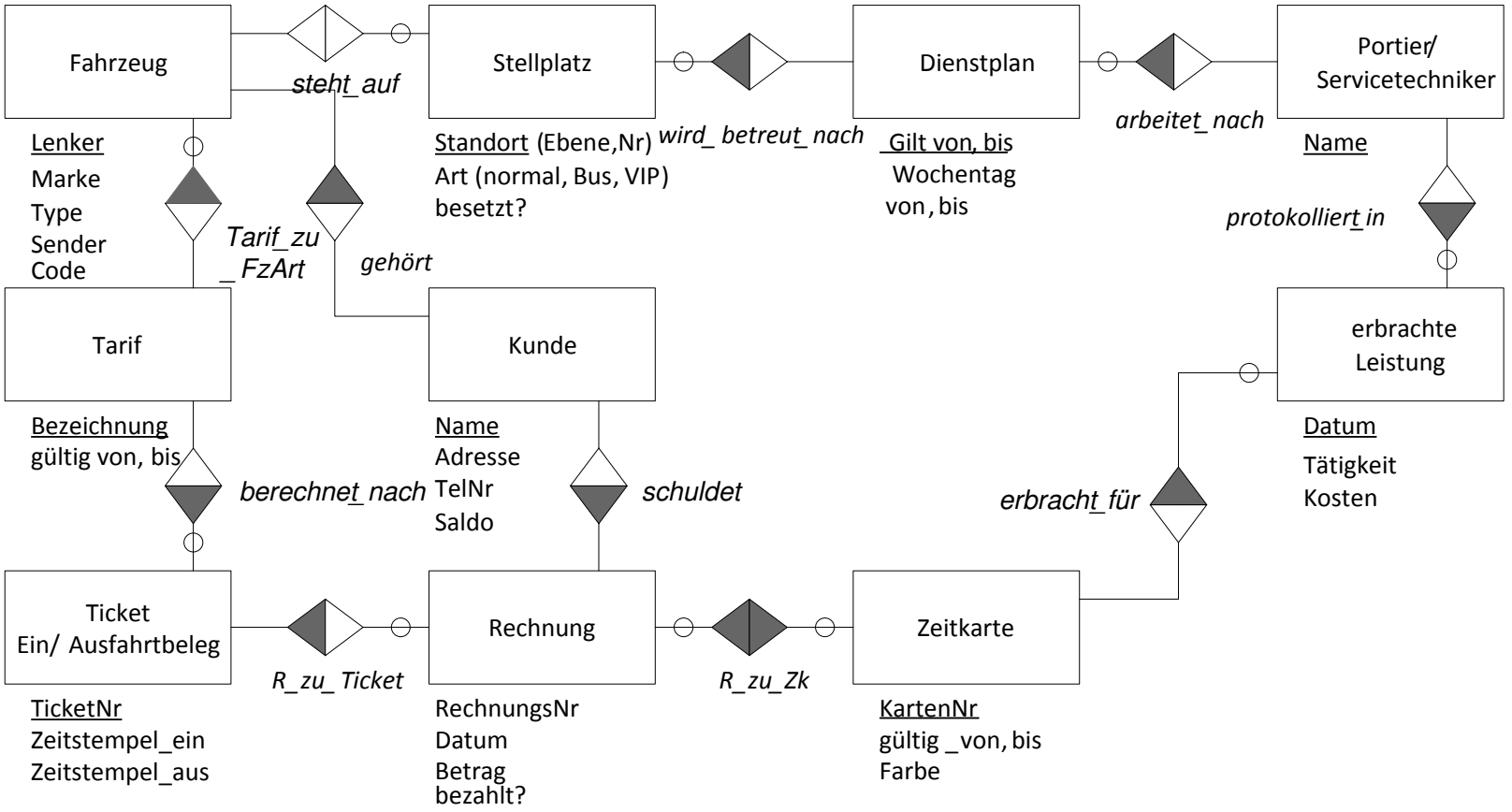
- Neues Domänenmodell (unvollständig mit weiteren Issues):

# Überprüfung des Domänenmodells in Bezug auf den Anforderungstext

- Vom Anforderungstext ausgehen
  - Zu speichernde Daten identifizieren und markieren, etwa unterstreichen.
  - Daten in den Anforderungen werden von Domänenklassen gekapselt
  - **Operationen** + **Entities** in den Anforderungen führen zu den Domänenklassen
  - Markieren Sie im Text jene Daten, die durch eine Klasse abgedeckt werden.

- Domänenklassen systematisch durchgehen
  - Ist jede Klasse relevant, korrekt und vollständig modelliert?
  - Passen die Elemente der Klassen zu den Anforderungen? (Abweichungen sind zu dokumentieren)
  - Sind die Assoziationen zwischen den Klassen richtig? – Multiplizitäten überprüfen.
  - Mögliche Ergebnisse:
    - Klasse ist unvollständig: AttributX und AttributY fehlt.
  - Markieren Sie jede überprüfte Klasse, sodass Sie jederzeit feststellen können
    - Welche Klassen sind noch zu überprüfen?
    - Gibt es im Text Daten, die gar nicht gespeichert werden können?

# Entity Relationship Diagramm mit Fehlern



# Überprüfung eines Diagramms in Bezug auf ein übergeordnetes Diagramm

- Vom übergeordneten Diagramm ausgehen
  - Relevantes Scope markieren.
  - Gehen Sie das Scope systematisch durch.
  - Markieren Sie alle Aspekte, die durch das untergeordnete Diagramm abgedeckt werden.
- Das untergeordnete Diagramm systematisch durchgehen
  - Sind die Gemeinsamkeiten mit dem übergeordneten Diagramm korrekt und vollständig modelliert?
  - Markieren Sie jedes überprüfte Element, sodass Sie jederzeit feststellen können
    - Welche Elemente sind noch zu überprüfen?
    - Gibt es im übergeordneten Diagramm Elemente, die nicht adressiert 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

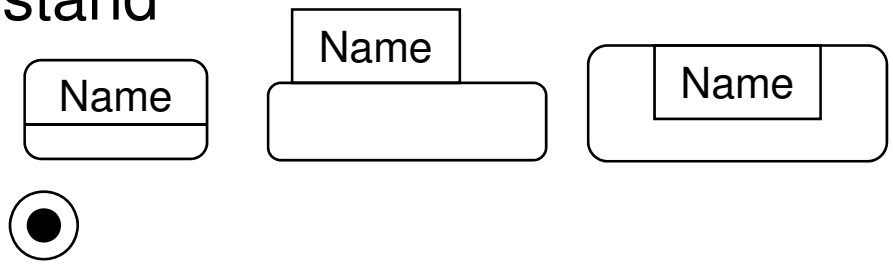
- Modelliert werden
  - Die Zustände, in denen sich die Objekte einer Klasse befinden können
  - Die möglichen Zustandsübergänge (Transitionen) von einem Zustand zum anderen
  - Die Ereignisse, die Transitionen auslösen
  - Aktivitäten, die in Zuständen bzw. im Zuge von Transitionen ausgeführt werden

Aus: OO Modellierung, BIG, 2006

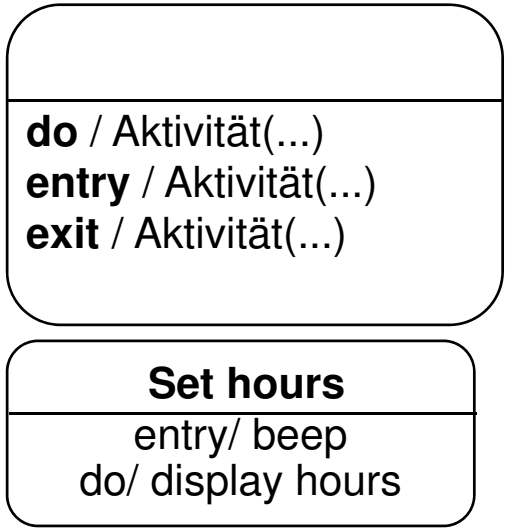
# Zustandsdiagramm

## 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

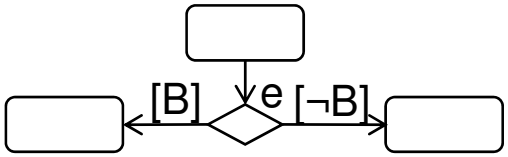
# Zustandsdiagramm

## 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()

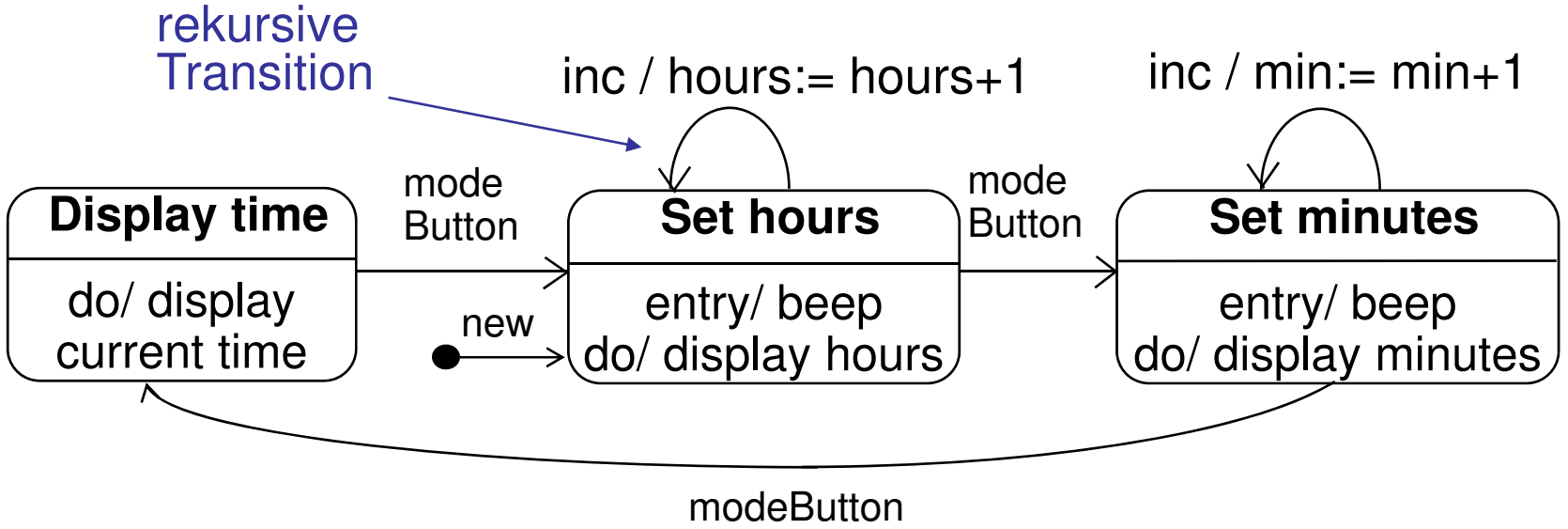



Aus: OO Modellierung, BIG, 2006

# Zustandsdiagramm

## Basiskonzepte – Beispiel: Klasse DigitalWatch

| DigitalWatch |
| --- |
| min : Integer = 0<br>hours : Integer = 0 |
| modeButton()<br>inc() |

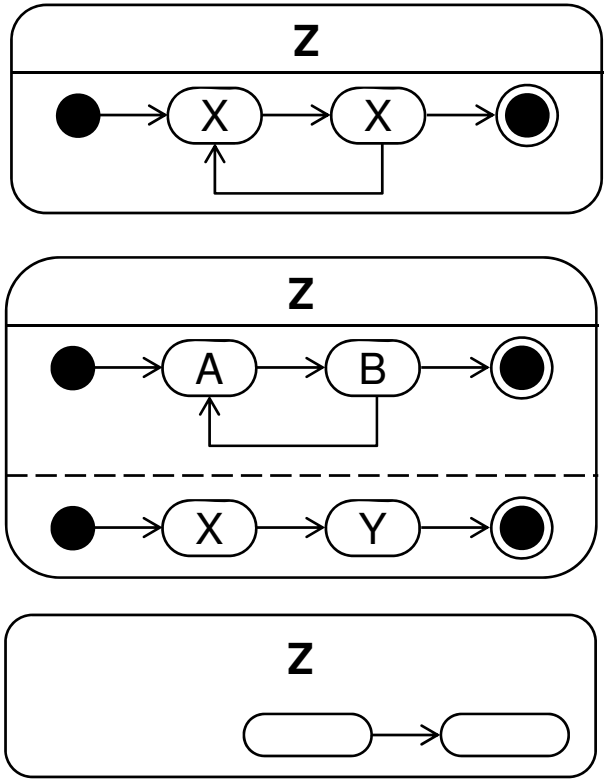



Aus: OO Modellierung, BIG, 2006

# Zustandsdiagramm

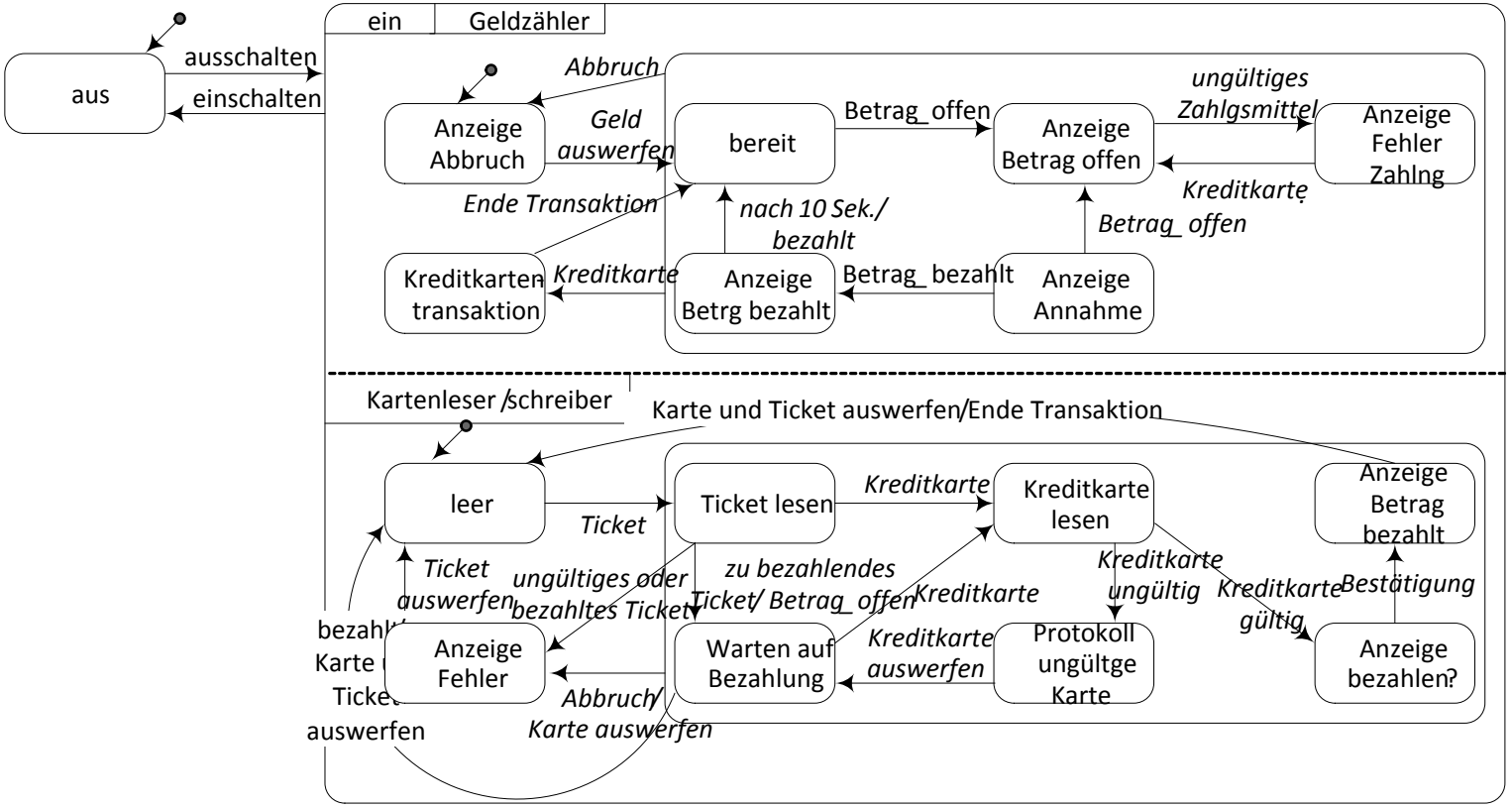
## Strukturierung – ODER- vs. UND-Verfeinerung

- 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

# Spezifikation Verhalten mit Fehlern



# Zusammenfassung

- Analytische Methoden dienen der Verbesserung der Produkt- und Prozessqualität sowie der Einschätzung der Produkt- Prozess- und Projektqualität.

- Die Durchführung von Reviews und Inspektionen folgt einem definierten Prozess.
- Rollen für Reviews und Inspektionen erfüllen definierte Aufgaben.
- Lesetechniken unterstützen die Reviewer und Inspektoren bei der Fehlerfindung.

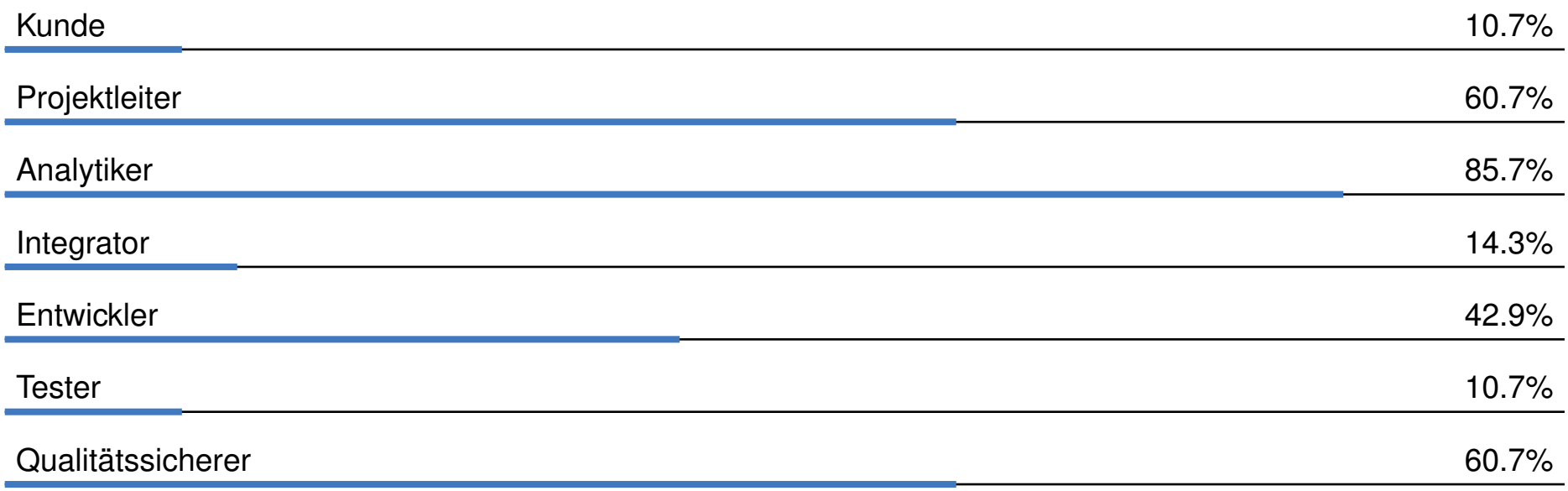
- Im Zentrum dieser analytischen Maßnahmen steht das zu überprüfende Produkt, nicht der Autor!

# Referenzen


- Biffl, Stefan: „*Software Inspection Techniques to support Project and Quality Management*", Shaker Verlag, 2001, ISBN: 3-8265-8512-7
- Biffl Stefan, Winkler Dietmar, Frast Denis: „Qualitätssicherung, Qualitätsmanagement und Testen in der Softwareentwicklung", Skriptum zur Lehrveranstaltung, 2004. http://qse.ifs.tuwien.ac.at/courses/skriptum/script.htm
- Booch Grady, James Rumbaugh, Ivar Jacobson; The UML User Guide; Addison Wesley, 2000, ISBN 0-201-57168-4
- Laitenberger, Oliver: "Cost-Effective Detection of Software Defects Through Perspective-based Inspections"; PhD Theses in Experimental Software Engineering Vol. 1, 2000
- Larman Craig; Applying UML and Patterns, An Introduction to Object Oriented Analysis and Design and the Unified Process, Prentice Hall, 1997, ISBN 0137488807
- Schatten, Biffl, Winkler, et al. Best Practice Software-Engineering - Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen; Spektrum Akademischer Verlag, 2010.
- Sommerville Ian: „Software Engineering", 9th Edition, Addison-Wesley, 2010.
- Thaller, Georg Erwin; Software Qualität; der Weg zu Spitzenleistungen in der Software-Entwicklung; VDE-Verlag, 2000; ISBN 3-8007-2497-4
- Wallmüller Ernest: „Software Quality Engineering", 3. Auflage, Hanser, 2011, ISBN 978-3-446-40405-2.

# Wie ist das Wetter heute?

# Wer soll an der Review eines "Analysemodells" teilnehmen?



# Welche Artefakte wären besonders sinnvoll zu reviewen?

Vermeiden Sie doppelte Antworten.

Sebastian Luzian — Votes: + 16 / - 0

Projektauftrag

Bernadette Maria Obermair — Votes: + 16 / - 0

Systemarchitektur

Martin Kamleithner — Votes: + 14 / - 1

Domänenmodell

Michael Pürmayr — Votes: + 12 / - 0

Schnittstellen

David Stemberger — Votes: + 7 / - 0

Software Requirement Specifications

Patrick Fritz Johann Raser — Votes: + 7 / - 0

Tests/Testplan

Anton Hößl — Votes: + 6 / - 0

Use-Cases

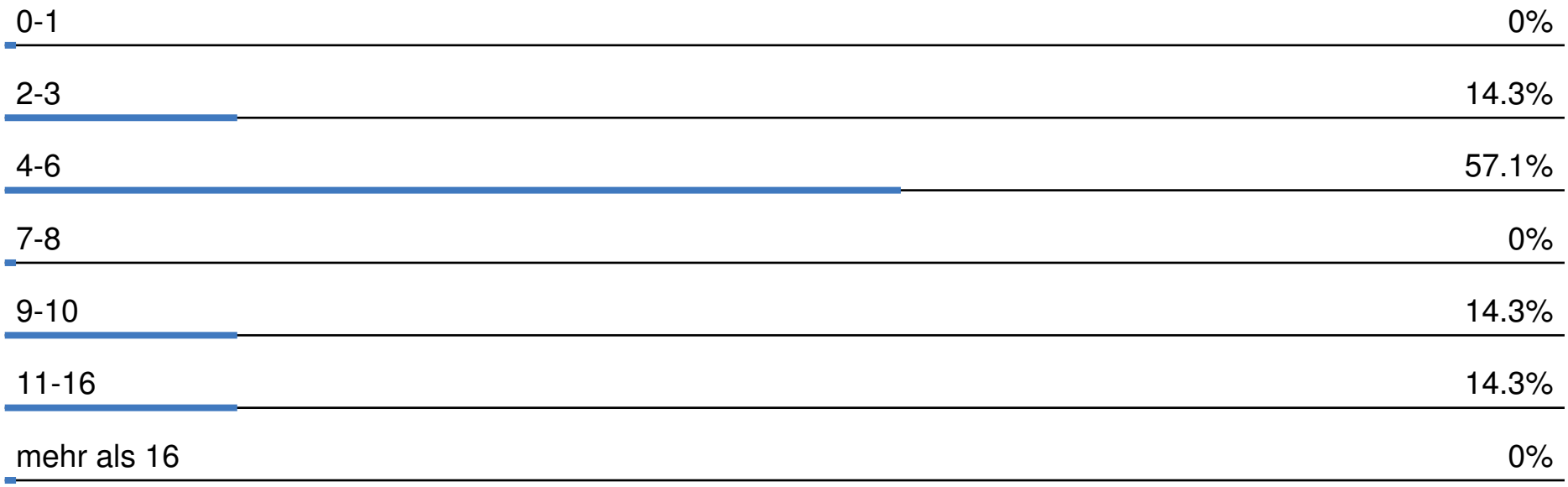
Robert Fischer — Votes: + 5 / - 0

Quelltext (Wartbarkeit,...)

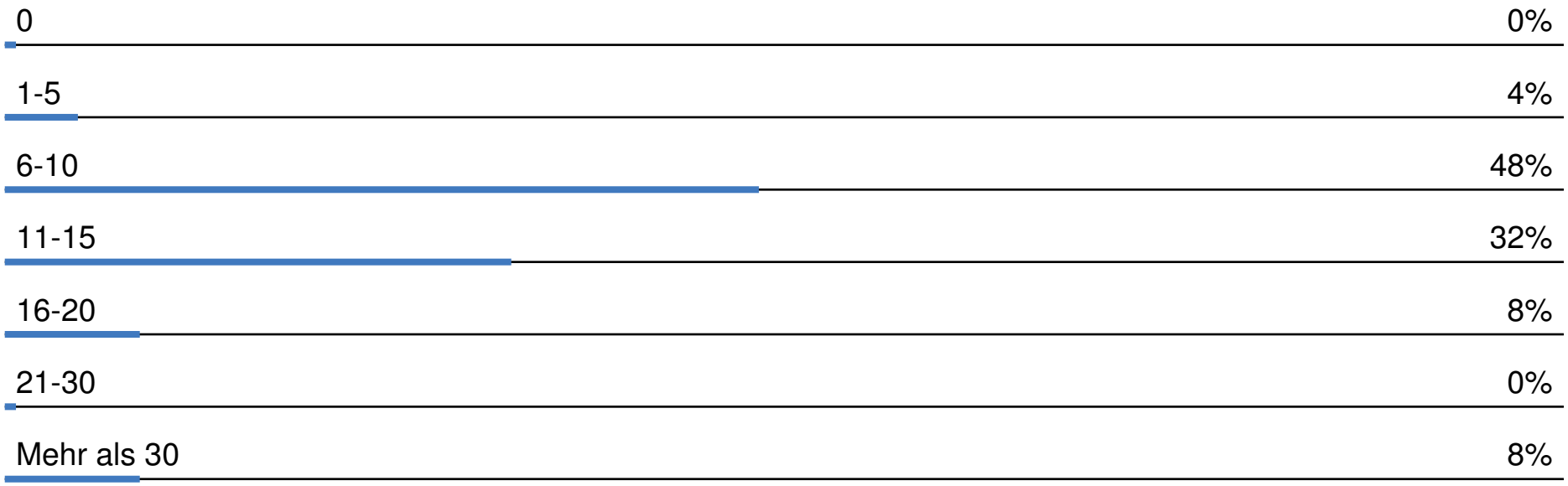
Julian Waibel — Votes: + 9 / - 4

Datenmodell

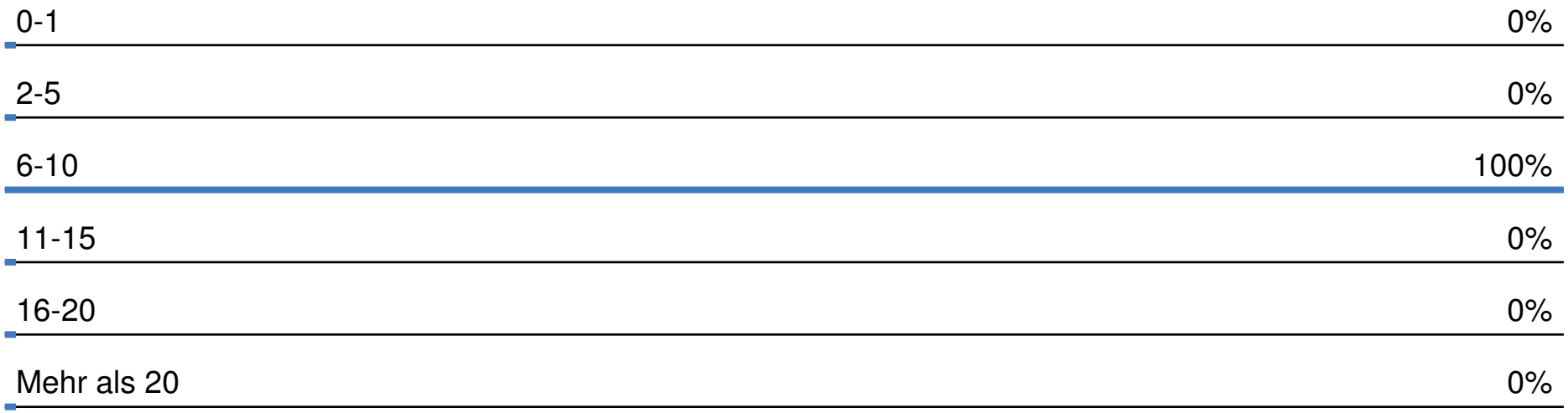
# Wie viele Reviews würden Sie einplanen?



# Wie viele Fehler schätzen Sie sind im ER-Diagramm "Parkgarage" enthalten (im Vergleich zum Anforderungstext)?



# Wie viele Fehler haben Sie in der Gruppe gefunden?



# Welche Fehler haben Sie im ER-Diagramm "Parkgarage" beobachtet?

Vermeiden Sie doppelte Antworten. Bewerten Sie Fehler, die Sie auch gefunden haben, positiv. Bewerten Sie Fehlermeldungen, die Sie für ungültig halten, negativ.

Robert Fischer — Votes: + 15 / - 0

Portier/Serviceleistung hat keine Qualifikation

Matthias Deimel — Votes: + 10 / - 0

Rechnung sollte einen Primary Key haben, hat aber keinen

Anton Hößl — Votes: + 8 / - 0

Schlüssel Name Kunde ist nicht eindeutig

Julian Waibel — Votes: + 6 / - 0

welche Zeitkarte zu welchem Kunden gehört ist nicht modelliert

David Schröder — Votes: + 5 / - 0

Tarif hängt von Dauer des Besuchs etc ab, im ER jedoch vom Fahrzeugtyp

Robert Fischer — Votes: + 4 / - 0

Portier/Serviceleistung hat Name als Primärschlüssel (Eindeutigkeit nicht sichergestellt)

Markus Goedl — Votes: + 4 / - 0

Stellplatz hat 0 oder 1 Fahrzeug und nicht genau 1 Fahrzeug. Dadurch entfällt Attribut "besetzt".

Markus Schaidinger — Votes: + 5 / - 1

VIP Status des Kunden nicht speicherbar

Stephan Goldschmidt — Votes: + 6 / - 2

Schlüssel bei Fahrzeug ist Kennzeichen

David Schröder — Votes: + 4 / - 0

Entität Bereich fehlt

Julian Waibel — Votes: + 3 / - 0

Anfragen und Rückmeldung der Kunden für Marketing nicht modelliert

Martin Kamleithner — Votes: + 3 / - 0

Beziehung Kunde/Zeitkarte fehlt

Lukas Pfeifhofer — Votes: + 3 / - 0

Rechnung steht n zu n mit Zeitkarten. Eigentlich sollte die Rechnung in Relation zur Leistung stehen

Michal Palczynski — Votes: + 2 / - 0

Entität Journal, dieim scenario vorhanden ist, fehlt im EER

Anton Hößl — Votes: + 2 / - 0

Fahrzeug Schlüssel nicht eindeutig von Kunde (name nicht eindeutig) möglich Kennzeichen verwenden

Marco Ramharter — Votes: + 2 / - 0

Fahrzeug hat kein Attribut Farbe

Anton Hößl — Votes: + 2 / - 0

Keine Entität Zeitkartenkunde/Dauerkunde aus Szenario 3/4

Markus Schaidinger — Votes: + 1 / - 0

Stellplatz - Dienstplan Relation nicht korrekt

Michael Pürmayr — Votes: + 1 / - 0

Bereiche nicht abgebildet
Kundentypen n abgebildet
Auto fehlt Attribut Farbe
Portier fehlt Qualität

David Tichy — Votes: + 1 / - 0

Kundentypen werden gewünscht im ER nicht vorhanden

Sebastian Luzian — Votes: + 1 / - 1

<Ticket> sollte <Ticket.gültig> enthalten

Christoph Michael Weber — Votes: + 1 / - 1

Sender mit Code ist Fahrzeug statt Kunden zugeordnet

Sebastian Luzian — Votes: + 2 / - 2

<Rechnung.Nettobetrag> und <Rechnung.Umsatzsteuer fehlt>. Für 1 Ticket nur 1 Rechnung!

Christoph Michael Weber — Votes: + 1 / - 3

Portier/Servicetechniker hat kein Attribut Qualifikation

Oleksandr Sribnyi — Votes: + 4 / - 9

Attribut <Stellplatz.Art> ist ein Tripel, sollte aber nur einen bestimmten Wert beinhalten

# Welche Fehler haben Sie im State Chart "Parkgarage/Ticketautomat" beobachtet?

Martin Kamleithner — Votes: + 9 / - 0

Aus Zustand "Anzeige Fehler Zahlung" solle man automatisch nach 5 Sekunden kommen.

Markus Goedl — Votes: + 9 / - 0

Der Anfangszustand des Geldzählers sollte "bereit" und nicht "Anzeige Abbruch" sein.

David Tichy — Votes: + 5 / - 0

Einschalten und ausschalten sind vertauscht

Michal Palczynski — Votes: + 3 / - 0

Anzeige Fehler ist kein geeignete Meldung beim Abbruch. Es sollte sich Anzeige Abbruch geben

Tobias Kain — Votes: + 2 / - 0

ein Abbruch während der Kreditkartentransaktion ist nicht möglich

Judith Pitzer — Votes: + 3 / - 1

wenn ein gültiges Ticket gelesen wird, sollte der offene Betrag angezeigt werden

Martin Kamleithner — Votes: + 2 / - 0

Kante "Anzeige Betrag offen" -> "Anzeige Annahme" über Ereignis "bezahlt" fehlt

Michael Pürmayr — Votes: + 2 / - 0

Vor Zustand "Warten auf Bezahlung" fehlt Zustand "Anzeige offener Betrag"

Michael Pürmayr — Votes: + 2 / - 0

Nach "Anzeige Betrag bezahlt" fehlt ein Ereignis "Wechselgeld auszahlen"

Michael Pürmayr — Votes: + 1 / - 0

Nach Anzeige Fehler Zahlung sollte "nach 5 Sek/..." stehen

Sebastian Luzian — Votes: + 1 / - 0

Während einer Transaktiron kann nicht abgebrochenw erden

Patrick Fritz Johann Raser — Votes: + 1 / - 0

Wenn Barzahlung ist der Kartenleser weiterhin im Zustand "Warten auf Bezahlung", in dem Fall sollte aber das Ticket gestempelt und ausgeworfen werden

Matthias Deimel — Votes: + 1 / - 0

Zustand Anzeige Annahme, kann nicht betreten werden, sollte Kante Betrag_offen sollte gedreht und durch Betreg_bezahlt ersetzt werden

Anton Hößl — Votes: + 1 / - 0

Szenario 3. Initialzustand bereit existiert nicht (default ist Zustand leer)

Michael Pürmayr — Votes: + 1 / - 0

Nach Zustand "Anzeige bezahlt" fehlt das Ereignis "Ausfahrtszeitstempel auf Ticket schreiben"

Christina Greil — Votes: + 1 / - 0

Bereits eingeworfenes Geld sollte ausgeworfen werden, wenn eine Kreditkarte eingelesen wird, ist aber nicht so.

Anton Hößl — Votes: + 2 / - 1

Szenario 3. Keine Routine zum Barzahlen

Michael Pürmayr — Votes: + 1 / - 0

Zustand "Kreditkarte lesen" sollte nur von "Warten auf Bezahlung" erreicht werden

Markus Goedl — Votes: + 2 / - 1

Den Zustand "aus" sollte man mit "einschalten" verlassen und nicht mit "ausschalten".

Robert Fischer — Votes: + 1 / - 0

Der Kartenschreiber schreibt nicht auf den Stempel (Szenario 1). Dieser Zustand existiert nicht.

David Stemberger Votes: + 1 / - 2

Abbruch führt zu Anzeige Fehler

Christoph Michael Weber Votes: + 1 / - 4

Nach State Anzeige Betrag bezahlt sollte ein passender Ausfahrtszeitstempel auf das Ticket geschrieben werden

Michal Palczynski Votes: + 3 / - 12

Beim Klicken auf Abbruch wird der Ticket laut dem chart nicht ausgeworfen