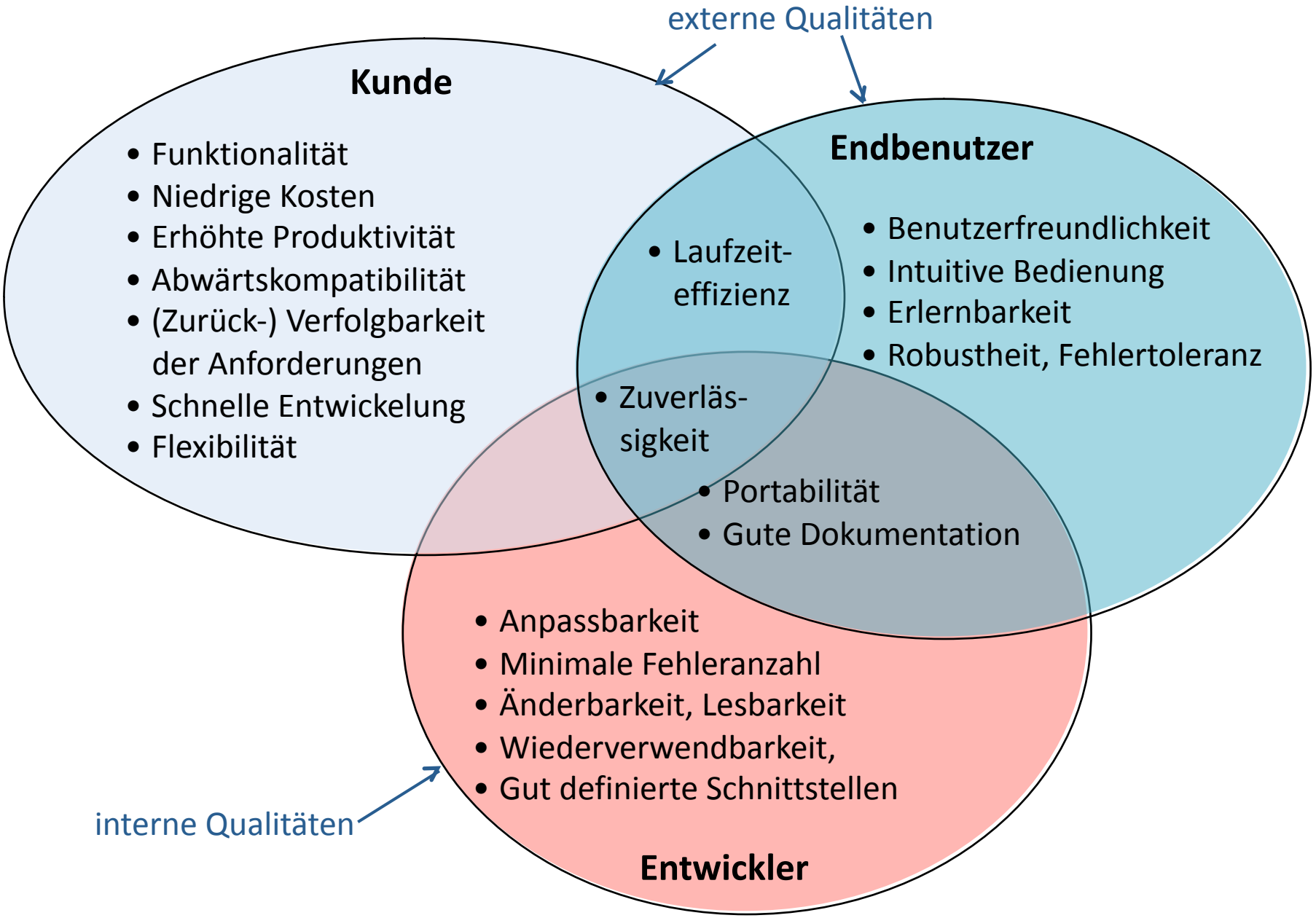
Workflows:
Systementwurf und Objektentwurf

Tutorium 5

11. März 2009

Systementwurf

Entwurfsziele



Dekomposition in Subsysteme

Dienste

- Dienst: Menge von Operationen mit gemeinsamem Zweck

- **Dienstspezifikation**: vollständig typisierte Menge von Operationen

Dekomposition in Subsysteme

Subsystem-Schnittstelle

Ist eine Menge vollständig typisierter, zusammengehörender Operationen

- Fasst einen vom Subsystem angebotenen Dienst zusammen.
- Spezifiziert Interaktion und Informationsfluss von/zu den Grenzen des Subsystems, aber nicht innerhalb des Subsystems
- Sollte wohldefiniert und schlank sein

Wird durch ein Subsystem-Schnittstellen-Objekt repräsentiert.

Dekomposition in Subsysteme

Subsystem

stark kohärente Menge von Klassen, Assoziationen, Operationen, Events und Nebenbedingungen die einen Dienst realisieren

Dekomposition in Subsysteme

Kopplung und Kohärenz

Ziel: Wartbarkeit durch Reduzierung von Abhängigkeiten

Kriterien: Kopplung und Kohärenz

- Subsysteme sollten maximale Kohäsion und minimale Kopplung haben
- Die meisten Interaktionen sollten innerhalb einzelner Subsysteme ablaufen, nicht über die Subsystemgrenzen hinweg.

Namensräume versus Subsysteme

- Packages sind nur Namensräume, keine Kapselungseinheiten
- Subsysteme werden als Komponenten (s. nächster Abschnitt) realisiert
  - Sie haben klar definierte Kapselungsgrenzen (Schnittstellen)
  - Sie begrenzen somit die möglichen Abhängigkeiten zwischen Subsystemen (da nur über die Schnittstellen zugegriffen werden kann)

Software-Komponenten

Definition von Szyperski

„Eine Softwarekomponente ist eine Kompositionseinheit mit vertraglich spezifizierten Schnittstellen und nur expliziten Kontextabhängigkeiten."

„Eine Softwarekomponente kann unabhängig eingesetzt werden und wird von Dritten zusammengesetzt."

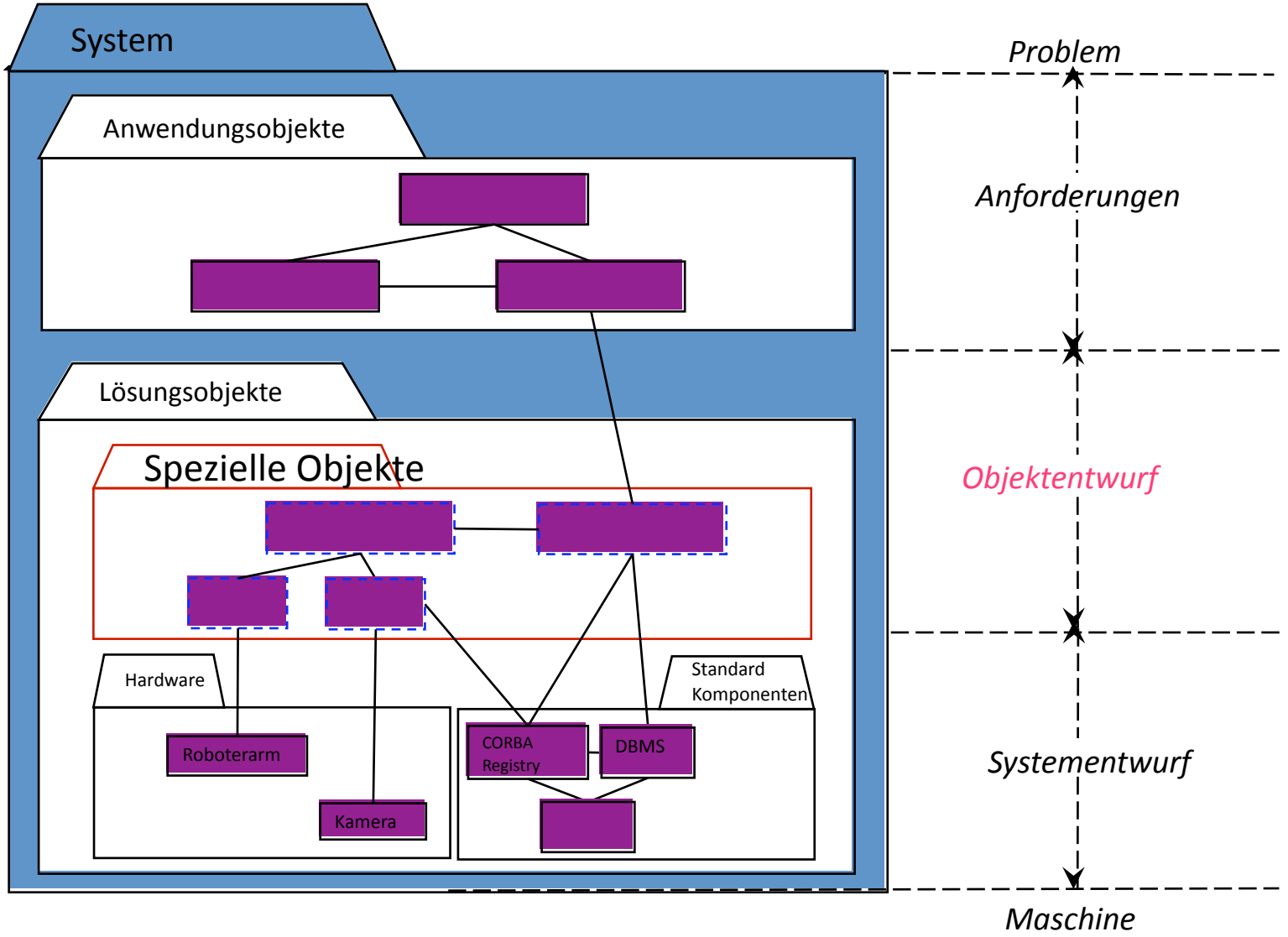
Komponentenbasierte Softwareentwicklung

Charakteristika

- Strikte Trennung zwischen Schnittstelle-spezifikation und Implementation
- Verfügbarkeit als Binärcode
- Plattformunabhängigkeit
- Ortstransparenz
- Wohldefinierter Zweck, der mehr als ein einzelnes Objekt umfasst
- Wiederverwendbarkeit
- Kontextfreiheit
- Portabilität und Sprachunabhängigkeit
- Reflektive Fähigkeiten
- Plug & Play
- Konfiguration
- Zuverlässigkeit
- Eignung für Integration / Komposition

Objektentwurf:

Die Lücke schließen



Schnittstellenspezifikation

Hilfsmittel

- Identifikation: **CRC-Karten**
- Festlegung: Signatur-Spezifikation
- Verfeinerung: **Design by Contract**
- Verfeinerung: Interaktions-Spezifikation durch **Behaviour Protocols**

} Verhaltens-Spezifikation

Design by Contract

Unterstützung in Java

- Anwendung
  - Man kann Assertions an beliebigen Stellen des Quellcodes verwenden um logische Ausdrücke zu überprüfen, von denen man annimmt, dass sie stets *wahr* sind.
    - Liefert ein solcher Ausdruck `false` zurück, wird ein **`AssertionError`** ausgelöst und die Ausführung des Programms stoppt.
    - Die ausgelösten Fehler sind vom Typ `Error` und nicht vom Typ `Exception`
    - Sie müssen daher nicht abgefangen werden!
- Syntax

```
assert boolean-expression;
assert (c != null);
```

oder:

```
assert boolean-expression : String;
assert (c != null) : "Customer is null";
```