Florian Deißenböck

Erweiterbare Architekturen

23. September 2005

Architektenworkshop

Gliederung

Hintergrund

Persönliches, Software-Qualität, Wartbarkeit

ConQATs Architektur

Entwurf, Prototypen, Konfiguration

Architektur-Evolution

Chancen, Risiken, Anomalien, Konsolidierung

Hintergrund

Persönliches

1998–2003	Studium der Informatik an der TU München
2001	Studium am AIT, Bangkok, Thailand
2003	Diplom (Diplomarbeit Prof. Broy, G. Beneken)
seit 2004	wiss. Mitarbeiter am Lehrstuhl Prof. Broy im Kompetenzzentrum Software-Wartung (CCSM)

Erweiterbare Architekturen

- DA: Architektur für erweiterbare Quelltext-Generatoren
- CCSM-Tool-Suite inkl. *CloneDetective*
- Eclipse
- ConQAT

Software-Qualität

Q-Modell »Wartbarkeit«

- Beschreibung der Einflussfaktoren
- Erklärung der Zusammenhänge
- Anleitung zur Bewertung
- ▶ Einflussfaktoren sind zahlreich und vielfältig

Q-Prozess

- »Erhaltung ist einfacher als Reparatur«
- zeitnahes/kontinuierliches Qualitäts-Controlling
- Erhöhung der Disziplin
- ▶ höhere Produktqualität und geringere Wartungsaufwände

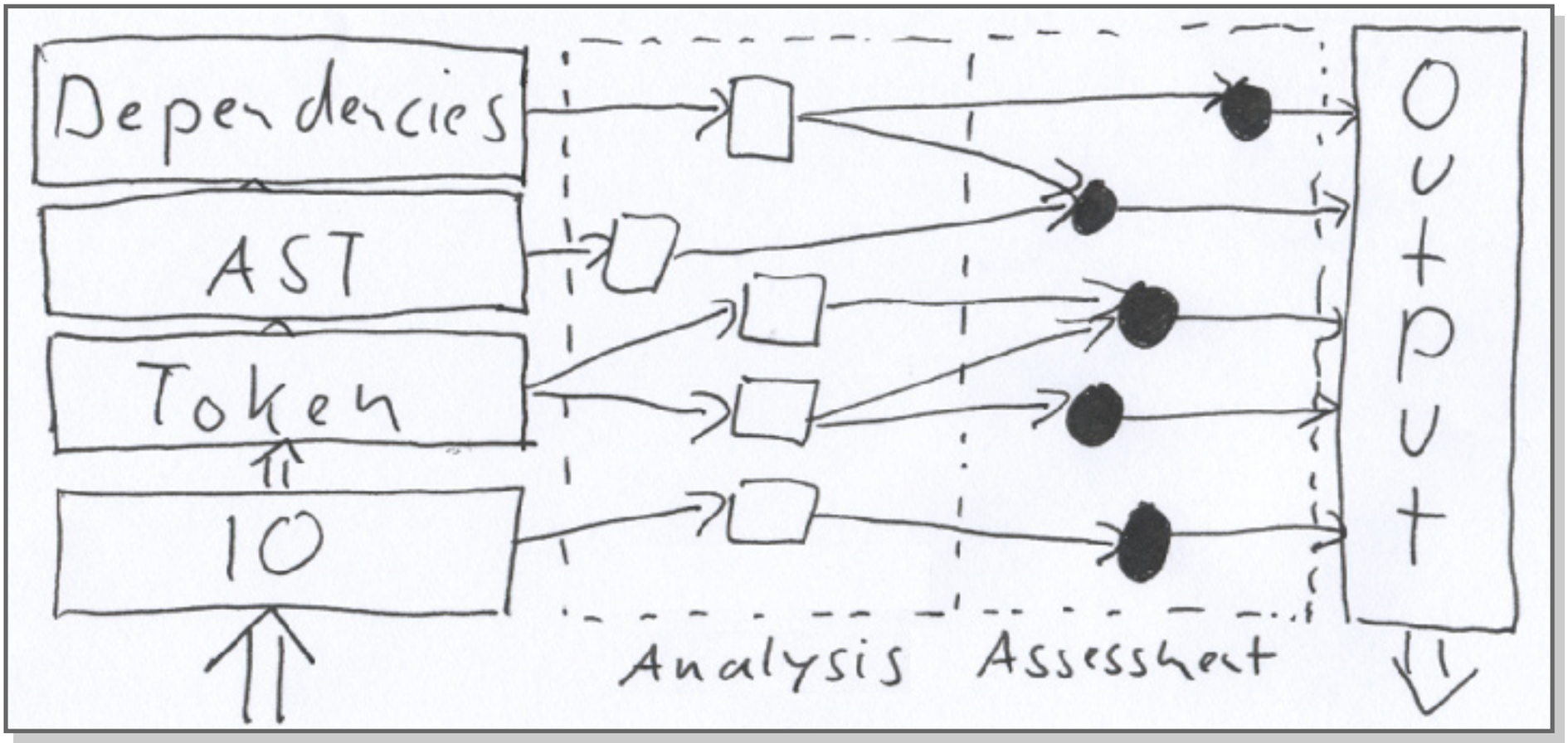
Tool-Support

- Qualitäts-Controlling ist teuer
- Automatisierung notwendig
- Entwicklung von ConQAT

Anforderungen

- Integrative Darstellung
- Batch-Betrieb
- Informations-Verdichtung
- Vielfältigkeit
- Erweiterbarkeit
- Flexibilität
- Skalierbarkeit

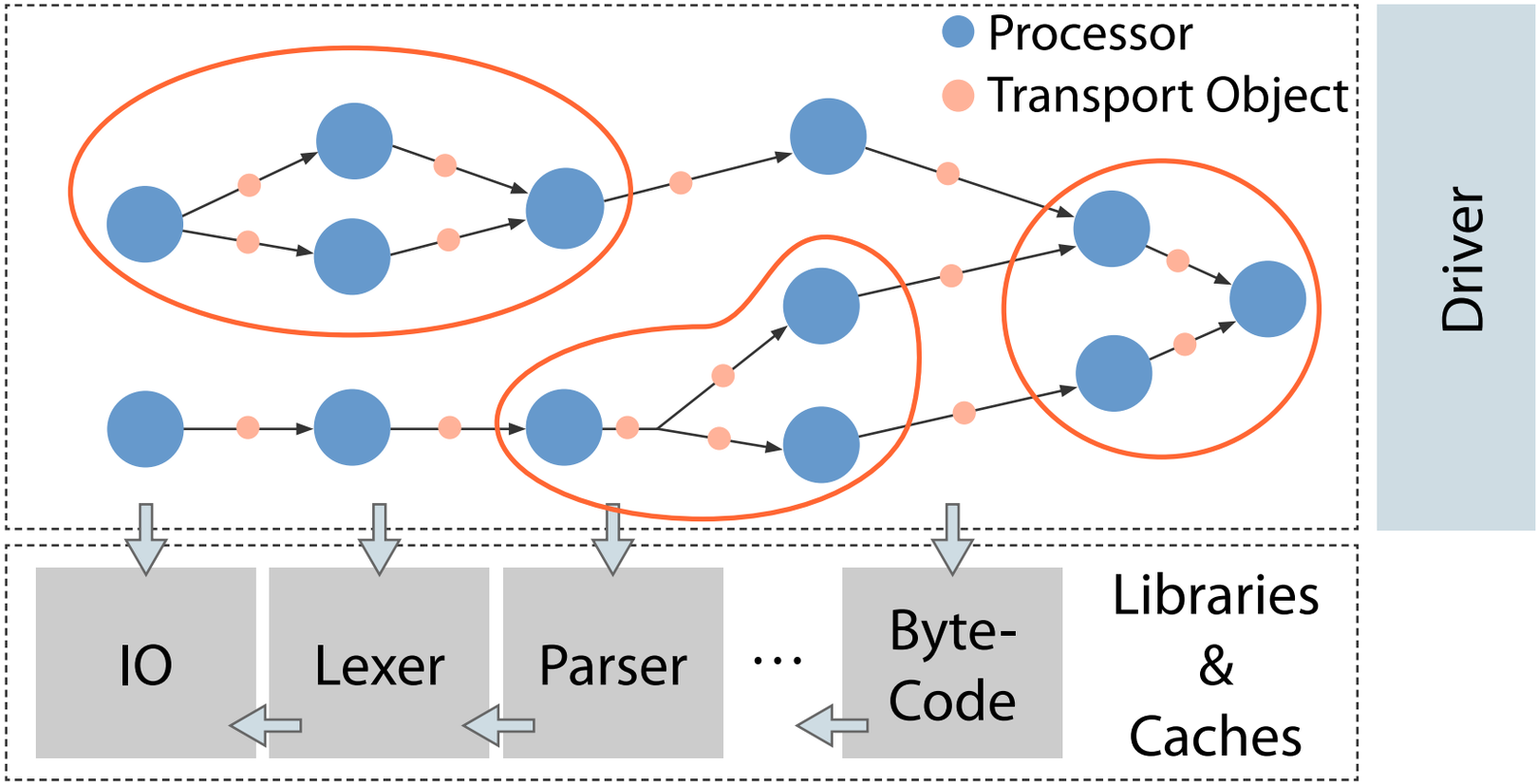
Architektur-Entwurf



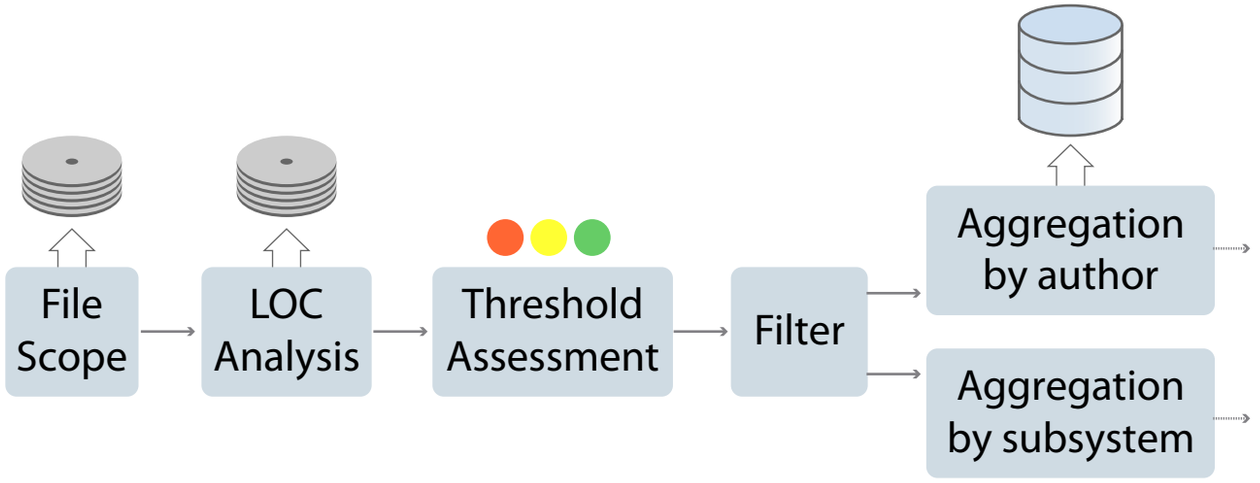
Prototypen

- Entwicklung verschiedener Prototypen
- Spektrum
  - Flexibilität vs Mächtigkeit
  - Komposition
  - Infrastruktur
- keine »ideale« Stelle auf den Spektrum
- Ausbruch aus dem Spektrum

Architektur



Beispiele



Scoping

- Dateisystem
- SCM
- Unit-Tests
- UML-Modelle

Analyse

- Metriken
- Toter Code
- PMD-Adapter
- Dokumentation
- Review-Kommentare

Ausgabe

- HTML
- Text

Konfiguration

```
<processor id="source" class="FileSystemScope">
        <root  dir="src" />
</processor>
```

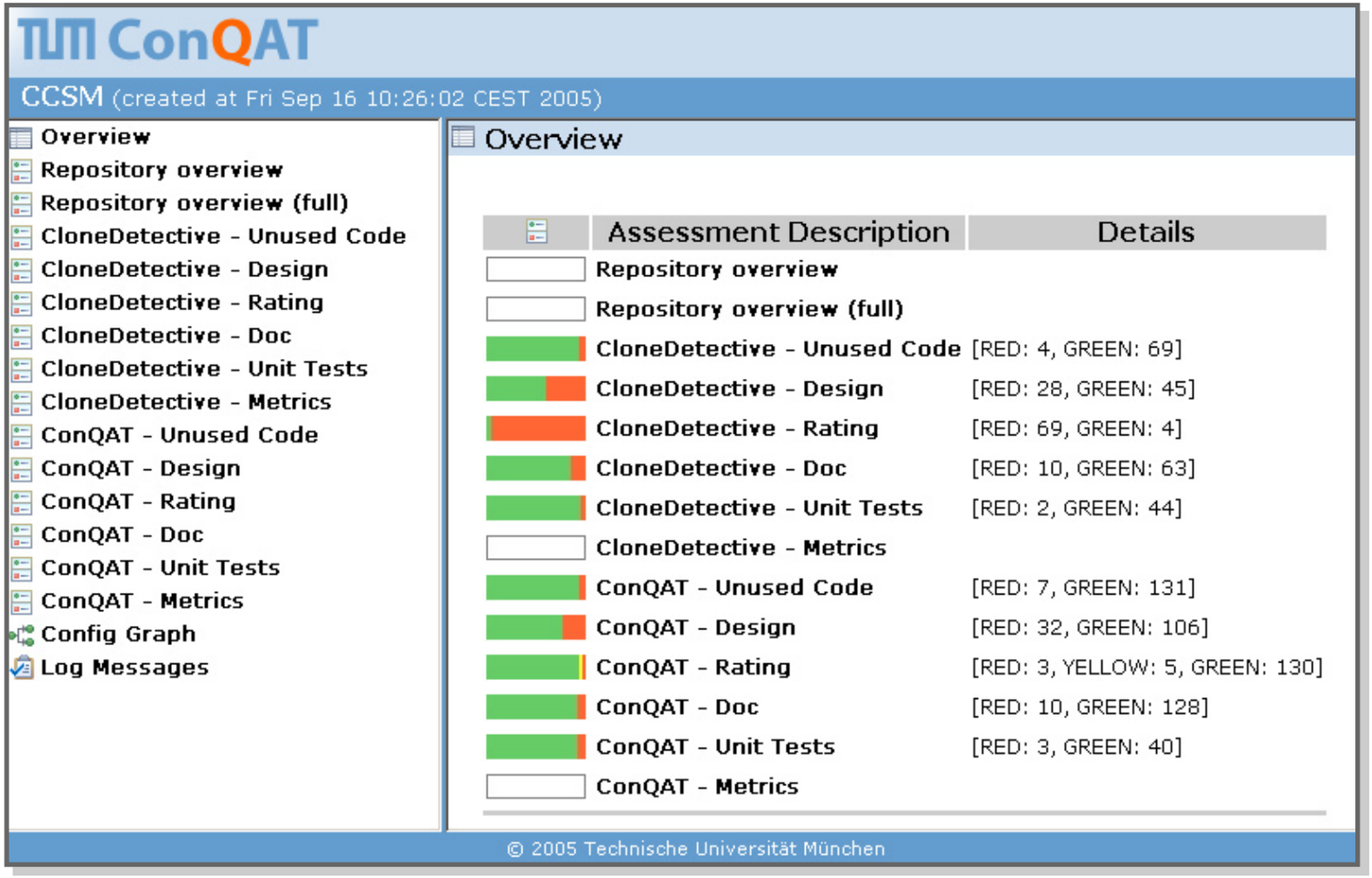
```
@AConfigProcessor(description = "File systems scope")
public class FileSystemScope implements IProcessor {

    @AConfigElement(name = "root")
    public void setRootDirectory(
            @AConfigAttribute(name = "dir")  String rootDir) {
        ...
    }

    public IFileSystemElement proces

        ....
    }
}
```

Load-time Type Checking

```
<processor id="loc-analysis" class="LOCAnalyzer">
    <input ref="@source" />
</processor>
```

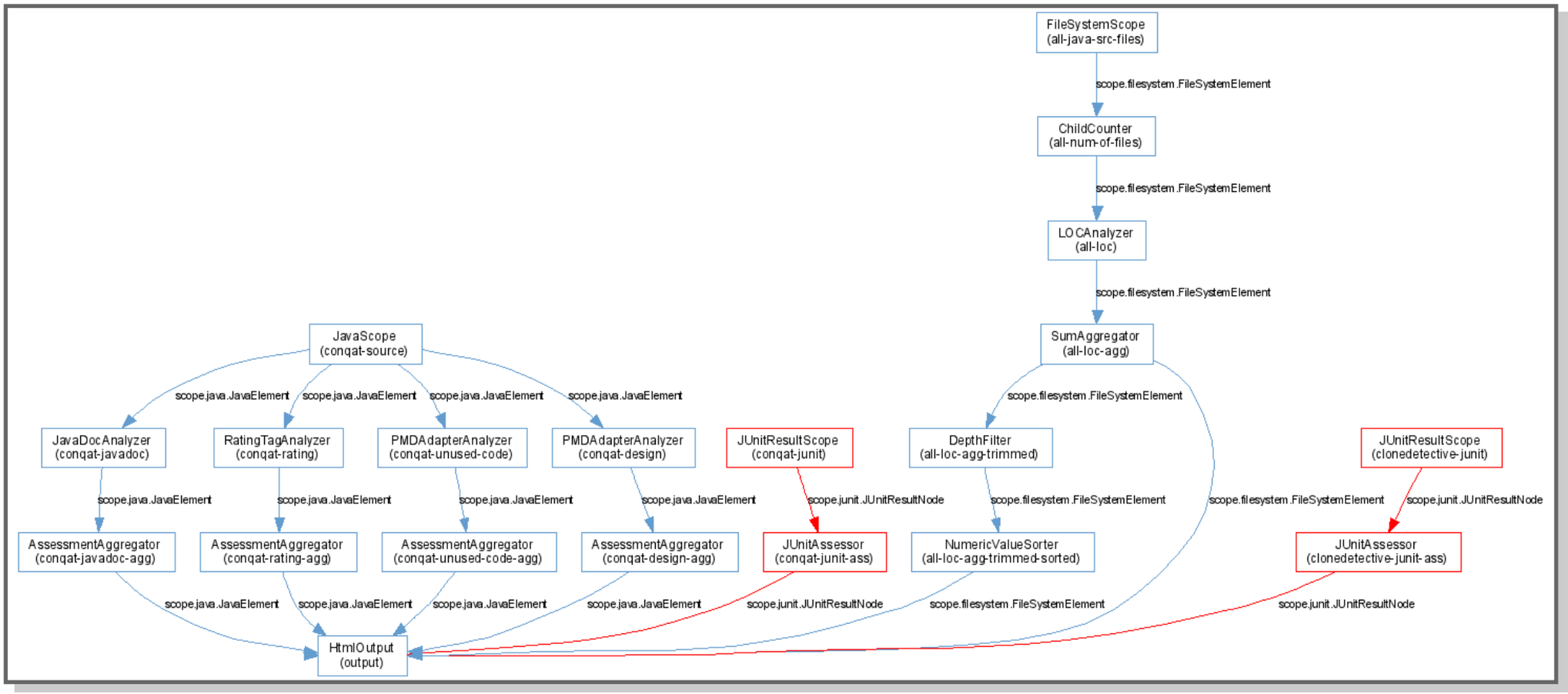
```
@AConfigProcessor(description = „Count LOC")
public class LOCAnalyzer implements IProcessor {

    @AConfigElement(name = „input)
    public void setRoot(
            @AConfigAttribute(name = „ref")  IFileSystemElement root) {
        ...
    }
    ....
}
```

ConQAT in Action

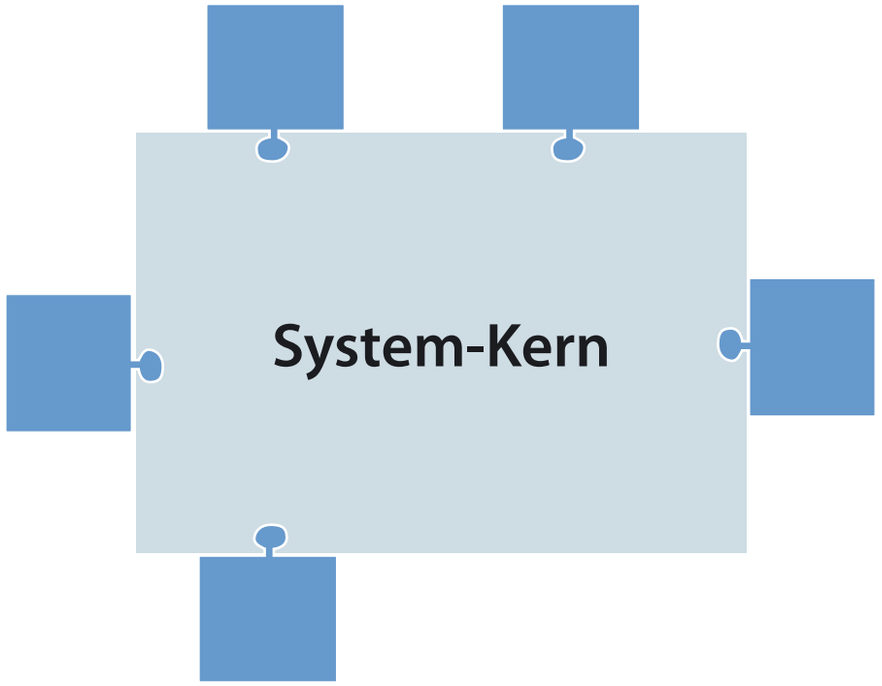


normalization		
NormalizationFactory		
word		
WordStructure		Method getNextWord() has no JavaDoc comment.
WordNormalization		Method getNextUnit() has no JavaDoc comment.
IStructure		

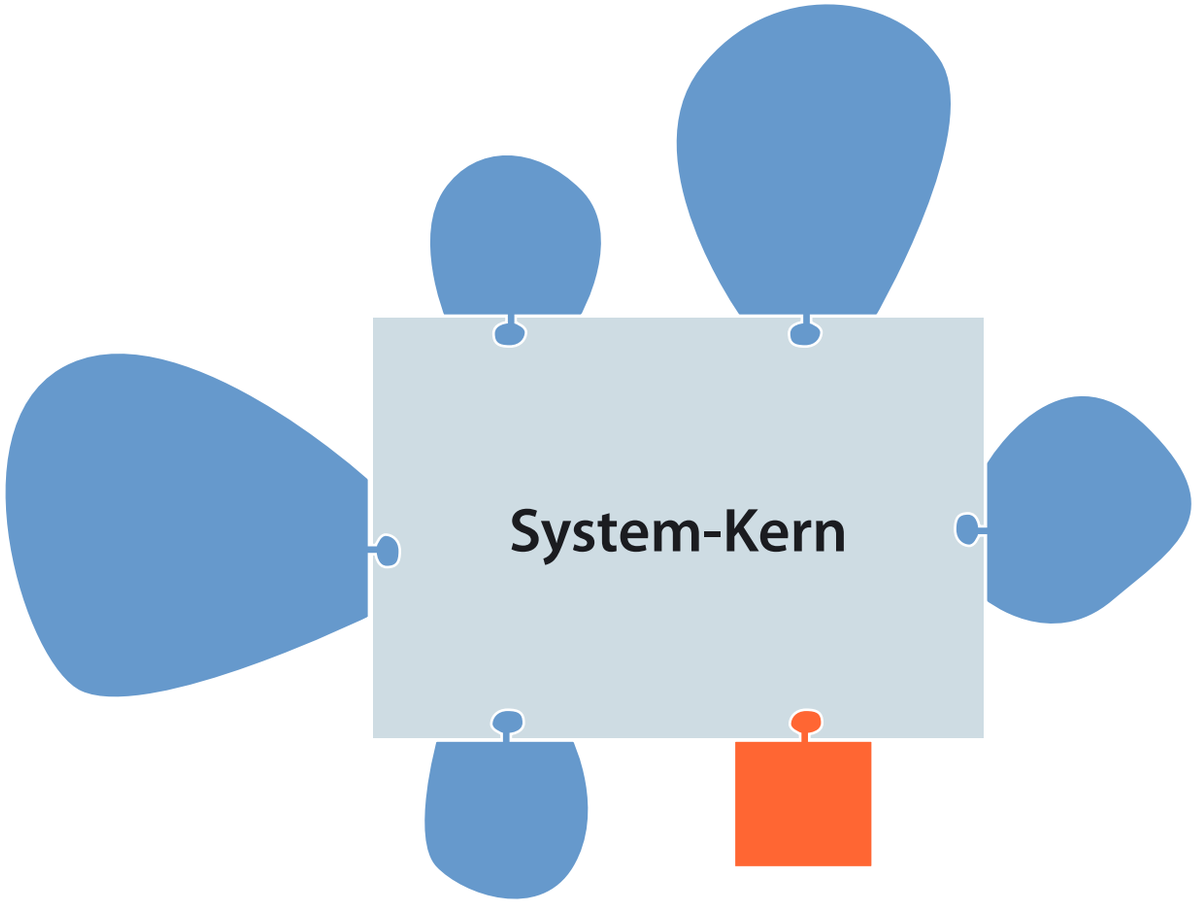
ConQAT in Action



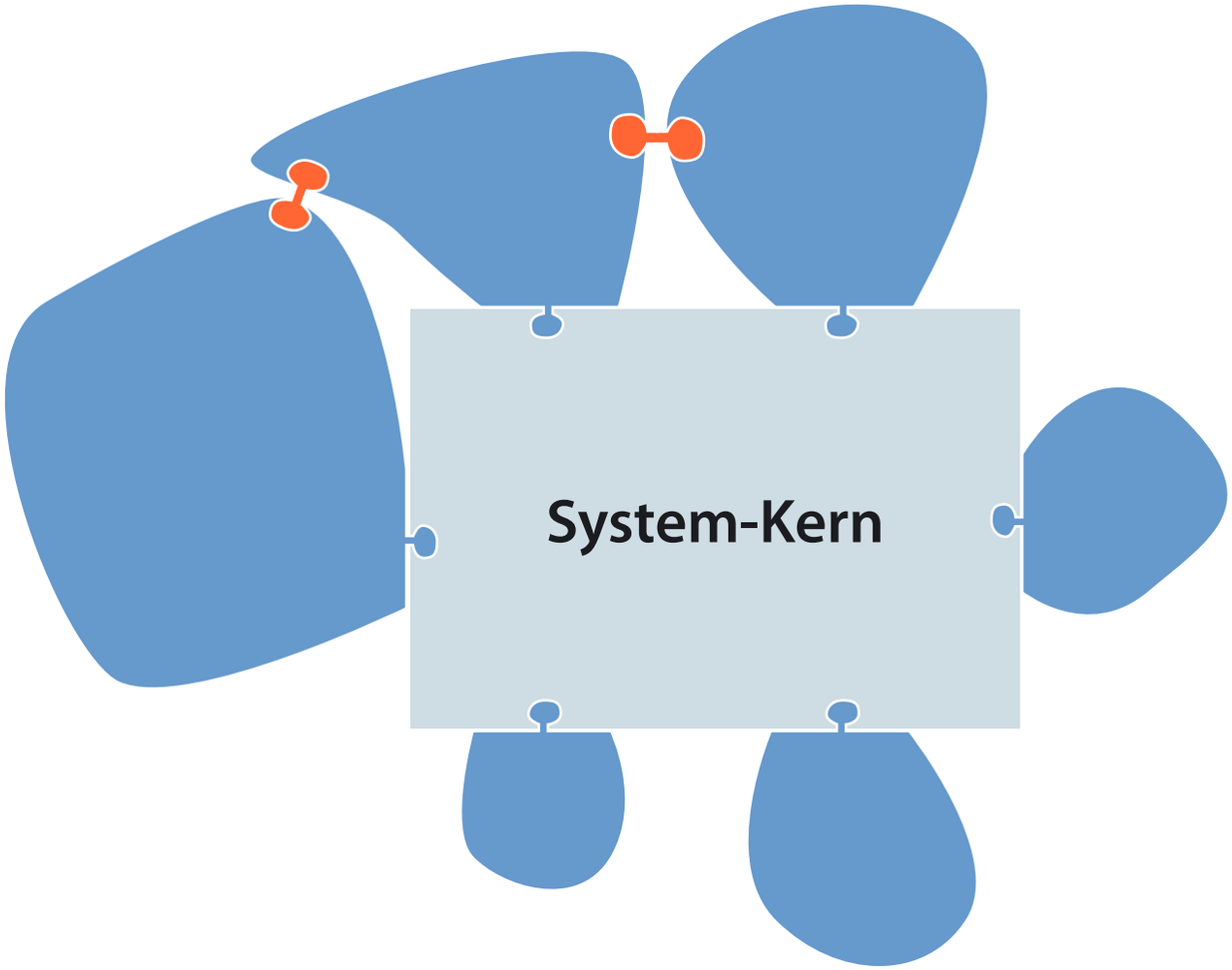
Evolution



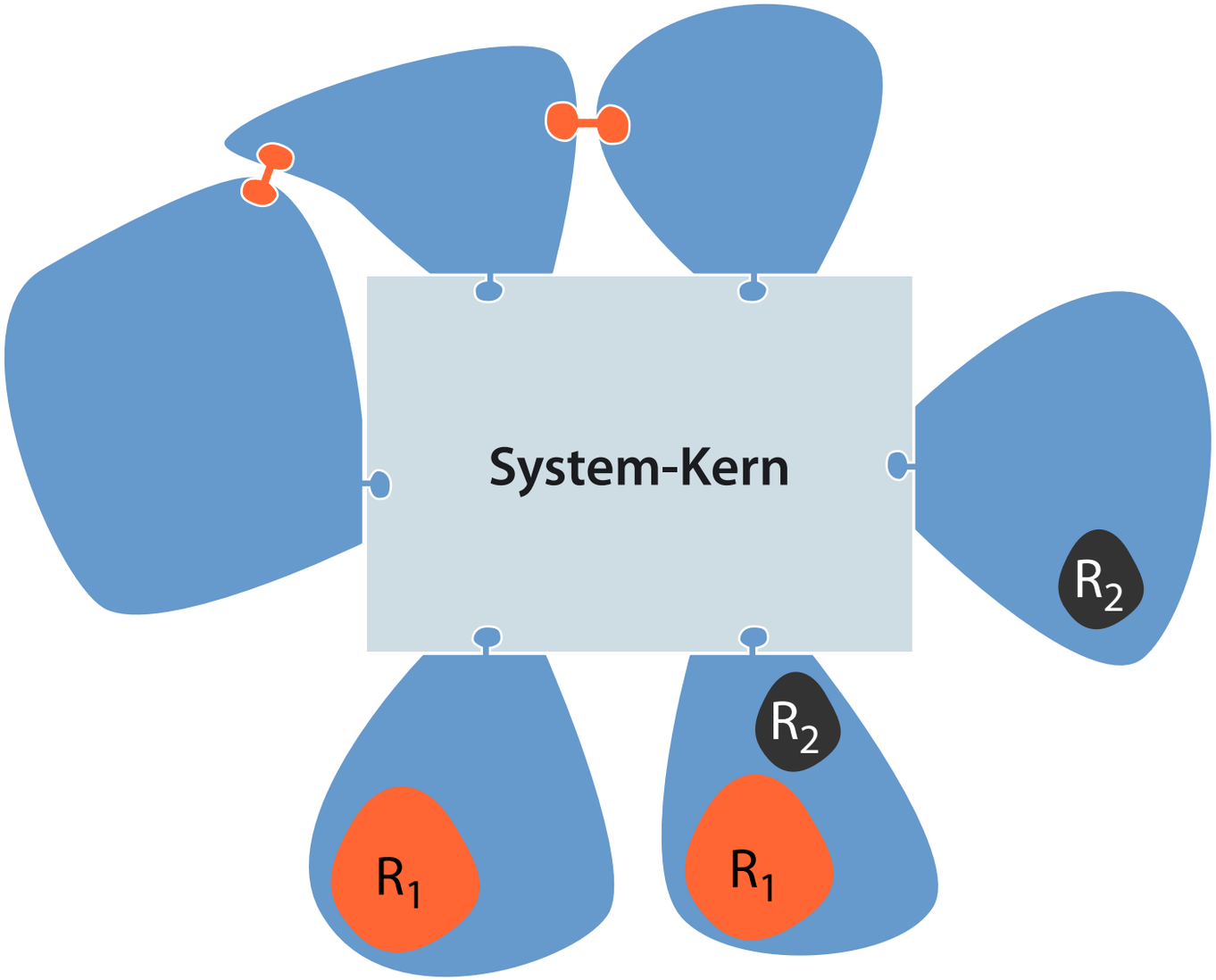
»Bloating«



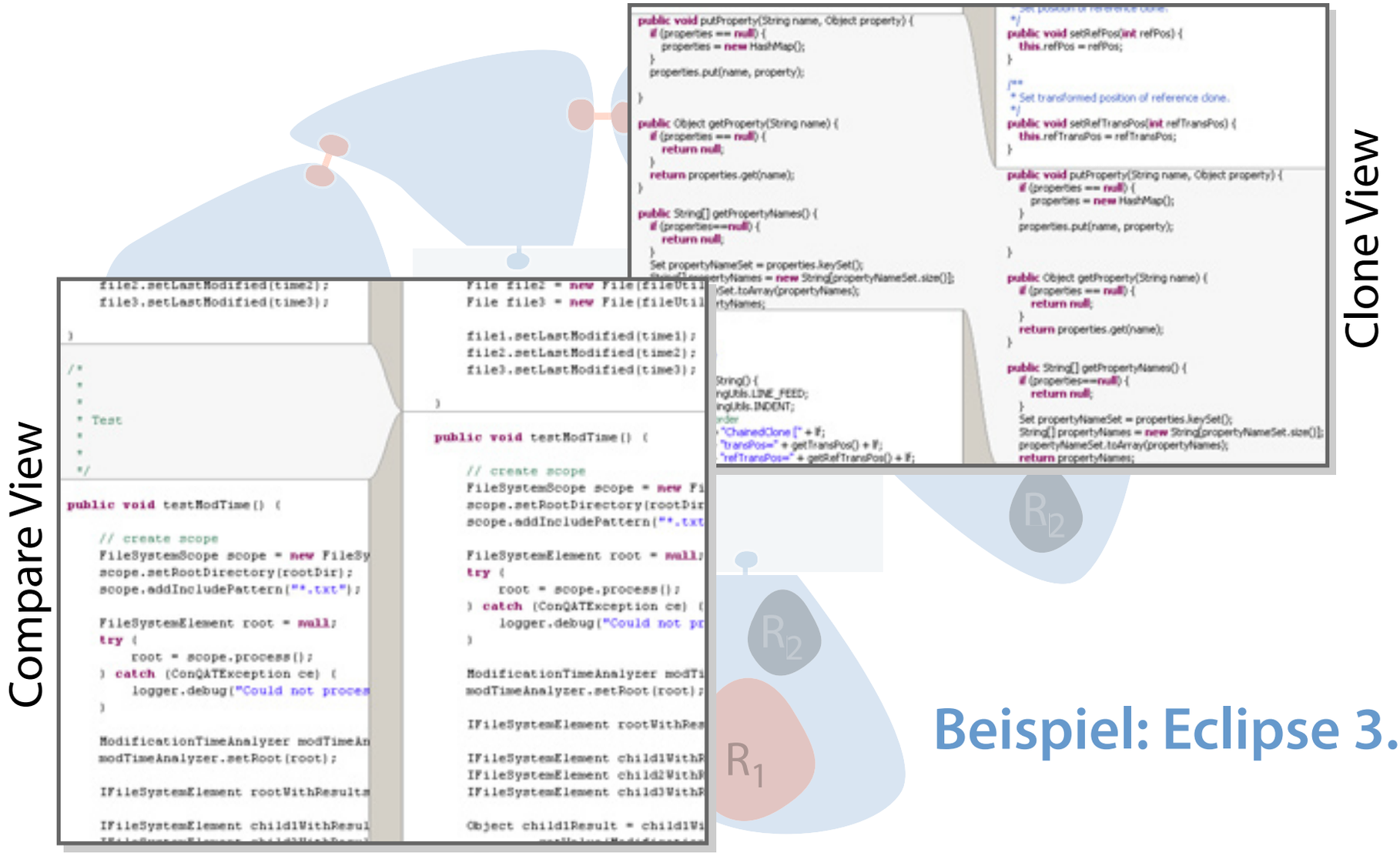
Abhängigkeiten



Redundanzen

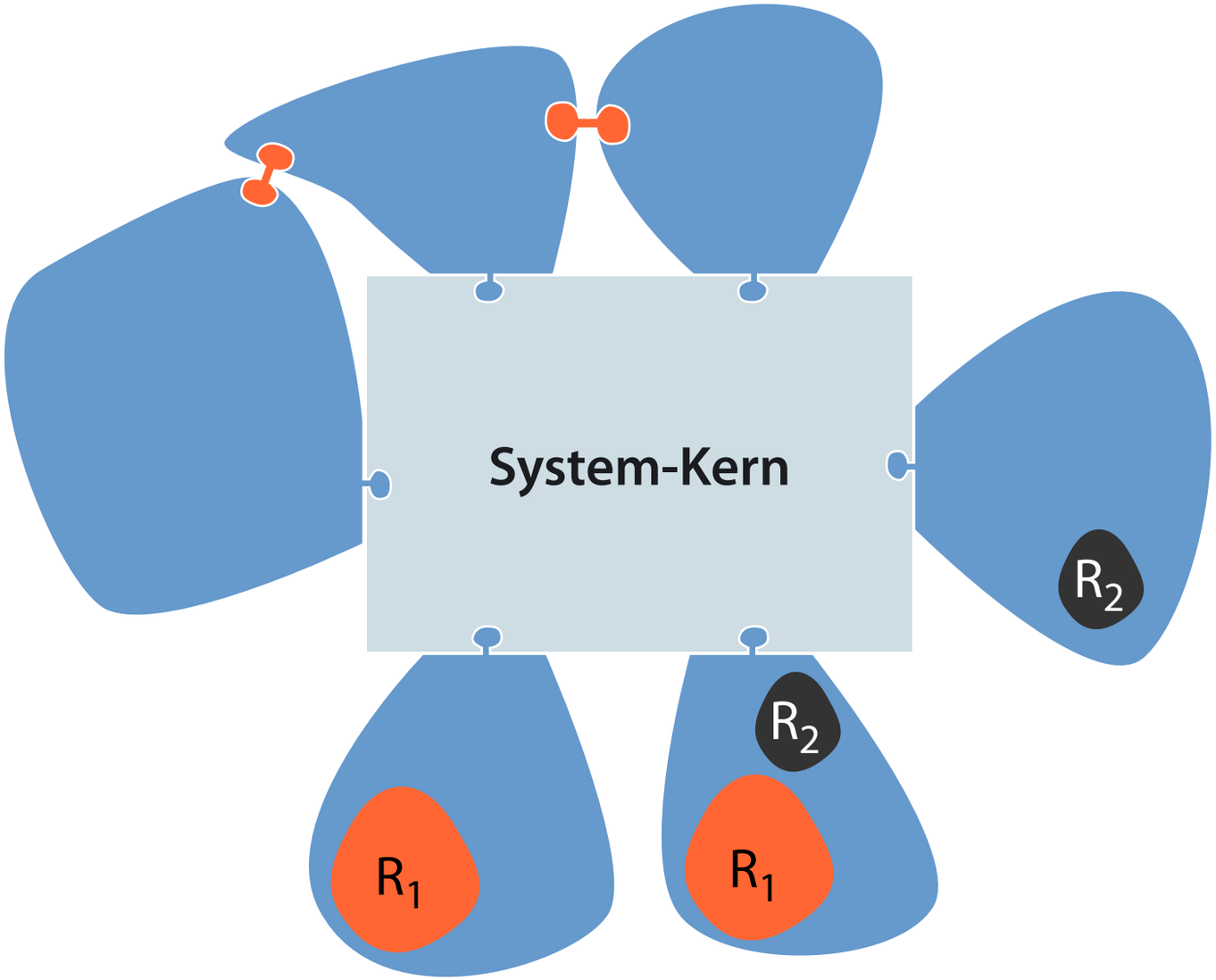


Redundanzen

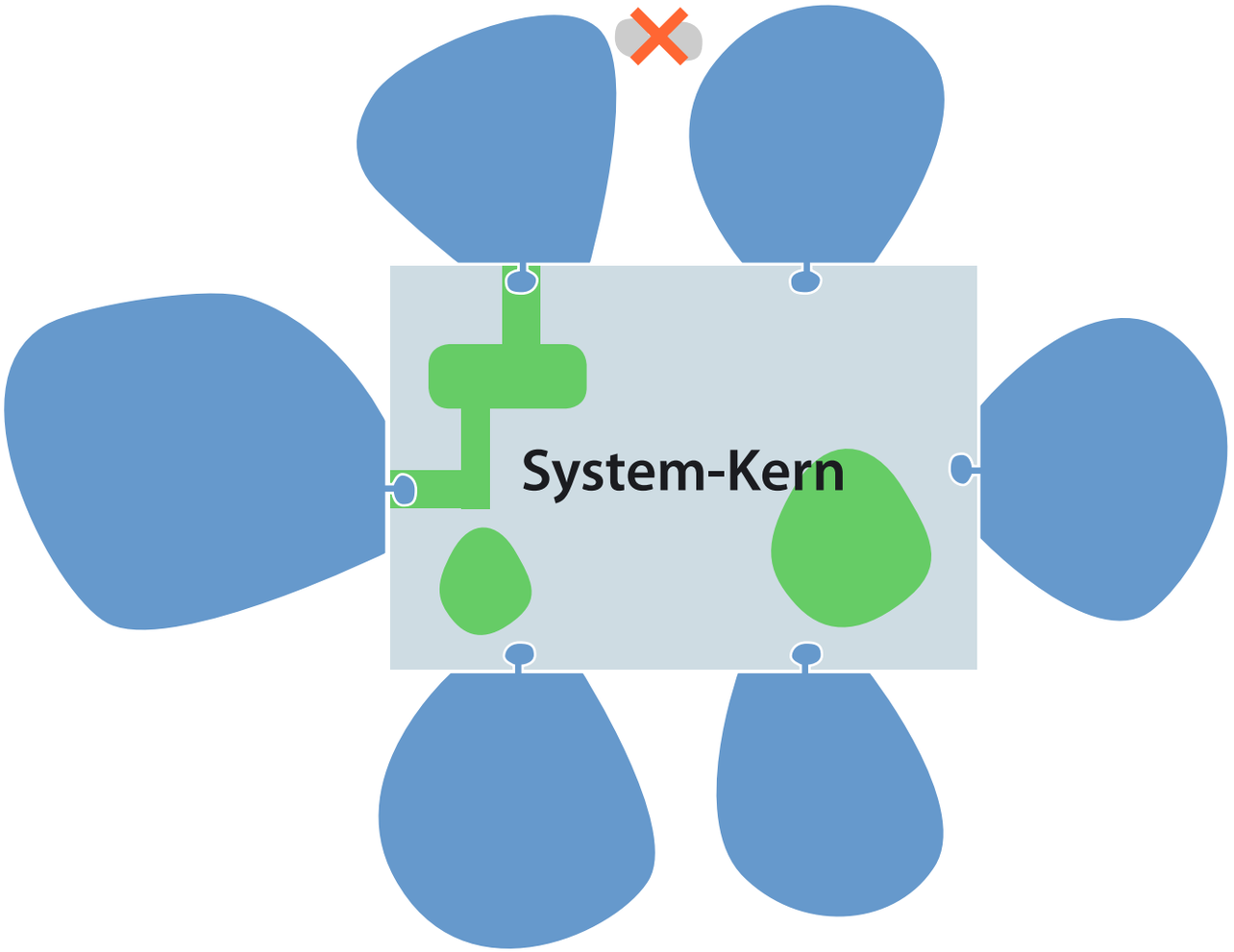


Beispiel: Eclipse 3.1

Redundanzen



Konsolidierung



Zusammenfassung

Agile Architektur-Evolution

- in unserem Projekt sehr positive Erfahrungen
  - gradueller Reifungsprozess *(Sandbox)*
  - kurze Entwicklungszeiten
  - ideal für verteilte Entwicklung
  - erleichtert durch Typ-Sicherheit
- kontinuierliches Controlling
  - Change-Management
  - manuelle Reviews
  - Clone-Detection
  - semi-automatische Architektur-Überprüfung
- kontinuierliche Konsolidierung *(Refactoring)*