Methodische Entwicklung von Modellierungswerkzeugen,
INFORMATIK 2009, Lübeck, 1.10.2009

Minimal-invasive generative Software-Entwicklung mit dem Eclipse Graphical Modeling Framework (GMF)

Jens Gulden

Lehrstuhl für Wirtschaftsinformatik und
Unternehmensmodellierung

ICB Institut für Informatik und Wirtschaftsinformatik

Institut für Informatik und
Wirtschaftsinformatik (ICB)

UNIVERSITÄT
DUISBURG
ESSEN

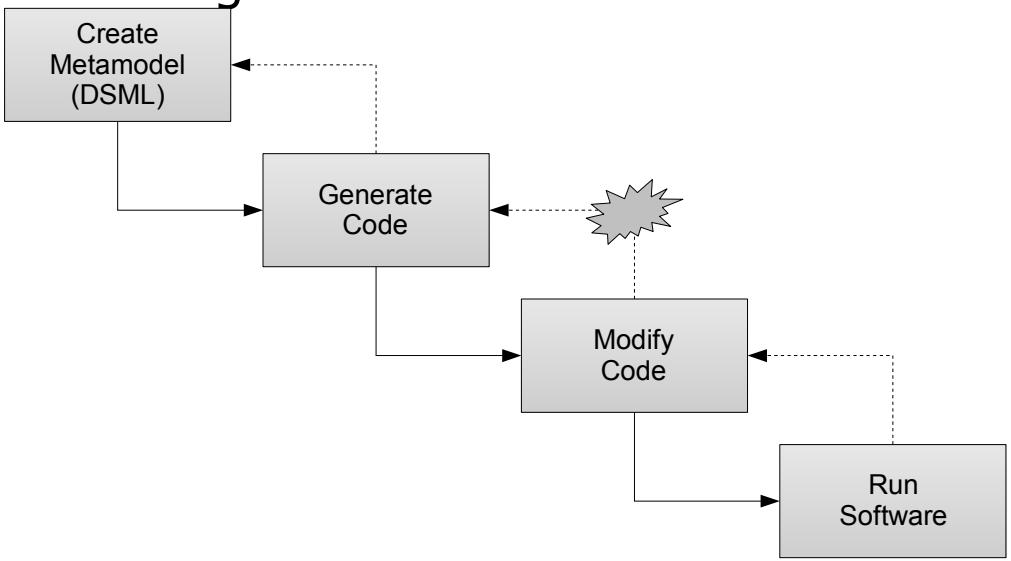
Generative Softwareentwicklung

- **100% Code-Generierung**: Idealbild der Theorie?
- *nein:* Wiederverwendungs**reichweite** stark **eingeschränkt**
- steigt durch **manuelle Änderungen**
  - maximale **Freiheitsgrade**
  - Strukturierungsmöglichkeiten **orthogonal** zum generierten Code
- → **Integration manueller Modifikationen** in theoretische Betrachtung

Methodische Integration manueller Programmierung

Bisher:

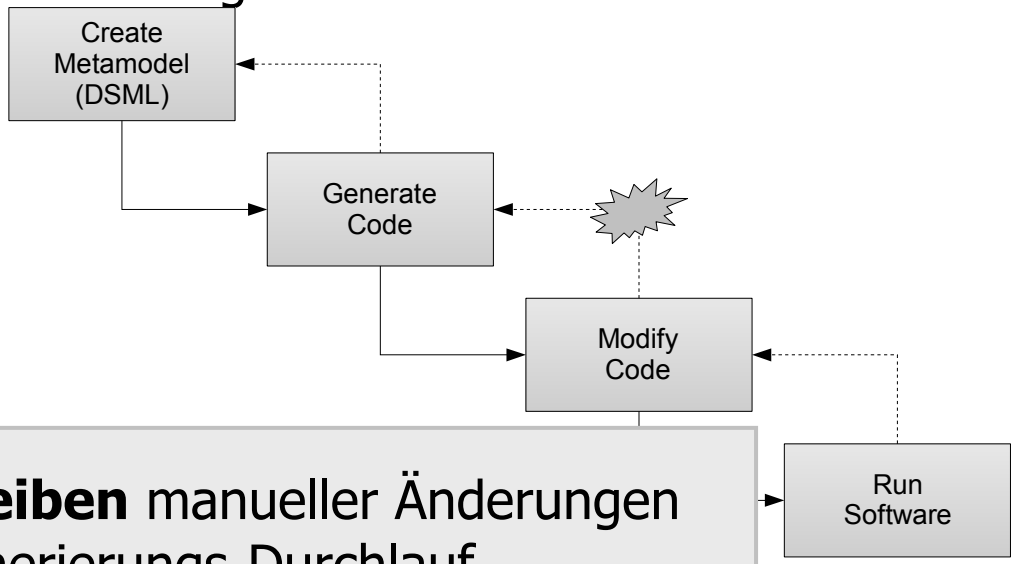
- **generative** Softwareentwicklung ↔ **manuelle** Modifikation
- generierter **Quellcode** ist einziger Ort, an dem **Änderungen persistent** gemacht werden



Methodische Integration manueller Programmierung

Bisher:

- **generative** Softwareentwicklung ↔ **manuelle** Modifikation
- generierter **Quellcode** ist einziger Ort, an dem **Änderungen persistent** gemacht werden

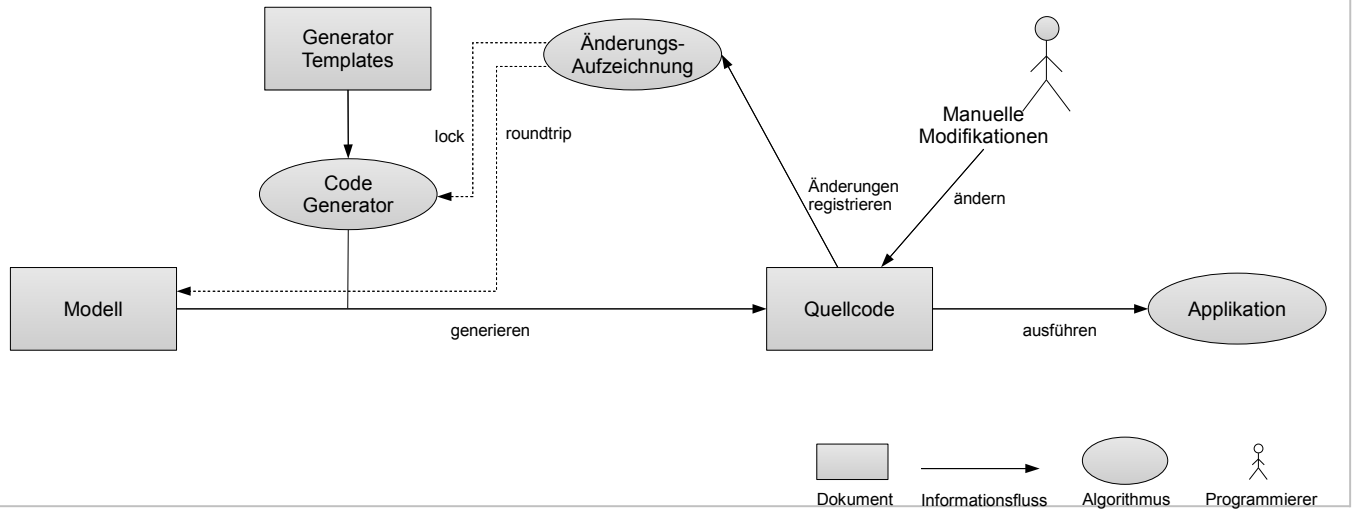


Problem: **Überschreiben** manueller Änderungen bei neuem Code-Generierungs-Durchlauf.

Traditionelle Ansätze

- Überschreiben manueller Änderungen traditionell gelöst z.B. durch:
  - **Protected Block Verfahren**
  - **Roundtrip-Verfahren**
  - **3-Wege-Vergleichsverfahren**

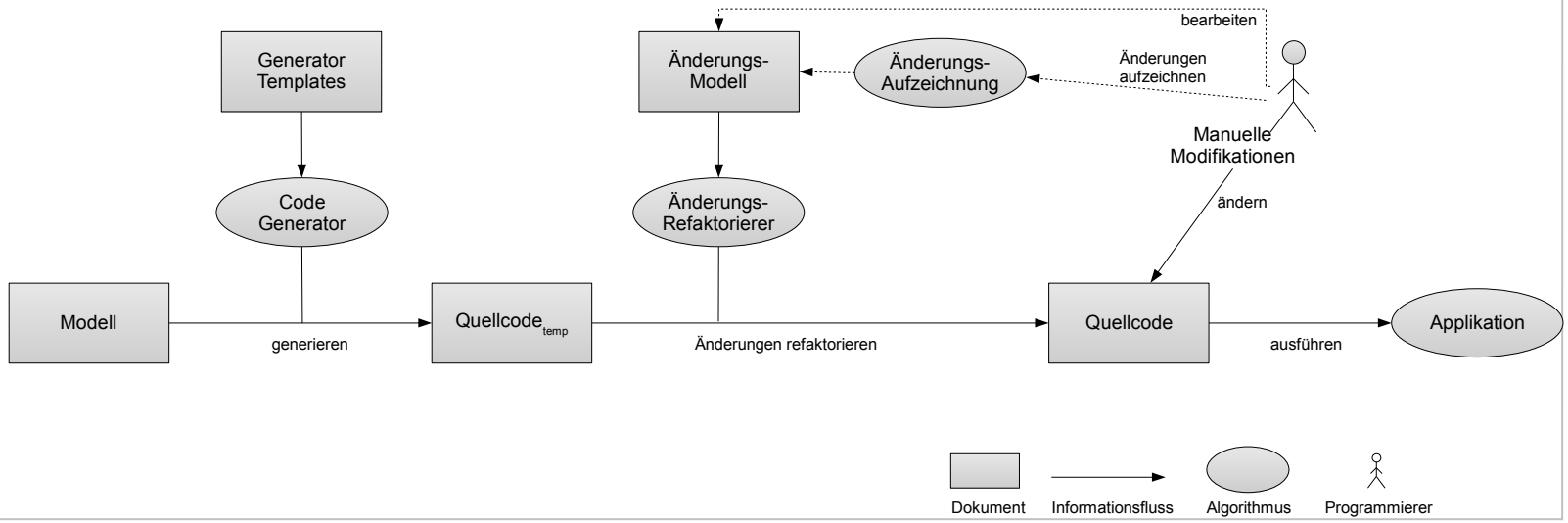
→ Symptom-Bekämpfung, keine echte Lösung



Methodische Alternative

Besser:

- **Verhindern**, dass Problem des Überschreibens von Code überhaupt erst **entsteht**

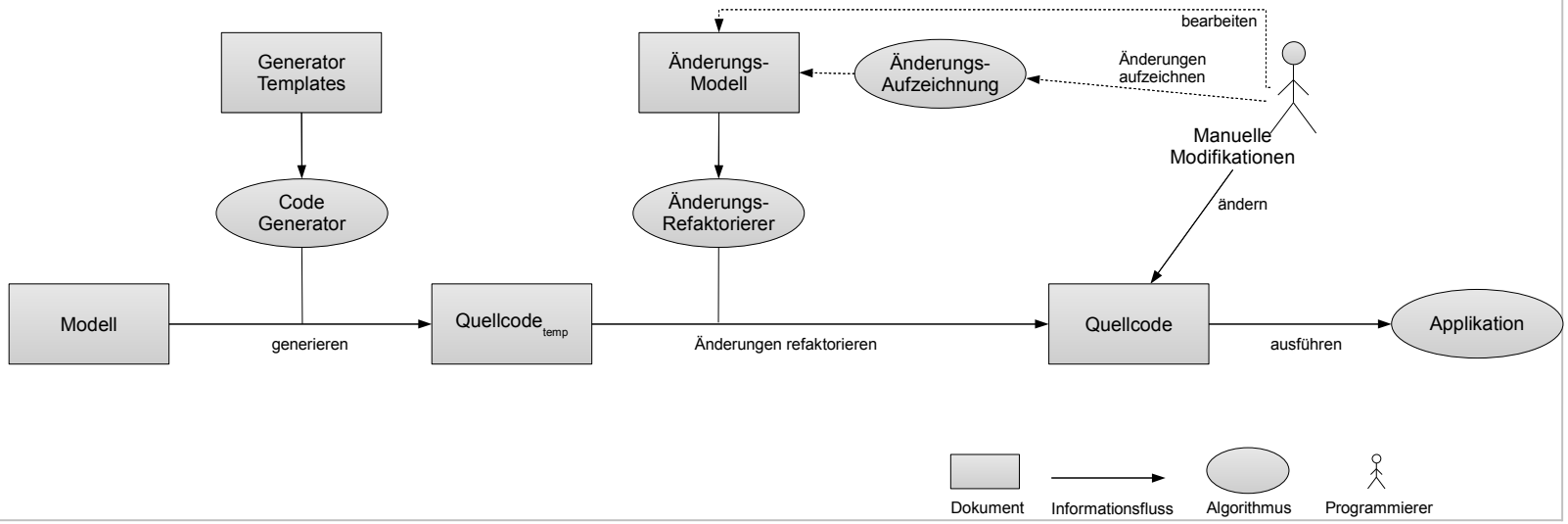


Methodische Alternative

Besser:

- **Verhindern**, dass Problem des Überschreibens von Code überhaupt erst **entsteht**

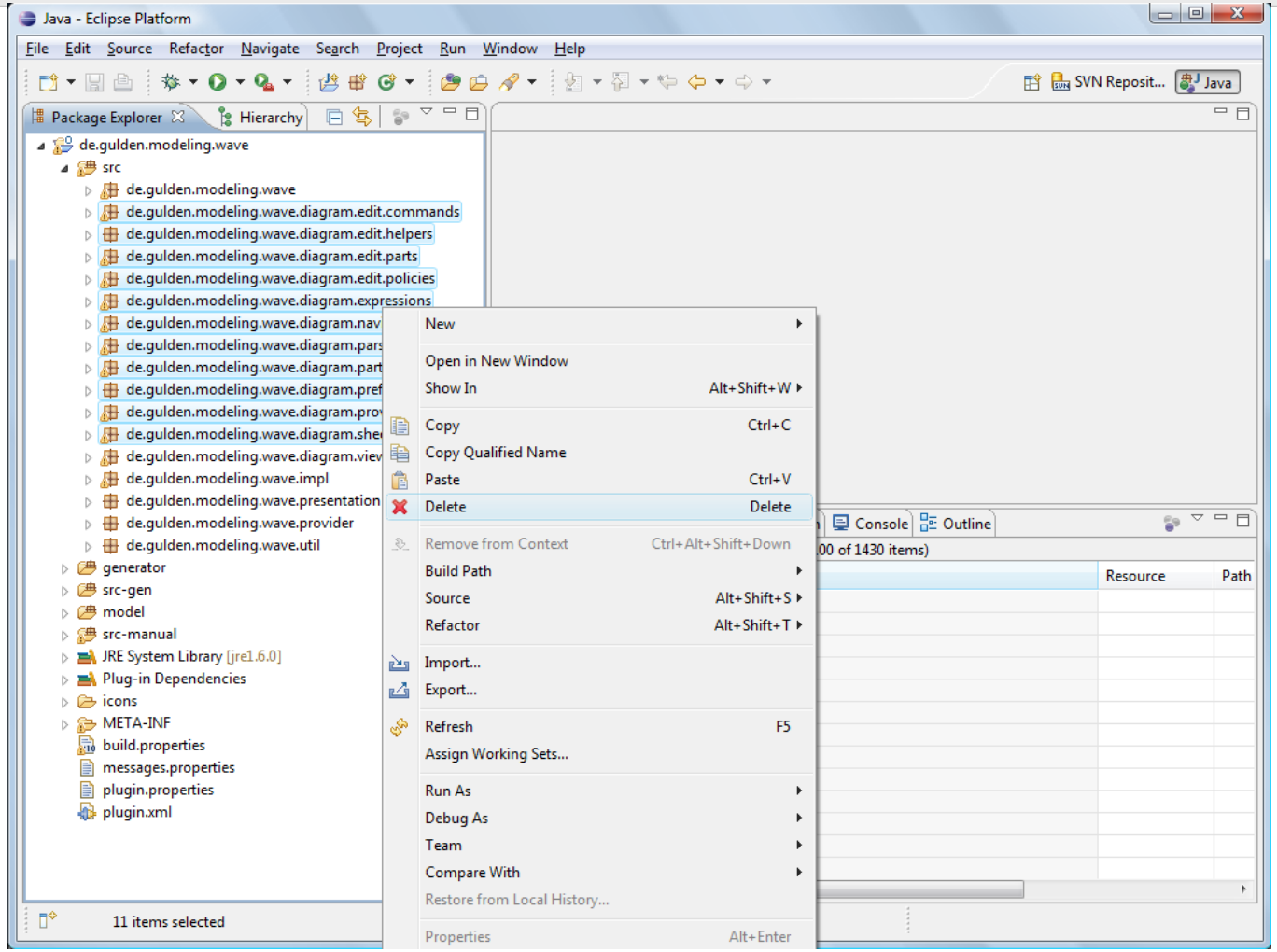
Lösen i.S.v. **Auflösen** eines Problems.



Methodische Alternative

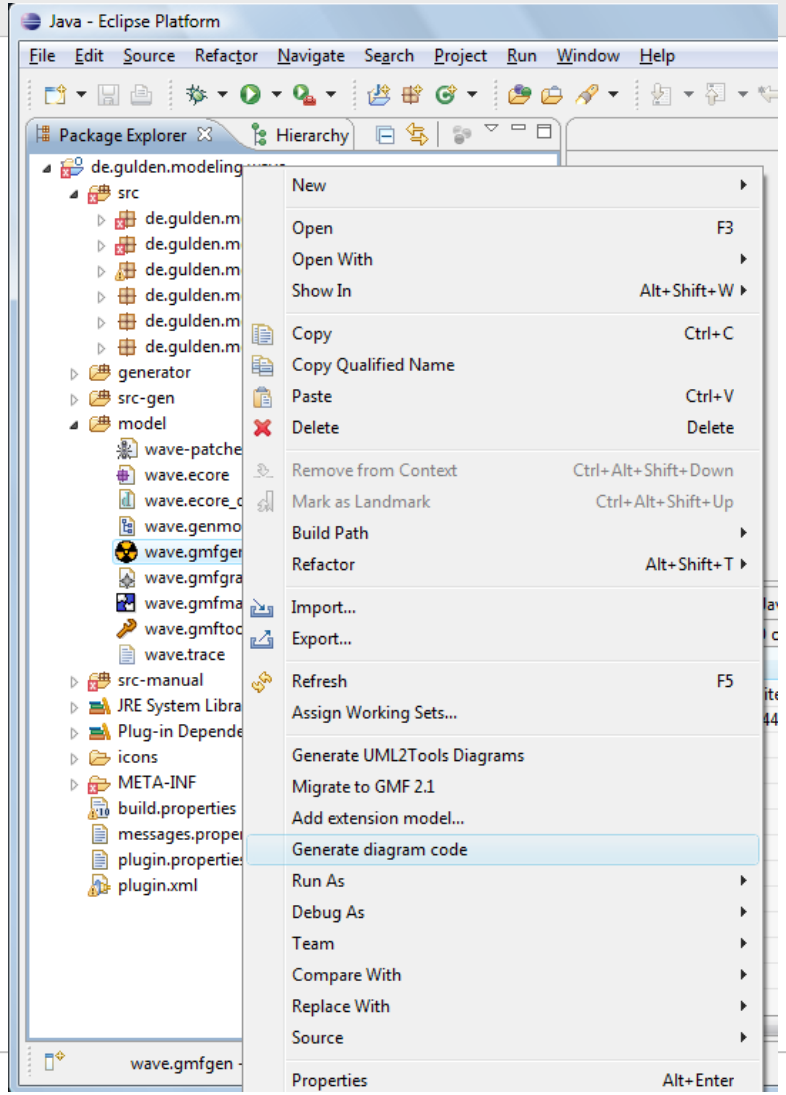
- Überschreiben von Quellcode ist **nicht per se** ein Problem
- **Grund**, dass ein Problem entsteht, ist, dass manuelle Änderungen **nur im generierten** Quellcode erfasst werden
- Idee der **„Änderung"** kann aber anders expliziert werden, z.B.
  - externes **Änderungs-Protokoll**
  - vgl. **Liste von Refactoring** Operationen

Demonstration: WAVE



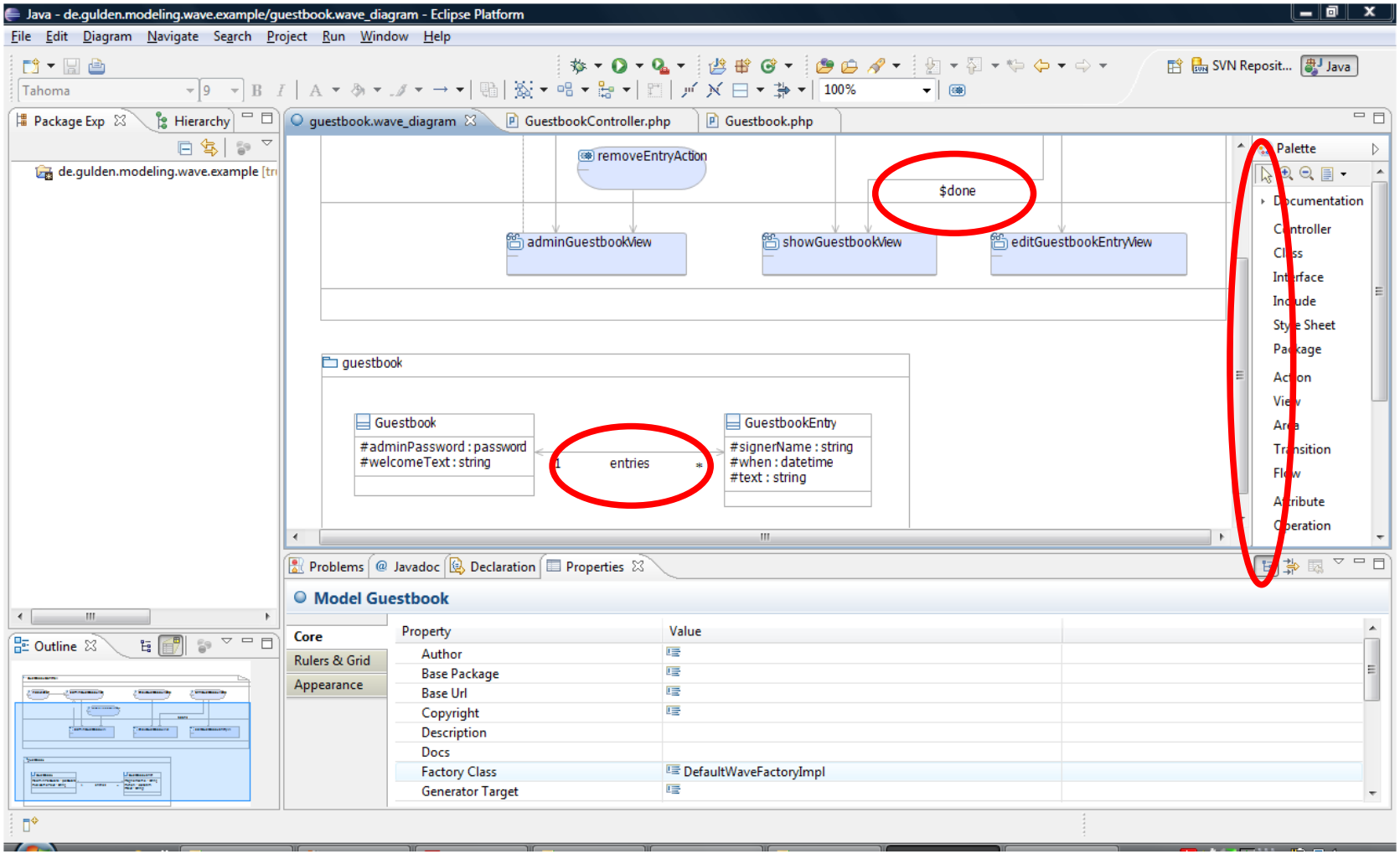
http://wave.berlios.de/

Demonstration: WAVE



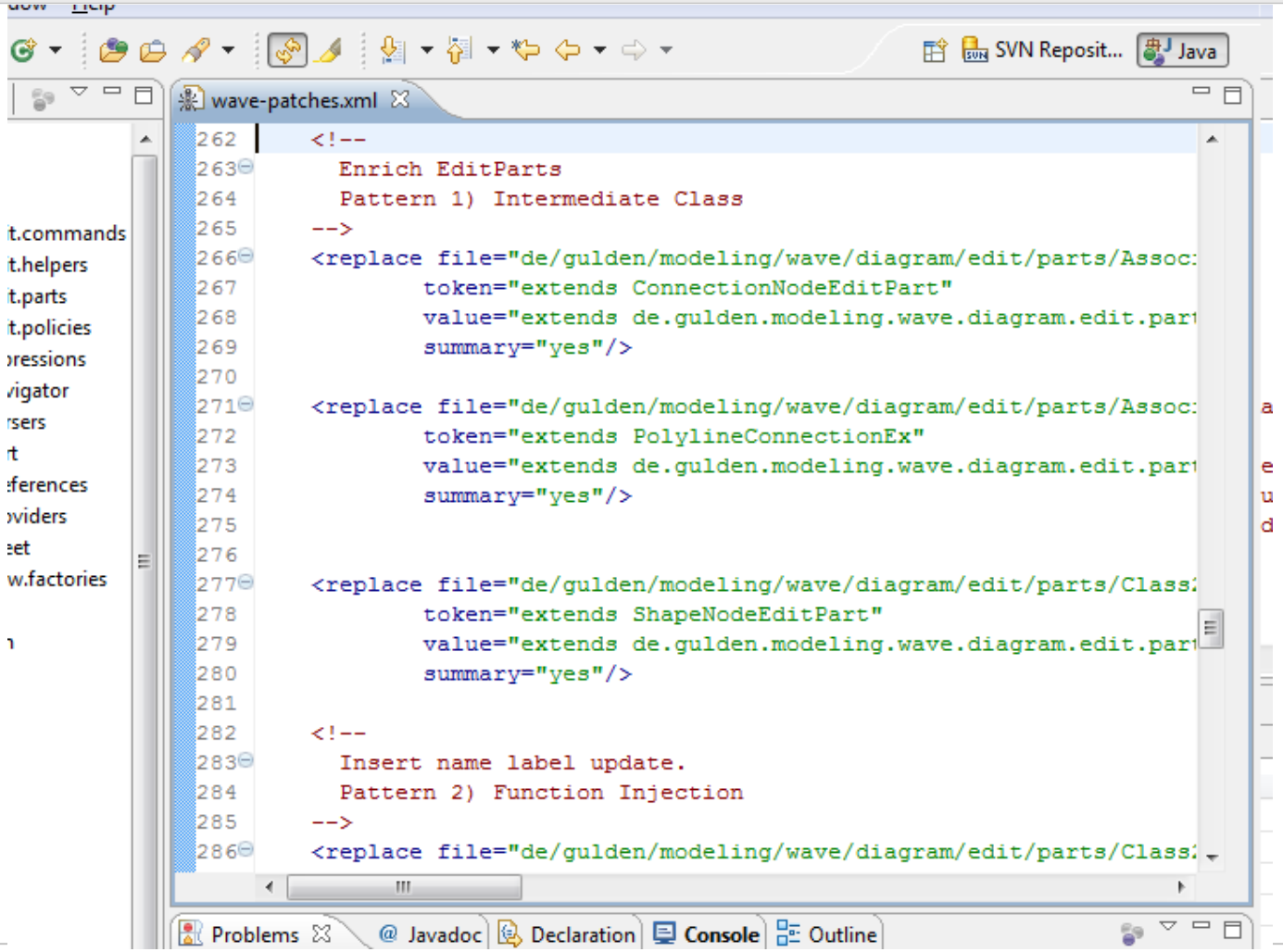
http://wave.berlios.de/

Demonstration: WAVE



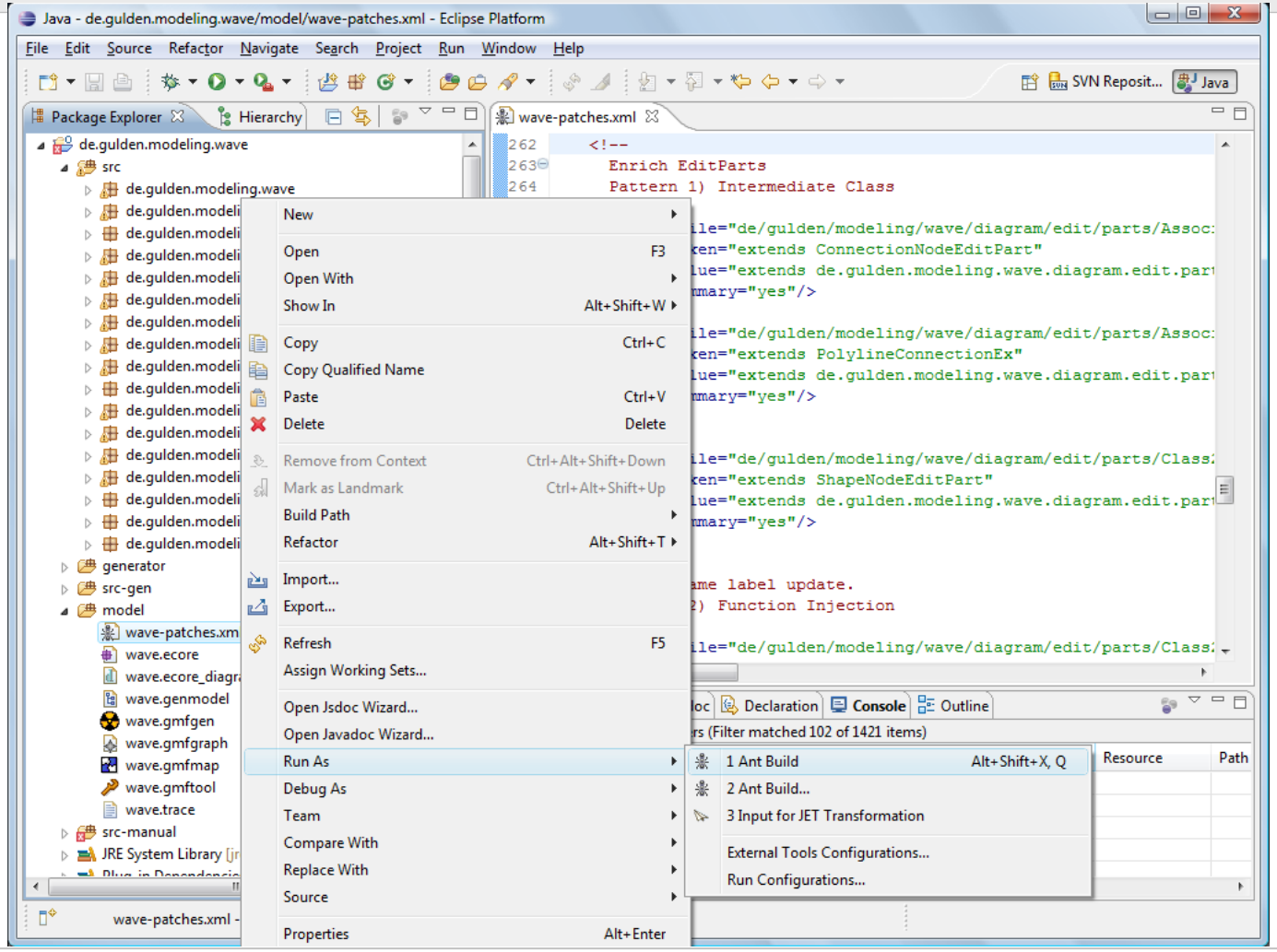
http://wave.berlios.de/

Demonstration: WAVE



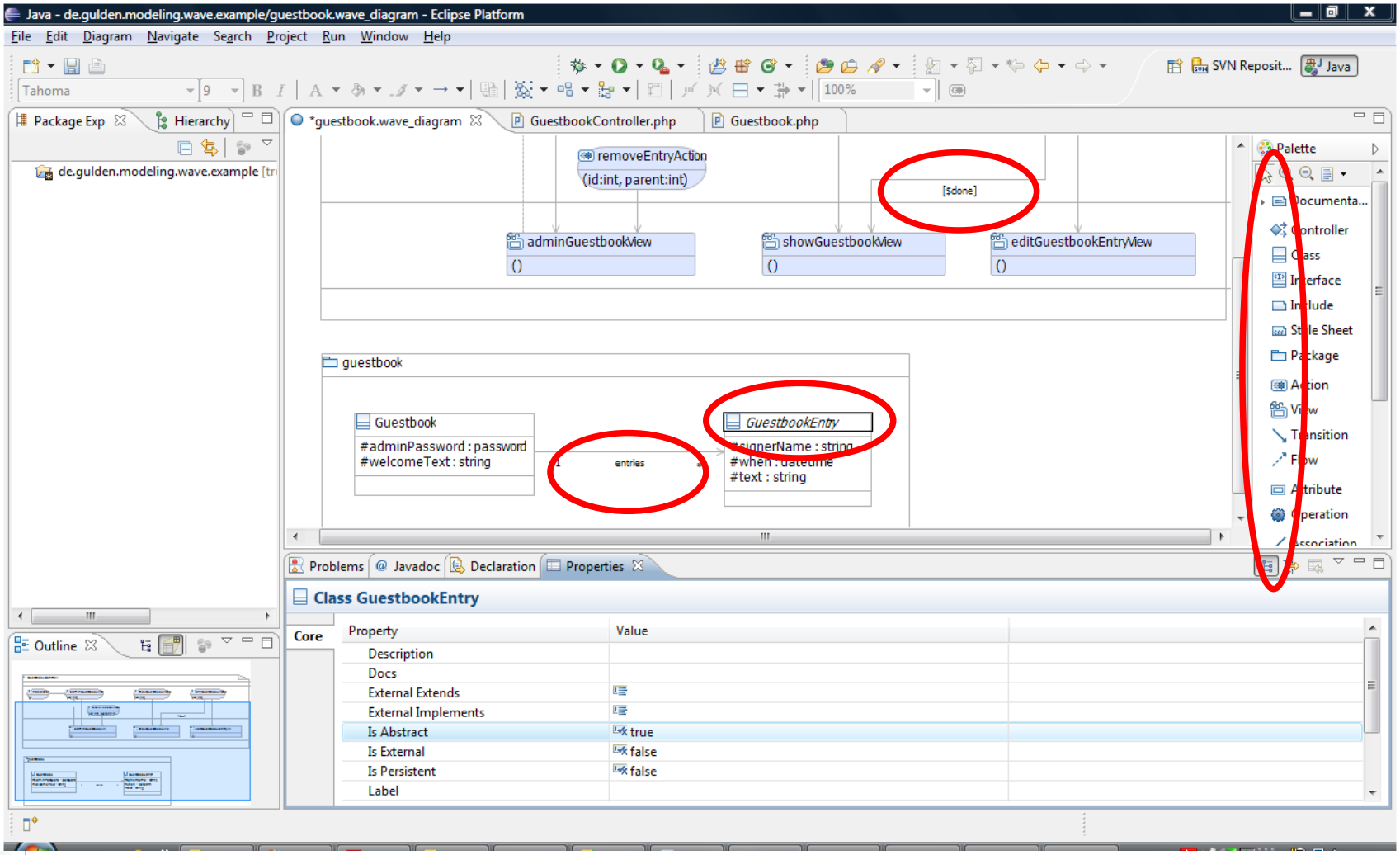
http://wave.berlios.de/

Demonstration: WAVE



http://wave.berlios.de/

Demonstration: WAVE



http://wave.berlios.de/

Änderungs-Modell

- Mögliche **Varianten**:
  - **explizit**: Programmierer erstellt Änderungs-Protokoll
    → im Folgenden
  - **implizit**: Entwicklungsumgebung erfasst manuelle Änderungen am Quellcode, Liste von Refactoring-Operationen wird erstellt

„Minimal invasive Modifikation“

Anforderungen:

- optimales **Verhältnis aus Änderungsaufwand und Nutzen** durch möglichst geringe syntaktische Änderungen am generierten Code
  - Änderungs-Protokoll soll **leicht zu pflegen** sein
  - spätere **Änderungen am Code-Generator** sollen effizient handhabbar bleiben
  - Evolution **zu Grunde liegender Frameworks** muss möglich bleiben

Muster minimal invasiver Eingriffe

- Minimal-invasive Änderungen an objektorientiertem Java-Code können in **Grundmuster** eingeteilt werden
  - Forderung nach Minimalität grenzt Menge der sinnvoll betrachteten Muster ein
- **„Refactoring zweiter Ordnung"**
  - Liste von traditionellen Refactoring-Operationen
  - z.B.: Kombination aus Änderung von Superklassen-Referenz und Erzeugen einer neuen „Zwischenklasse"
- **„Aspekt-Orientierung"** auf Code-Ebene zur Compile-Zeit

Muster 1 – Zwischenklasse

- **zwischen ursprünglicher Oberklasse und aktueller Klasse** wird eine Zwischenschicht in der Vererbungshierarchie eingeführt, die die gewünschte Funktionalität bereitstellt
  - Erlaubt Ergänzen / Ersetzen von Methoden

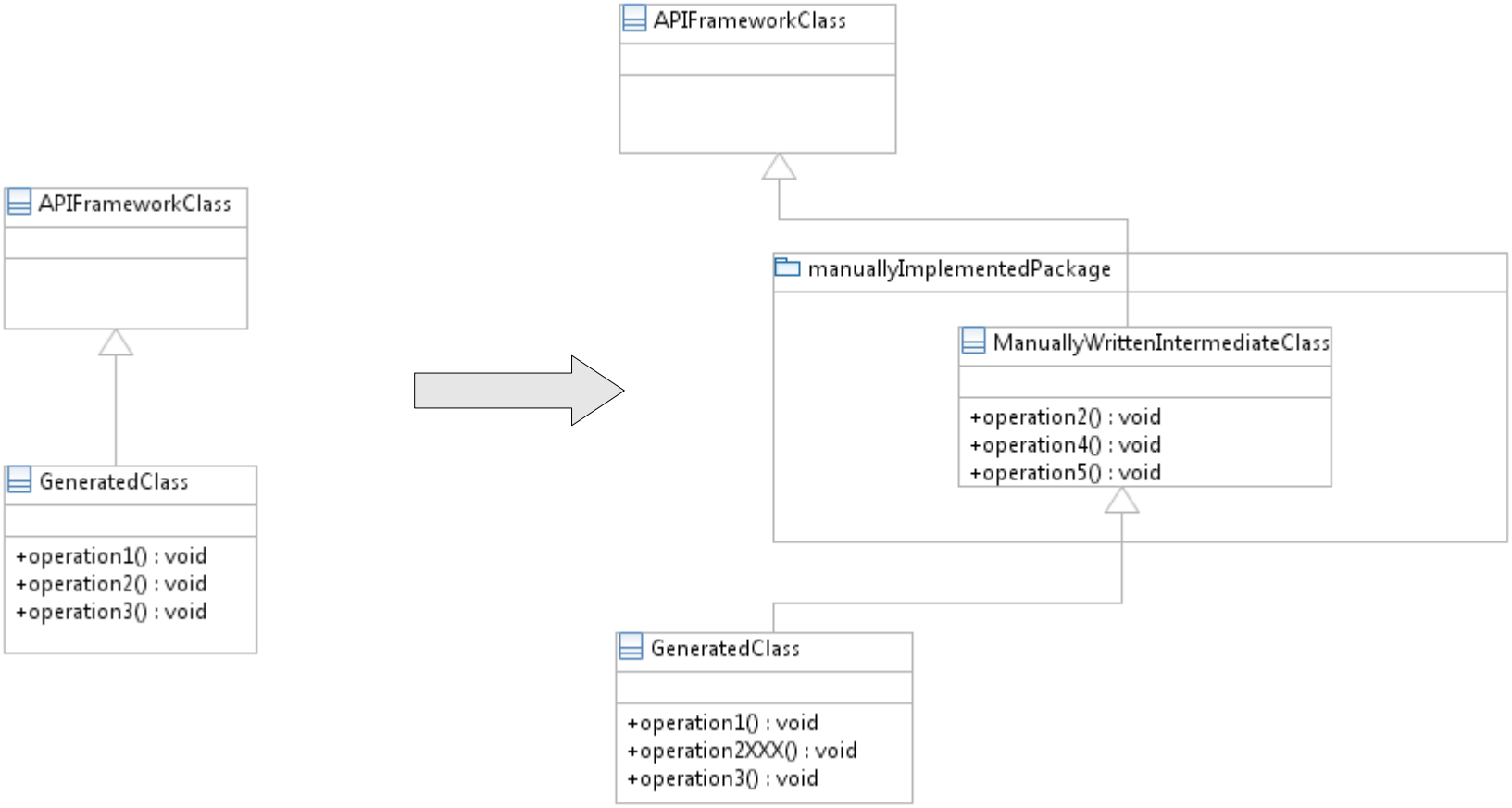
1) `extends`-Klausel patchen

   Suchen & ersetzen leicht, Idempotenz sichergestellt

2) externe Klassendefinition manuell erstellen

3) ggf. zu ändernde Methoden in generierter Klasse deaktivieren
   - `void getWert()` → `void getXXXWert()`

Muster 1 – Zwischenklasse



Muster 1 – Zwischenklasse

- Beispiel

```
  <!--
  Enrich EditParts
  Pattern 1) Intermediate Class
  -->
<replace
  file="de/gulden/modeling/wave/diagram/edit/parts/AssociationRelationshipEditPart.java"
  token="extends ConnectionNodeEditPart"
  value="extends de.gulden.modeling.wave.diagram.edit.parts.AssociationRelationshipEditPartBase"
  summary="yes"/>

<replace
  file="de/gulden/modeling/wave/diagram/edit/parts/AssociationRelationshipEditPart.java"
  token="extends PolylineConnectionEx"
  value="extends de.gulden.modeling.wave.diagram.edit.parts.AssociationRelationshipFigureBase"
  summary="yes"/>
```

Muster 1 – Zwischenklasse

- Beispiel

```
  <!--
  Enrich EditParts
  Pattern 1) Intermediate Class
  -->
<replace
  file="de/gulden/modeling/wave/diagram/edit/parts/AssociationRelationshipEditPart.java"
  token="extends ConnectionNodeEditPart"
  value="extends de.gulden.modeling.wave.diagram.edit.parts.AssociationRelationshipEditPartBase"
  summary="yes"/>

<replace
  file="de/gulden/modeling/wave/diagram/edit/parts/AssociationRelationshipEditPart.java"
  token="extends PolylineConnectionEx"
  value="extends de.gulden.modeling.wave.diagram.edit.parts.AssociationRelationshipFigureBase"
  summary="yes"/>
```

Muster 1 – Zwischenklasse

- Beispiel

```
  <!--
  Enrich EditParts
  Pattern 1) Intermediate Class
  -->
<replace
  file="de/gulden/modeling/wave/diagram/edit/parts/AssociationRelationshipEditPart.java"
  token="extends ConnectionNodeEditPart"
  value="extends de.gulden.modeling.wave.diagram.edit.parts.AssociationRelationshipEditPartBase"
  summary="yes"/>

<replace
  file="de/gulden/modeling/wave/diagram/edit/parts/AssociationRelationshipEditPart.java"
  token="extends PolylineConnectionEx"
  value="extends de.gulden.modeling.wave.diagram.edit.parts.AssociationRelationshipFigureBase"
  summary="yes"/>
```

Muster 2 – Funktions-Injektion

- **Algorithmen verändern** durch Einfügen von **Funktionsaufrufen**
- Optional: Aufruf statischer Funktion in externer Klasse
- Suchen-und-Ersetzen schwieriger
  - eindeutiges Muster finden
  - Idempotenz gewährleisten, obwohl am Rand eingefügt wird (z.B. „Semikolon-Trick“)
- 1) **Patch**
- 2) statische Funktion in externer Klasse **manuell** erstellen

Muster 2 – Funktions-Injektion

- Beispiel

```
<!--
 Let conditions on Transitions and Flows appear in square brackets.
 [de.gulden.modeling.wave.util.WaveUtil.toCondition(String)]
 Pattern 2) Function Injection
-->
<replace
  file=".../wave/diagram/edit/parts/ActionToViewTransitionConditionEditPart.java"
  token="setLabelTextHelper(getFigure(), getLabelText());"
  value="setLabelTextHelper(getFigure(),
         de.gulden.modeling.wave.util.WaveUtil.toCondition(getLabelText()));"
  summary="yes" />

<replace
  file=".../wave/diagram/edit/parts/ViewTransitionConditionEditPart.java"
  token="setLabelTextHelper(getFigure(), getLabelText());"
  value="setLabelTextHelper(getFigure(),
         de.gulden.modeling.wave.util.WaveUtil.toCondition(getLabelText()));"
  summary="yes" />
```

Muster 2 – Funktions-Injektion

- Beispiel

```
<!--
 Let conditions on Transitions and Flows appear in square brackets.
 [de.gulden.modeling.wave.util.WaveUtil.toCondition(String)]
 Pattern 2) Function Injection
-->
<replace
  file=".../wave/diagram/edit/parts/ActionToViewTransitionConditionEditPart.java"
  token="setLabelTextHelper(getFigure(), getLabelText());"
  value="setLabelTextHelper(getFigure(),
         de.gulden.modeling.wave.util.WaveUtil.toCondition(getLabelText()));"
  summary="yes" />

<replace
  file=".../wave/diagram/edit/parts/ViewTransitionConditionEditPart.java"
  token="setLabelTextHelper(getFigure(), getLabelText());"
  value="setLabelTextHelper(getFigure(),
         de.gulden.modeling.wave.util.WaveUtil.toCondition(getLabelText()));"
  summary="yes" />
```

Muster 2 – Funktions-Injektion

- Beispiel

```
<!--
 Let conditions on Transitions and Flows appear in square brackets.
 [de.gulden.modeling.wave.util.WaveUtil.toCondition(String)]
 Pattern 2) Function Injection
-->
<replace
  file=".../wave/diagram/edit/parts/ActionToViewTransitionConditionEditPart.java"
  token="setLabelTextHelper(getFigure(), getLabelText());"
  value="setLabelTextHelper(getFigure(),
         de.gulden.modeling.wave.util.WaveUtil.toCondition(getLabelText()));"
  summary="yes" />

<replace
  file=".../wave/diagram/edit/parts/ViewTransitionConditionEditPart.java"
  token="setLabelTextHelper(getFigure(), getLabelText());"
  value="setLabelTextHelper(getFigure(),
         de.gulden.modeling.wave.util.WaveUtil.toCondition(getLabelText()));"
  summary="yes" />
```

Muster 3 – Constructor-Ergänzung

- Spezialfall von Funktions-Injektion
- ganze **Constructor-Methode** ist **einzufügen**
- normalerweise besser via **Zwischenklasse**,
  für Constructor aber **nicht möglich**
- vorgehen:
- 1) Suchen nach „`super();`“ eines Default-Constructors
- 2) Ersetzen mit

```
super() ; }
public MyClass(String a, int b) {
 ... manual code ...
```

Muster 3 – Constructor-Ergänzung

- Beispiel

```
  <!--
    Add additional constructor.
    Pattern 3) Constructor enhancement
  -->
<replace file="de/gulden/modeling/wave/impl/TaggedValueImpl.java" summary="yes">

  <replacetoken>super();</replacetoken>

  <replacevalue>super() ;
  }

  public TaggedValueImpl(String key, String value) {
        super() ;
        this.setKey(key);
        this.setValue(value);
  </replacevalue>
  </replace>
```

Muster 3 – Constructor-Ergänzung

- Beispiel

```
  <!--
    Add additional constructor.
    Pattern 3) Constructor enhancement
  -->
<replace file="de/gulden/modeling/wave/impl/TaggedValueImpl.java" summary="yes">

  <replacetoken>super();</replacetoken>

  <replacevalue>super() ;
  }

  public TaggedValueImpl(String key, String value) {
        super() ;
        this.setKey(key);
        this.setValue(value);
  </replacevalue>
  </replace>
```

Muster 3 – Constructor-Ergänzung

- Beispiel

```
  <!--
    Add additional constructor.
    Pattern 3) Constructor enhancement
  -->
<replace file="de/gulden/modeling/wave/impl/TaggedValueImpl.java" summary="yes">

  <replacetoken>super();</replacetoken>

  <replacevalue>super() ;
  }

  public TaggedValueImpl(String key, String value) {
        super() ;
        this.setKey(key);
        this.setValue(value);
  </replacevalue>
  </replace>
```

Muster 4 – „einfache“ Refactoring-Operationen

- Umbenennungen
- Ändern von Dateipfaden
- Einfügen konstanter Werte
- Auskommentieren

Muster 4 – „einfache“ Refactoring-Operationen

- Beispiel

```
  <!--
    Use smaller font size for text-labels of relationships.
  Pattern 4) Simple Refactoring
  -->
<replace file="../wave/diagram/edit/parts/DependencyRelationshipStereotypeEditPart.java"
       token="getFontHeight(),"
       value="getFontHeight()-2,"
       summary="yes" />


<!--
  Load icons from bundle 'de.gulden.modeling.wave', instead of 'wave'.
  Pattern 4) Simple Refactoring
-->
<replace file="../wave/diagram/part/WavePaletteFactory.java"
         token="/wave/icons"
         value="/de.gulden.modeling.wave/icons"
         summary="yes" />
```

Muster 4 – „einfache“ Refactoring-Operationen

- Beispiel

```
  <!--
    Use smaller font size for text-labels of relationships.
  Pattern 4) Simple Refactoring
  -->
<replace file="../wave/diagram/edit/parts/DependencyRelationshipStereotypeEditPart.java"
       token="getFontHeight(),"
       value="getFontHeight()-2,"
       summary="yes" />


<!--
  Load icons from bundle 'de.gulden.modeling.wave', instead of 'wave'.
  Pattern 4) Simple Refactoring
-->
<replace file="../wave/diagram/part/WavePaletteFactory.java"
         token="/wave/icons"
         value="/de.gulden.modeling.wave/icons"
         summary="yes" />
```

Muster 4 – „einfache“ Refactoring-Operationen

- Beispiel

```
  <!--
    Use smaller font size for text-labels of relationships.
  Pattern 4) Simple Refactoring
  -->
<replace file="../wave/diagram/edit/parts/DependencyRelationshipStereotypeEditPart.java"
       token="getFontHeight(),"
       value="getFontHeight()-2,"
       summary="yes" />


<!--
  Load icons from bundle 'de.gulden.modeling.wave', instead of 'wave'.
  Pattern 4) Simple Refactoring
-->
<replace file="../wave/diagram/part/WavePaletteFactory.java"
         token="/wave/icons"
         value="/de.gulden.modeling.wave/icons"
         summary="yes" />
```

Kontakt

Jens Gulden

E-Mail: jens.gulden@uni-due.de
Telefon: +49 / 201 / 1834150
Telefax: +49 / 201 / 1834011
Raum: R09 R04 H38

„WAVE" – Web Application Visual Environment:

http://wave.berlios.de/