

# Palladio Fluent API Model Generator

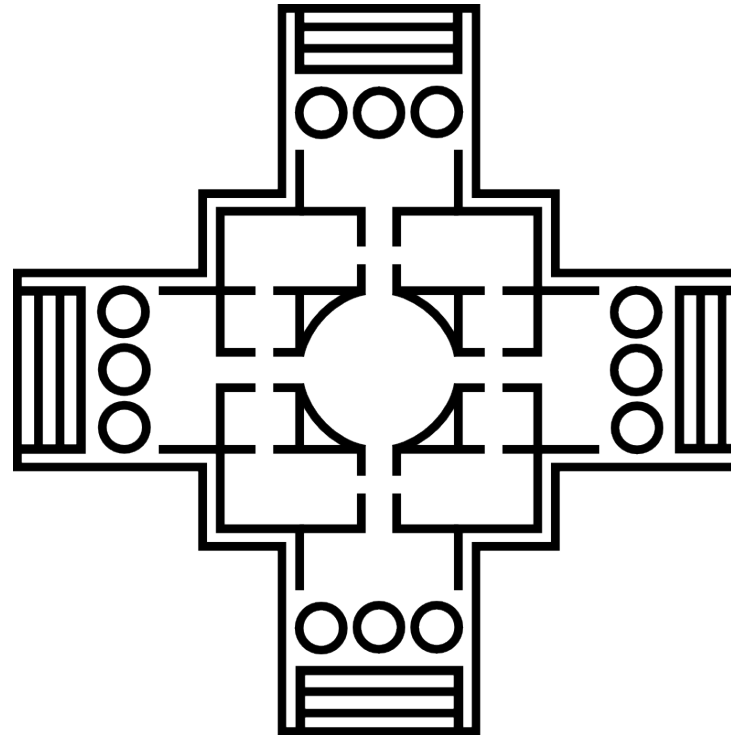
## Praktikum: Werkzeuge für Agile Modellierung

ARCHITECTURE-DRIVEN REQUIREMENTS ENGINEERING,  
INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION, KIT-FAKULTÄT FÜR INFORMATIK



# Inhalt

- Palladio
- Palladio Repository Model
- Fluent Interfaces
- Palladio Fluent API Model Generator



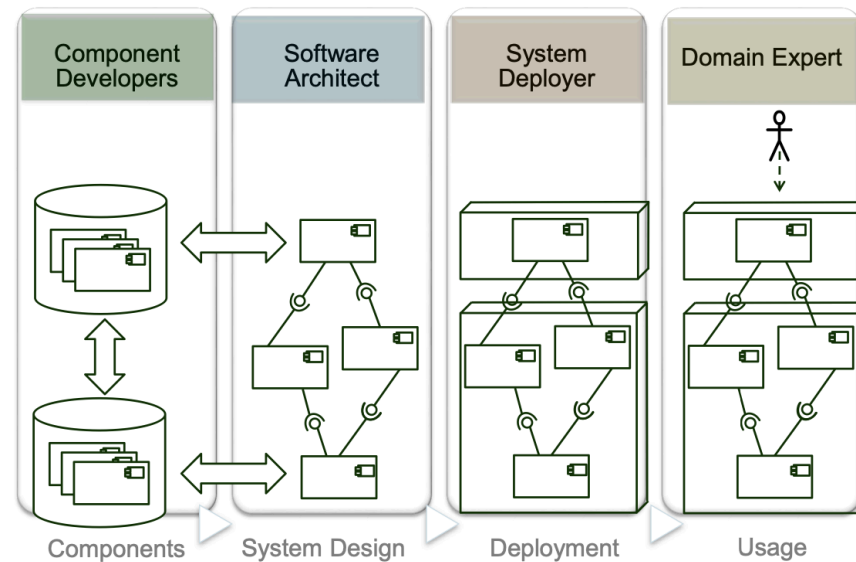
# PALLADIO

# Palladio

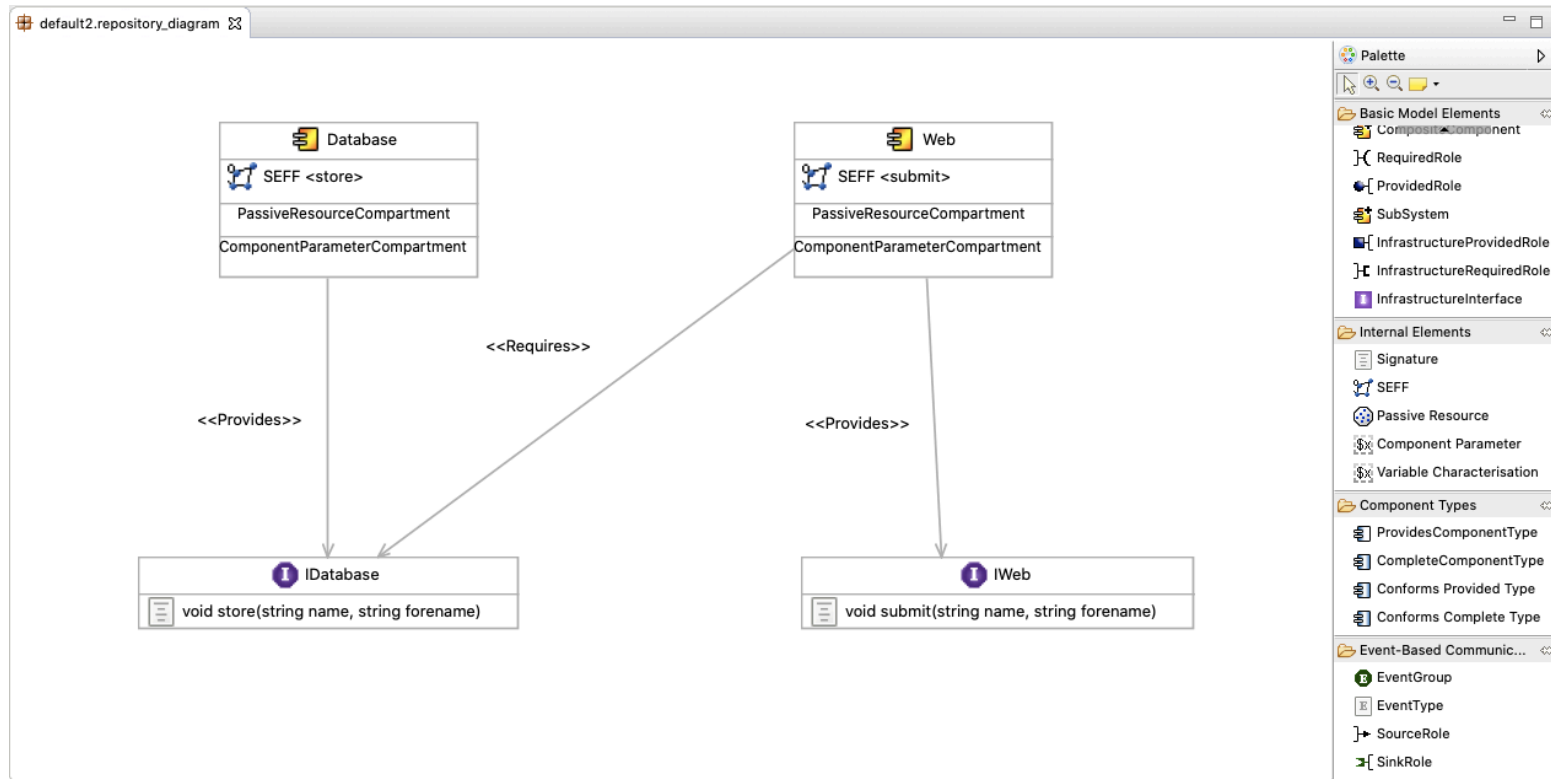
- Tool für die Simulation von Software-Architektur
- Analyse und Vorhersage von qualitativen Eigenschaften der Software bezüglich:
  - Performance Engpässe
  - Skalierbarkeit
  - Verlässlichkeit
  - Wartbarkeit
  - Kosten
- Kern des Palladio Ansatzes: Palladio Component Model (PCM)

# Palladio Component Model (PCM)

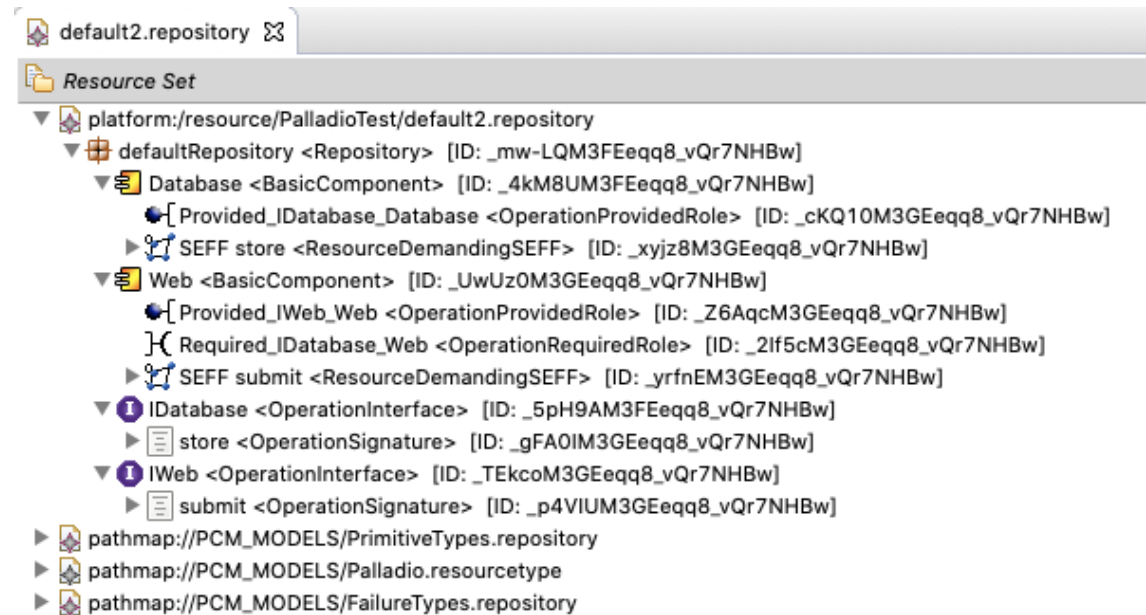
- ist eine Domänenspezifische Modellierungssprache (DSL)
- Erstellung von PCM Instanzen
  - Repository model
  - Usage model
  - System model
  - Resource environment model



# PALLADIO REPOSITORY MODEL



## Erstellung eines Palladio Repository Model mit Hilfe des graphischen Editors



# Palladio Repository Model

Baumansicht





Ein Design Prinzip

# FLUENT INTERFACES

# Fluent Interfaces

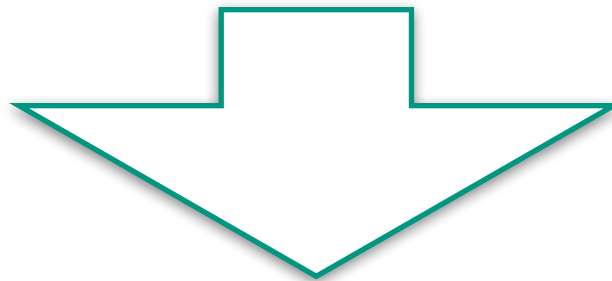
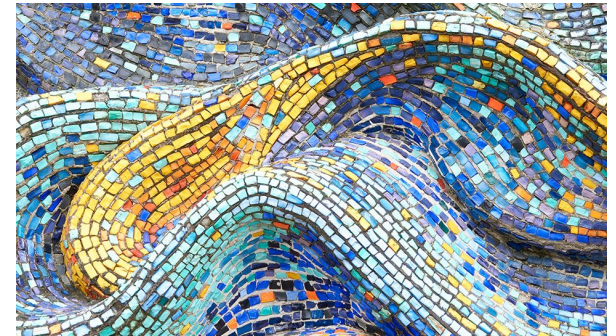
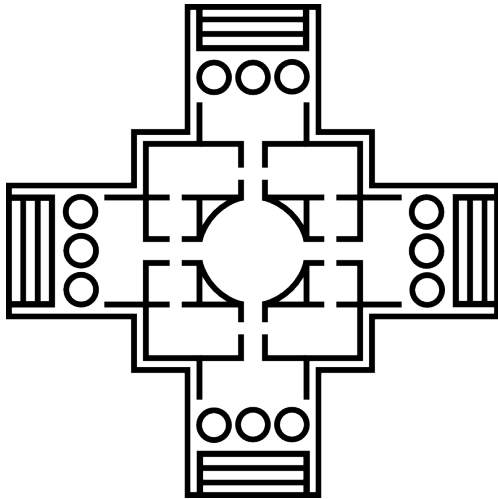
- Besonders nützlich, um Objekte zu erstellen und zu manipulieren
- Ziel: lesbarer Code
- Methodenverkettung
  - die sich wie ein natürlich-sprachlicher Satz liest
- Gibt einen Rahmen vor und bietet eine natürliche Intuition für verfügbare Features
- Beispiele: Java Stream API, JMock

# Fluent Interfaces: Beispiel

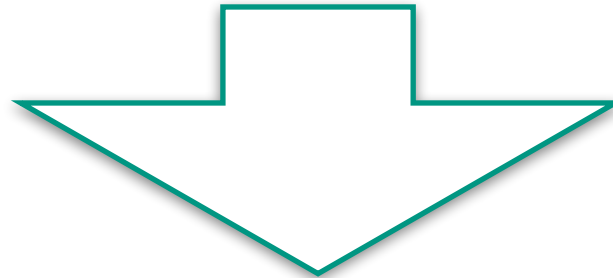
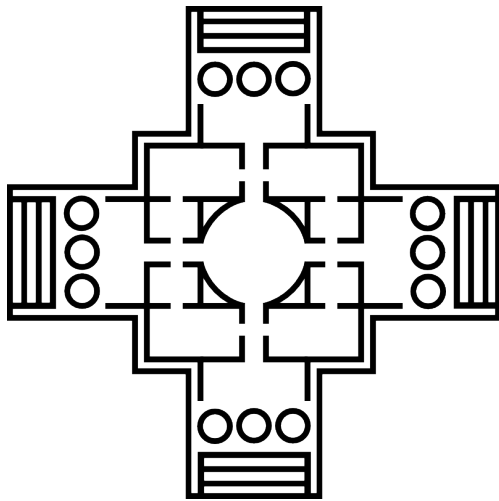
```
private void makeNormal(Customer customer) {  
    Order o1 = new Order();  
    customer.addOrder(o1);  
    OrderLine line1 = new OrderLine(6,  
Product.find("TAL"));  
    o1.addLine(line1);  
    OrderLine line2 = new OrderLine(5,  
Product.find("HPK"));  
    o1.addLine(line2);  
    OrderLine line3 = new OrderLine(3,  
Product.find("LGV"));  
    o1.addLine(line3);  
    line2.setSkippable(true);  
    o1.setRush(true);  
}
```

```
private void makeFluent(Customer customer) {  
    customer.newOrder()  
        .with(6, "TAL")  
        .with(5, "HPK").skippable()  
        .with(3, "LGV")  
        .priorityRush();  
}
```

Quelle: <https://martinfowler.com/bliki/FluentInterface.html> (Martin Fowler)



# PALLADIO FLUENT API MODEL GENERATOR



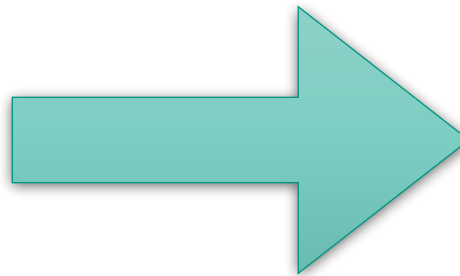
# PALLADIO FLUENT API MODEL GENERATOR

# Recherche

- Fluent Interfaces mit Builder Pattern
  - Erstellen einer Grammatik
- Kleiner Entwurf mit Basic Component

# Herangehensweise

“Create a new repository model. Add to this repository a basic component with name ‘Database’. The basic component ‘Database’ requires an interfaces with name ‘IDatabase’. This interface has a signature called ‘store’ with the parameters name and forename. ...“



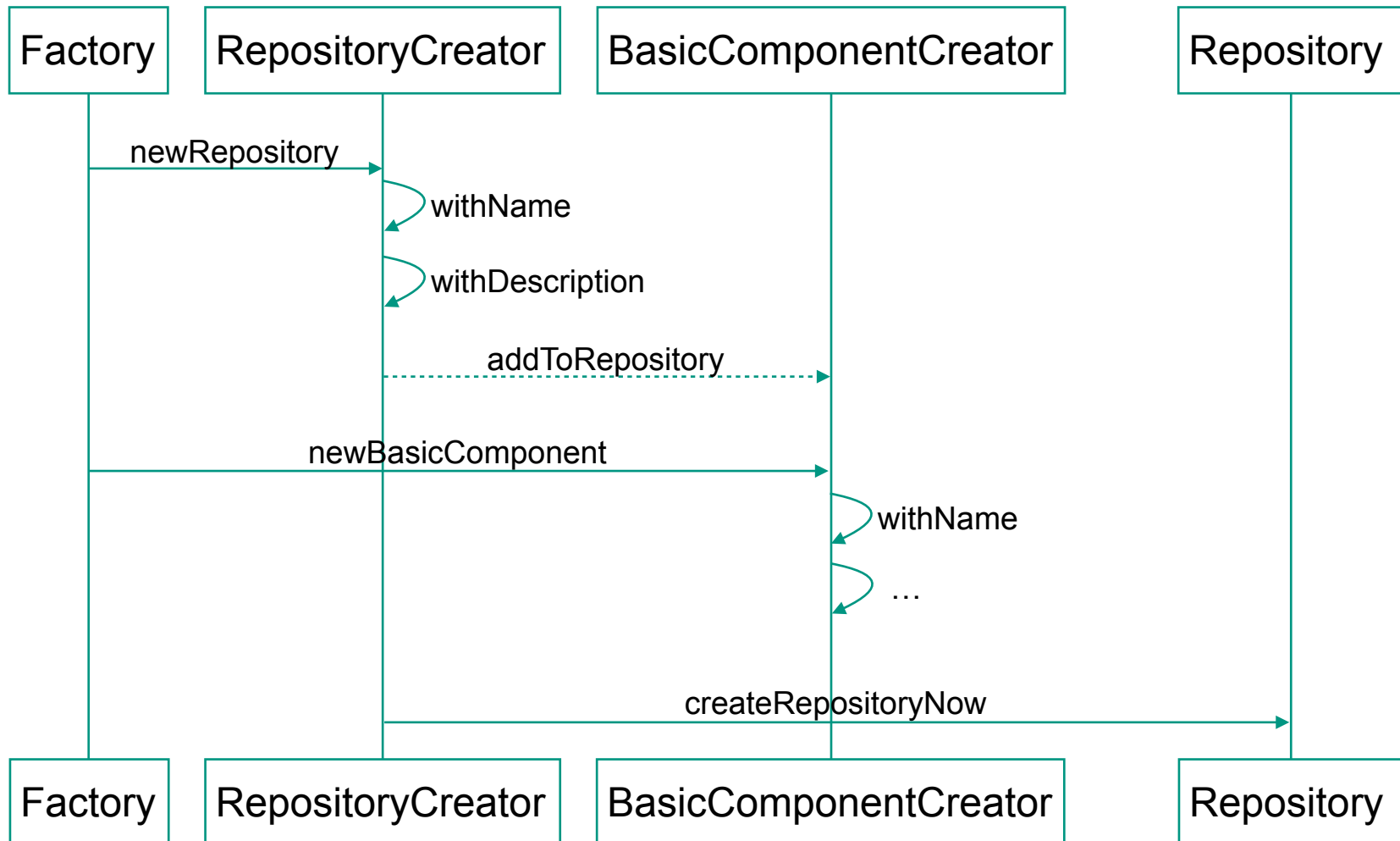
```

185
186 FluentRepositoryFactory create = new FluentRepositoryFactory();
187
188 Repository repository = create.newRepository()
189     .addToRepository(create.newBasicComponent()
190         .withName("Database")
191         .requires(create.newOperationInterface()
192             .withName("IDatabase")
193             .withOperationSignature(create.newOperationSignature()
194                 .withName("store")
195                 .withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
196                 .withParameter("name", Primitive.STRING, ParameterModifier.NONE))))
197     .createRepositoryNow();
198

```



# Fluent Interface Grammatik



# Recherche

- Fluent Interfaces mit Builder Pattern
  - Erstellen einer Grammatik
- Kleiner Entwurf mit Basic Component
- Methodenverkettung vs. Methodenverschachtelung
- PCM Backend
  - Was bietet der graphische Editor?
  - Was bietet die Baumansicht?
  - Was implementiert das Referenzprojekt?
  - Welche Factorys bietet das PCM Backend?
- ➔ Vollständige Implementierung

# Herausforderungen

## Dokumentation des Palladio Projekts

```

EList<Parameter> org.palladiosimulator.pcm.repository.InfrastructureSignature.getParameters__InfrastructureSignature()

```

Returns the value of the '**Parameters Infrastructure Signature**' containment reference list. The list contents are of type [org.palladiosimulator.pcm.repository.Parameter](#). It is bidirectional and its opposite is '[Infrastructure Signature Parameter](#)'.

If the meaning of the '**Parameters Infrastructure Signature**' containment reference list isn't clear, there really should be more of a description here...

**Returns:**

the value of the '**Parameters Infrastructure Signature**' containment reference list.

**See Also:**

[org.palladiosimulator.pcm.repository.RepositoryPackage.getInfrastructureSignature\\_Parameters\\_InfrastructureSignature\(\)](#)  
[org.palladiosimulator.pcm.repository.Parameter.getInfrastructureSignature\\_Parameter](#)

**@model**

opposite="InfrastructureSignature\_Parameter" containment="true" ordered="false"

**@generated**



■ Welche Modell-Elemente gibt es und in welcher Verbindung stehen sie zu anderen Modell-Elementen?

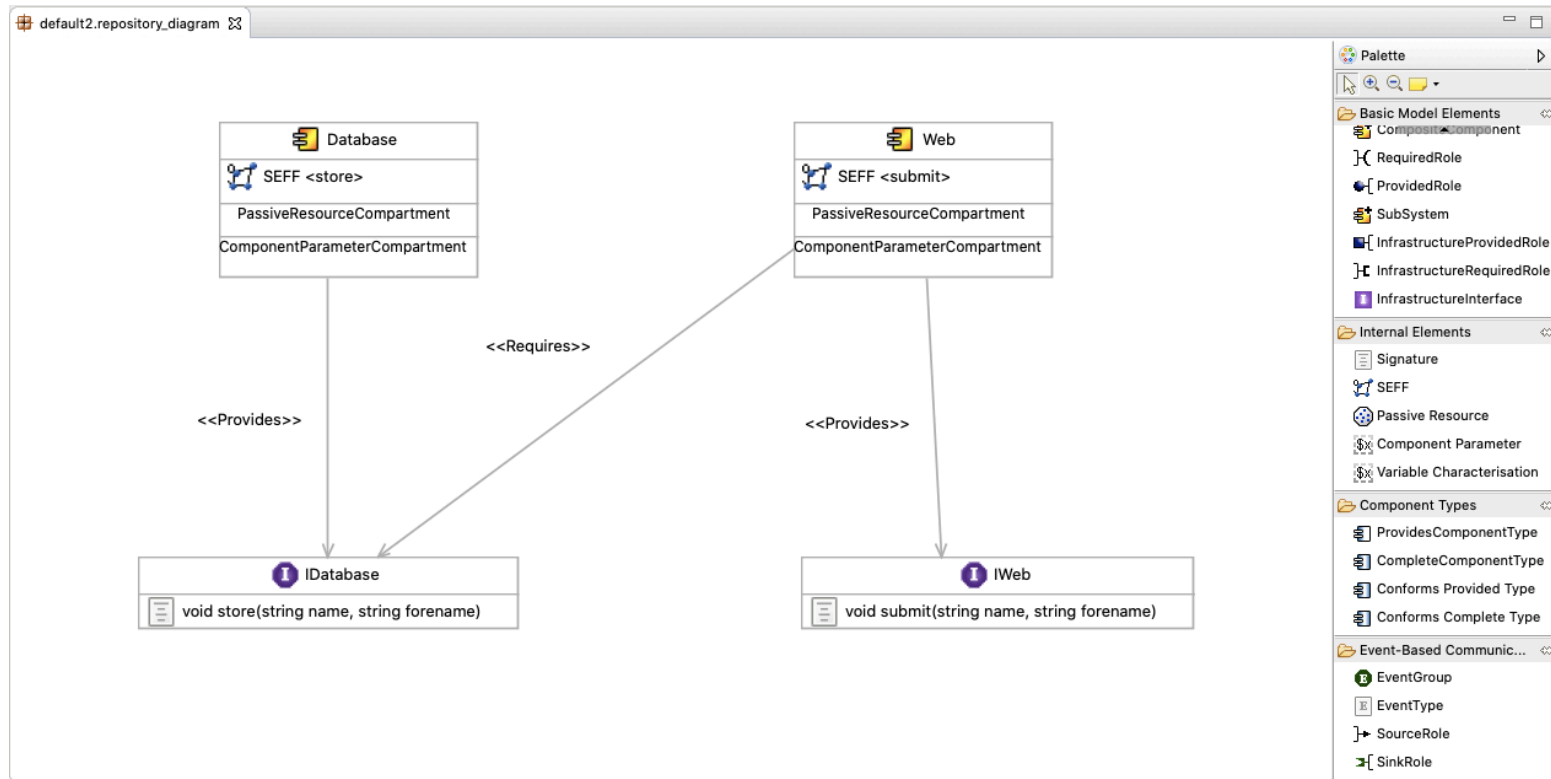
## Design Entscheidungen

- Welche Modell-Elemente sollten über die Factory erstellt werden?
- Wie stark sollte die Reihenfolge der Methodenaufrufe festgelegt werden?
- Sollte man Methoden, die Werte setzen und nicht hinzufügen, überhaupt mehrfach aufrufen können?

# ERGEBNIS

# Aufbau der FluentRepositoryFactory

- Die Factory erstellt
  - das Repository
- und folgende Elemente, die dem Repository hinzugefügt werden
  - Components
  - Interfaces
  - Data types
  - Failure types
- Interne Elemente, die ebenfalls über die Factory erstellt werden
  - SEFFs
  - Variable usages
  - Signatures
- Die Factory ermöglicht den Zugriff auf alle bereits erstellten Entitäten



## Erstellung eines Palladio Repository Model mit Hilfe des graphischen Editors

```

156 public static void exampleFluentAPI() {
157     // Factory
158     FluentRepositoryFactory create = new FluentRepositoryFactory();
159
160     Repository repository = create.newRepository()
161         // Database
162         .addToRepository(create.newOperationInterface().withName("IDatabase")
163             .withOperationSignature(create.newOperationSignature()
164                 .withName("store")
165                 .withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
166                 .withParameter("name", Primitive.STRING, ParameterModifier.NONE)))
167         .addToRepository(create.newBasicComponent().withName("Database")
168             .withServiceEffectSpecification(create.newSeff().onSignature(create.fetchOfSignature("store")))
169             .provides(create.fetchOfOperationInterface("IDatabase")))
170         // Web
171         .addToRepository(create.newOperationInterface().withName("IWeb")
172             .withOperationSignature(create.newOperationSignature()
173                 .withName("submit").withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
174                 .withParameter("name", Primitive.STRING, ParameterModifier.NONE)))
175         .addToRepository(create.newBasicComponent().withName("Web")
176             .withServiceEffectSpecification(create.newSeff().onSignature(create.fetchOfSignature("submit")))
177             .provides(create.fetchOfOperationInterface("IWeb"))
178             .requires(create.fetchOfOperationInterface("IDatabase")))
179         .createRepositoryNow();
180
181     saveRepository(repository, "./", "fluentAPIExample.repository", false);
182 }

```

## Ergebnis

Beispiel: Web und Database Komponenten mit ihren Schnittstellen

# Ergebnis: Palladio Fluent API Model Generator

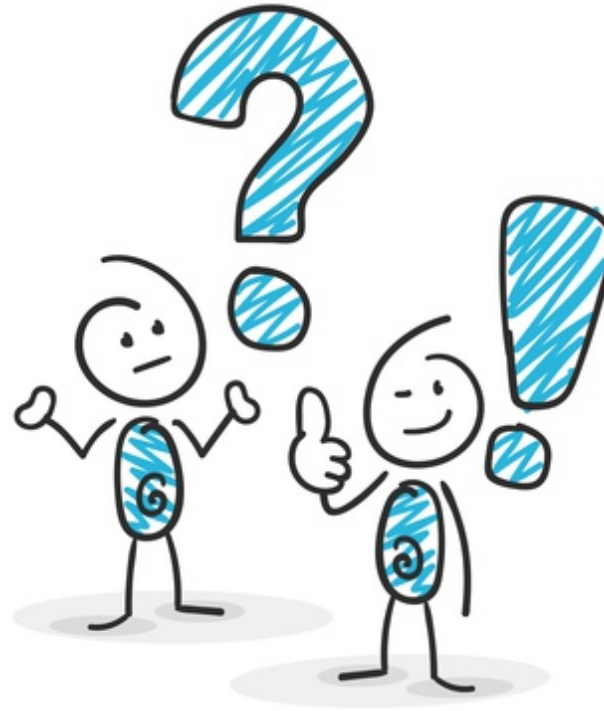
- Fluent API
  - zur Erstellung von Palladio Repository Objekten
  - mit nur einer Schnittstelle/Factory
  - mit einem klaren Rahmen, welche Elemente, wo hinzugefügt und/oder verbunden werden können
  - => auch für Einsteiger geeignet
- Vollständige Dokumentation (JavaDoc)
- Umfassende Projektbeschreibung und Einführung mit Beispielen (Readme)



```

80 public static void readmeExampleBackend() {
81     // Factory
82     RepositoryFactory repoFact = RepositoryFactory.eINSTANCE;
83     // Repository
84     Repository repository = repoFact.createRepository();
85
86     // Database component
87     BasicComponent databaseComponent = repoFact.createBasicComponent();
88     databaseComponent.setEntityName("Database");
89
90     // IDatabase interface
91     OperationInterface databaseInterface = repoFact.createOperationInterface();
92     databaseInterface.setEntityName("IDatabase");
93
94     // Signature store
95     OperationSignature store = repoFact.createOperationSignature();
96     store.setEntityName("store");
97     // with parameters forename, name
98     Parameter forename = repoFact.createParameter();
99     forename.setParameterName("forename");
100    forename.setDataType__Parameter(null); // referencing the imported data types poses another problem
101    Parameter name = repoFact.createParameter();
102    name.setParameterName("forename");
103    name.setDataType__Parameter(null);
104
105    // Providing connection from Database component to IDatabase interface
106    OperationProvidedRole dbProvIDb = repoFact.createOperationProvidedRole();
107    dbProvIDb.setProvidedInterface__OperationProvidedRole(databaseInterface);
108    dbProvIDb.setProvidingEntity__ProvidedRole(databaseComponent);
109
110    // Seff for Database component on service store
111    ResourceDemandingSEFF storeSeff = SeffFactory.eINSTANCE.createResourceDemandingSEFF();
112    storeSeff.setDescribedService__SEFF(store);
113    databaseComponent.getServiceEffectSpecifications__BasicComponent().add(storeSeff);
114
115    // Adding component + interfaces to the repository
116    repository.getComponents__Repository().add(databaseComponent);
117    repository.getInterfaces__Repository().add(databaseInterface);
118
119    // Web component
120    BasicComponent webComponent = repoFact.createBasicComponent();
121    databaseComponent.setEntityName("Web");
122
123    OperationInterface webInterface = repoFact.createOperationInterface();
124    databaseInterface.setEntityName("IWeb");
125
126    OperationSignature submit = repoFact.createOperationSignature();
127    submit.setEntityName("submit");
128    // with parameters forename, name
129    Parameter forename2 = repoFact.createParameter();
130    forename2.setParameterName("forename");
131    forename2.setDataType__Parameter(null);
132    Parameter name2 = repoFact.createParameter();
133    name2.setParameterName("forename");
134    name2.setDataType__Parameter(null);
135
136    OperationProvidedRole webProvIweb = repoFact.createOperationProvidedRole();
137    webProvIweb.setProvidedInterface__OperationProvidedRole(webInterface);

```



# FRAGEN?