



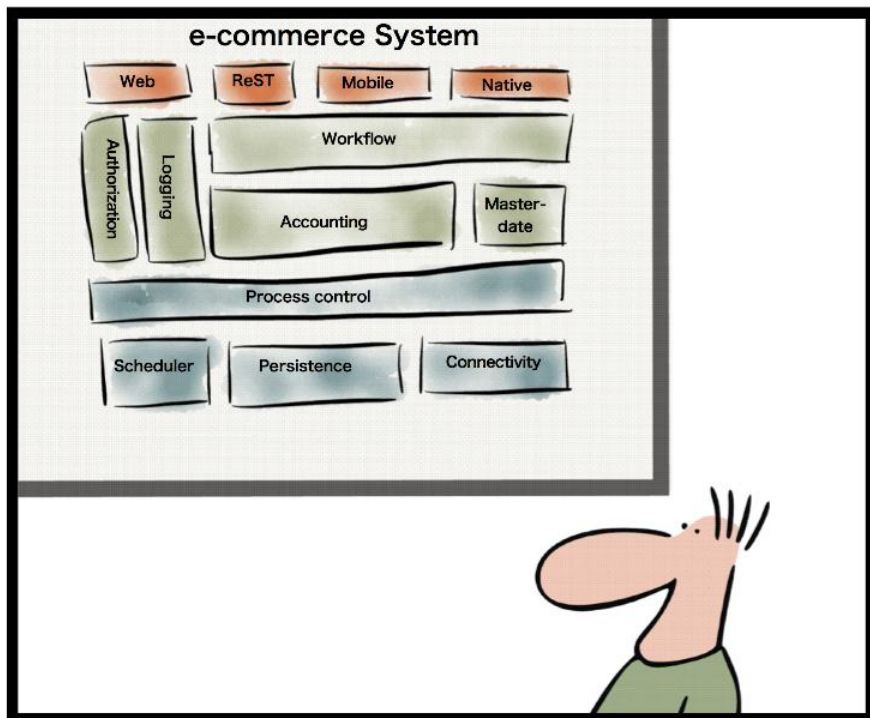
# Continuous Architecture Management

Erkennen und Verhindern von struktureller Erosion

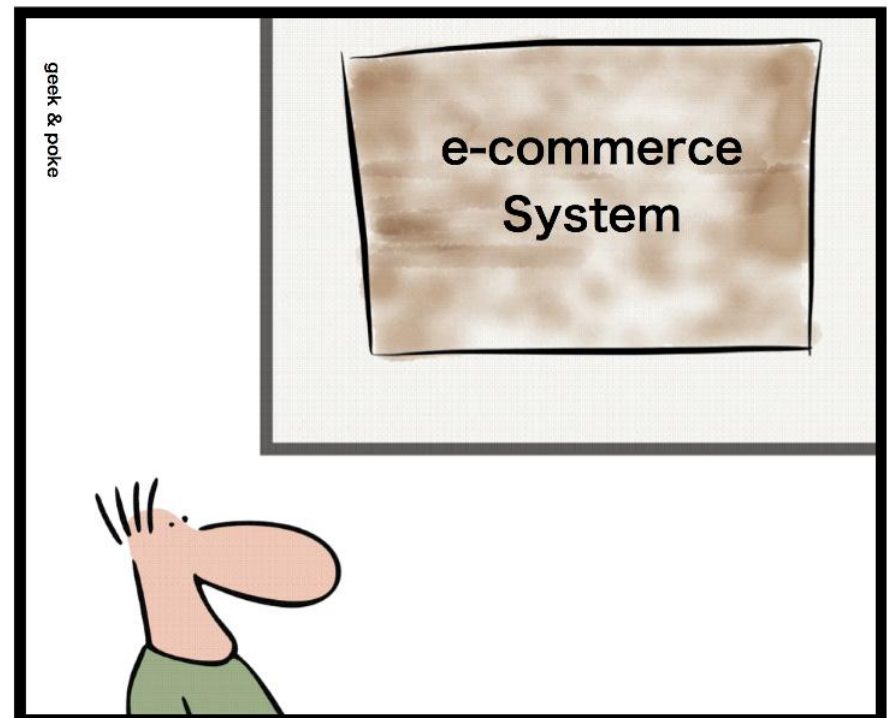
Ingmar Kellner  
hello2morrow GmbH  
April 2012



## “How to draw the architecture of your system”



RULE 1: JUST MAKE IT NICE



RULE 2: AND NOT REALISTIC!!!



# Agenda

- ▶ Motivation
- ▶ Theoretischer Hintergrund
- ▶ Entwicklungsprozess
- ▶ Demonstration von Sonargraph Architect



# Motivation

Als Software Architekt will man ...

- ▶ ... dass die Software eine hohe Qualität hat
- ▶ ... wissen, wie die interne Struktur wirklich aussieht
- ▶ ... strukturelle Erosion verhindern
- ▶ ... wissen, wo aktuell die Problemstellen sind
- ▶ ... existierende Software Module wiederverwenden
- ▶ ... eine aktuelle Dokumentation der Architektur

... Spaß haben bei der Arbeit!



## Widerstände

- ▶ Zeitdruck...
- ▶ Was bedeutet gute Qualität?
- ▶ Es ist zu kompliziert Metriken zu berechnen und sinnvolle Schwellwerte zu definieren
- ▶ Es ist aufwändig, die Architekturdokumentation mit der Entwicklung abzugleichen
- ▶ Die Abhängigkeiten zwischen Teilen der Software manuell zu überprüfen ist nicht machbar
- ▶ Fehlende Toolunterstützung, z.B. zeigt die IDE keine Warnung an, wenn eine unerlaubte Abhängigkeit eingebaut wird.



Und das ist oft die Folge:



HELLO2MORROW



## Einige Gründe für die strukturelle Erosion

- ▶ Wissen und Fähigkeiten im Team sind ungleich verteilt
- ▶ Kopplung und Komplexität wachsen schnell
- ▶ Vielfach gibt es keine formale Architektur
- ▶ In den meisten Projekten wird die Qualität am Schluss betrachtet
- ▶ Desinteresse an Qualität: Management betrachtet die Software als “black box”
- ▶ Das Gesetz der Software Entropy
  - ▶ Eine Software, die benutzt wird, wird verändert werden.
  - ▶ Wenn eine Software verändert wird, steigt die Komplexität, es sei denn, man arbeitet aktiv dagegen.



## Man weiß, dass man ein Problem hat, wenn...

- ▶ ... das System schwierig ist anzupassen, weil jede Änderung viele andere Änderungen nach sich zieht. [Rigidity]
- ▶ ... Änderungen zu Fehlern in konzeptionell verschiedenen Stellen führen. [Fragility]
- ▶ ... es schwierig ist, das System in wiederverwendbare Komponenten zu unterteilen. [Immobility]
- ▶ ... es schwieriger ist etwas richtig zu machen als falsch. [Viscosity]
- ▶ ... der Source Code schwierig zu verstehen ist. [Opacity]

*“The software starts to rot like a bad piece of meat”  
[Robert C. Martin in ASD]*





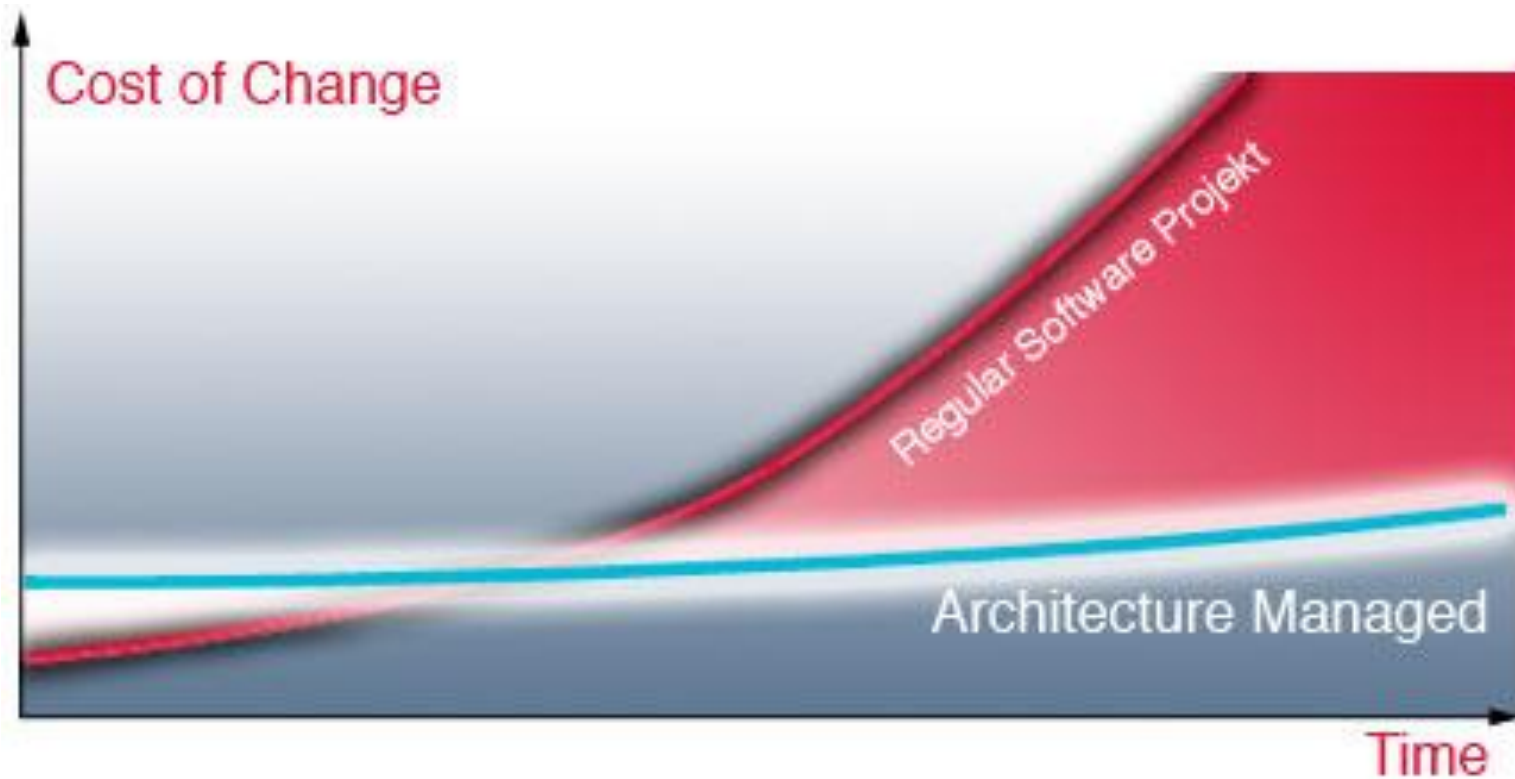
# Was ist technische Qualität?

Unsere Definition: “Technical quality of software can be defined as the level of conformance of a software system to a set a set of rules and guidelines derived from common sense and best practices. Those rules should cover software architecture, programming in general, testing and coding style.”

- ▶ Technische Qualität lässt sich nicht allein durch Testen erreichen
- ▶ Technische Qualität manifestiert sich in jeder Codezeile
- ▶ Vier Aspekte technischer Qualität:
  - ▶ Architektur und Abhängigkeitsstruktur
  - ▶ Software Metriken
  - ▶ Programmierregeln
  - ▶ Testbarkeit und Testabdeckung
- ▶ Welcher dieser Aspekte hat den größten Einfluss auf die Gesamtkosten?
- ▶ Messung erfolgt über Metriken und Zählung von Regelverletzungen



## Kosten von struktureller Erosion / Technical Debt





- ▶ Motivation
- ▶ Theoretischer Hintergrund
- ▶ Entwicklungsprozess
- ▶ Demonstration von Sonargraph Architect



## Sichtbares Verhalten und interne Struktur

- ▶ Das sichtbare Verhalten wäre ausreichend zu verifizieren, wenn
  - ▶ sich Anforderungen nicht ändern würden
  - ▶ Software fehlerfrei wäre
  - ▶ Technologien sich nicht ändern würden

Aber...

- ▶ Software muss angepasst, erweitert und korrigiert werden! Die “maintenance” Phase ist für ca. 90% aller Entwicklungskosten verantwortlich. [The Mythical Man Month]
- ▶ Der Aufwand für die “maintenance” ist hauptsächlich bestimmt durch die technische Qualität und die interne Struktur
- ▶ Software muss normalerweise eine gewisse Zeit erfolgreich eingesetzt werden, um sich zu amortisieren.



# Makro Ebene: Bsp einer einfachen “relaxed layered architecture”



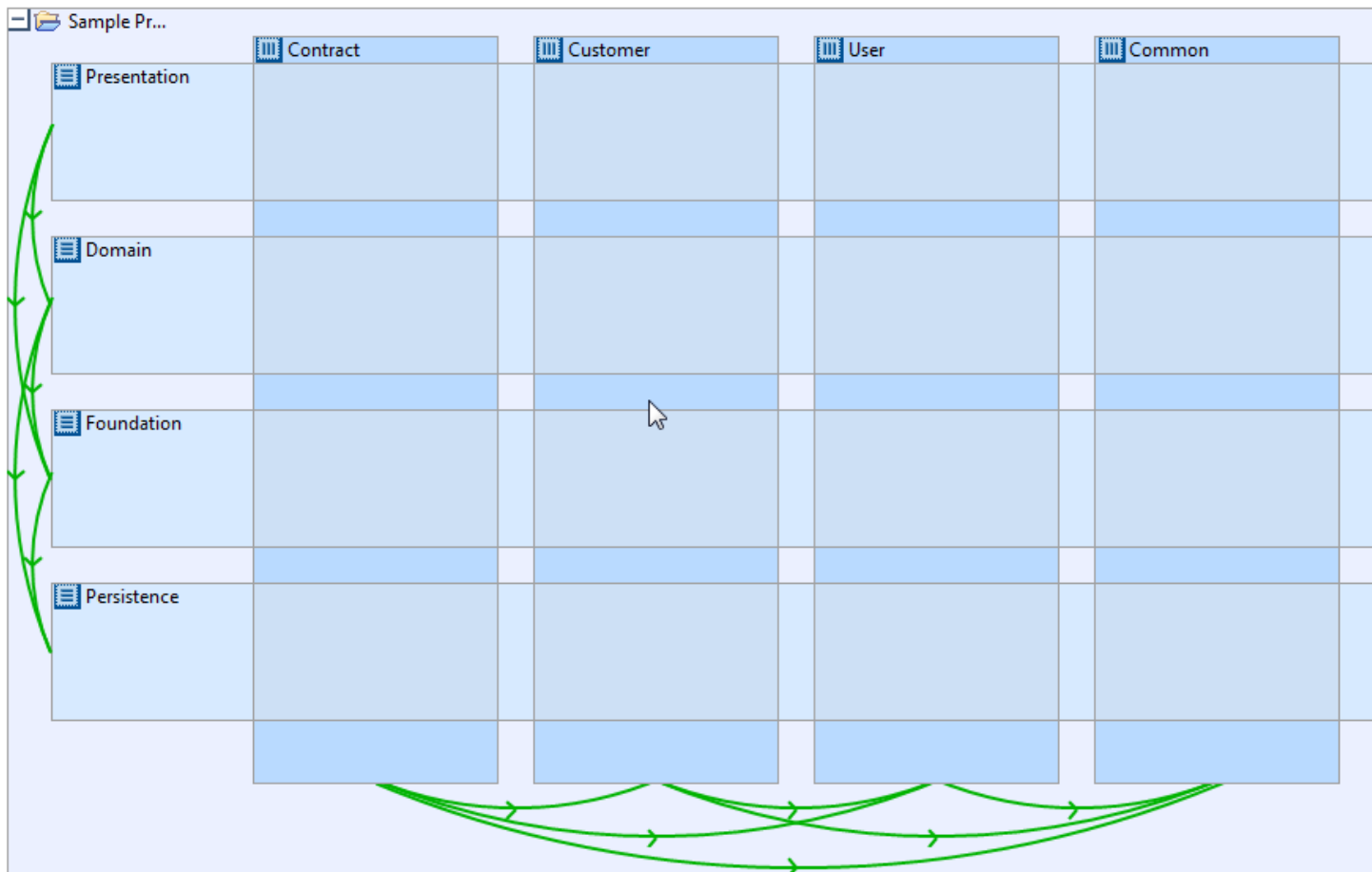


## Was ist mit fachlichen Aspekten?

- ▶ Technische Aspekte sind die Grundlage für die Geschäftslogik
- ▶ Fachliche Aspekte sind orthogonal zu technischen Schichten
- ▶ Auch fachliche Aspekte müssen klare Abhängigkeiten aufweisen

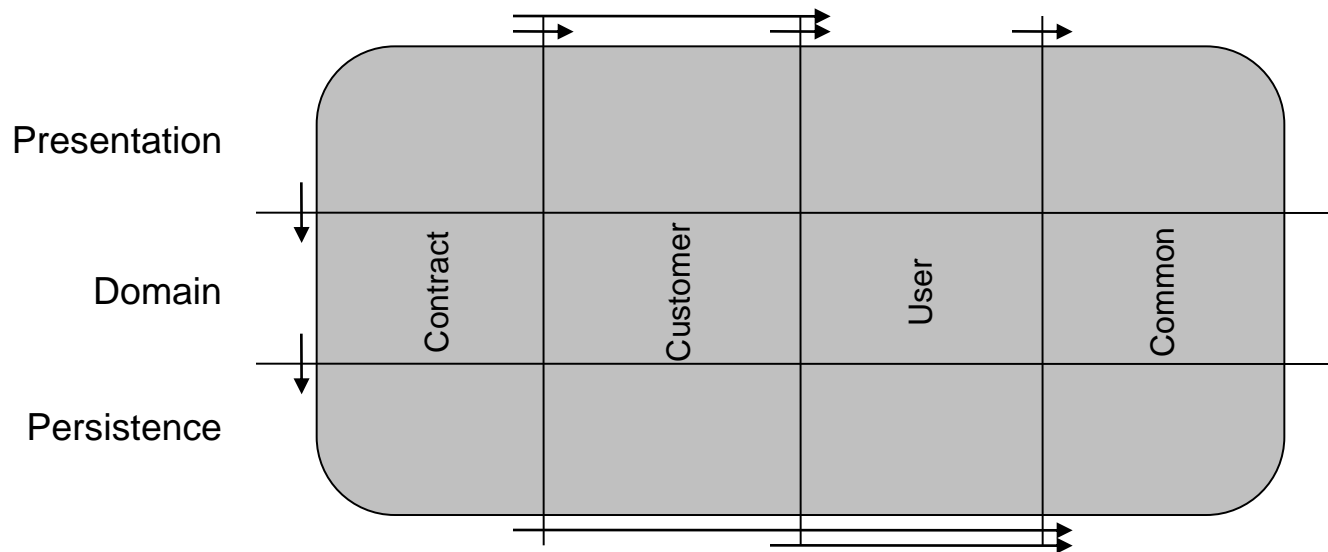


## Eine logische Architektur mit fachlichen Schnitten





## Erstellen einer logischen Architektur



- Schritt 1: Teile horizontal in technische Aspekte
- Schritt 2: Teile vertikal in fachliche Aspekte
- Schritt 3: Definiere erlaubte Abhängigkeiten





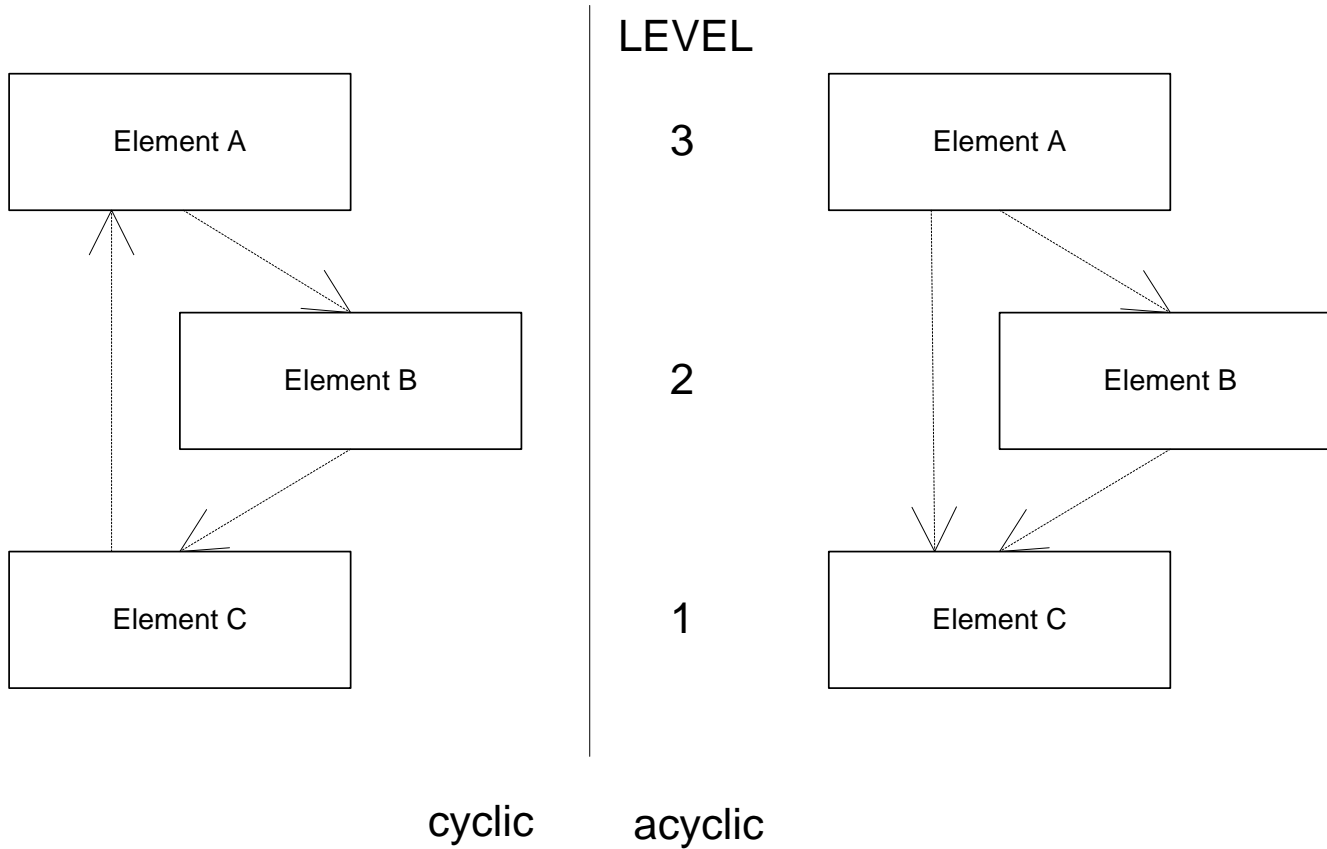
## Der Source Code implementiert die logische Architektur

- ▶ Entwickler setzen use cases um ...
  - ▶ ..., indem sie Klassen und Interfaces in Paketen anlegen
  - ▶ ... und berücksichtigen die vorhandenen Architektur-Restriktionen
  - ▶ ... und ordnen über eine Namenskonvention die physikalischen Artefakte in die logische Architektur ein.
- ▶ Der Source Code ist nicht isoliert
- ▶ Es ist keine Magie involviert – es gibt nichts Expliziteres als den Source Code

“The software is the design” [Robert C. Martin in ASD]



# Abhängigkeiten und Level



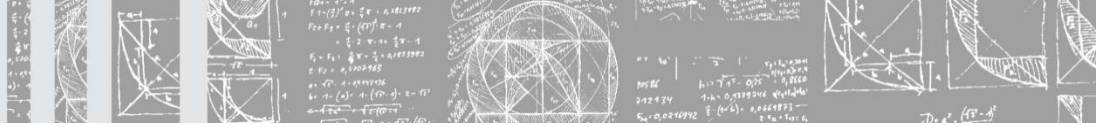


## Der Effekt von zyklischen Abhängigkeiten

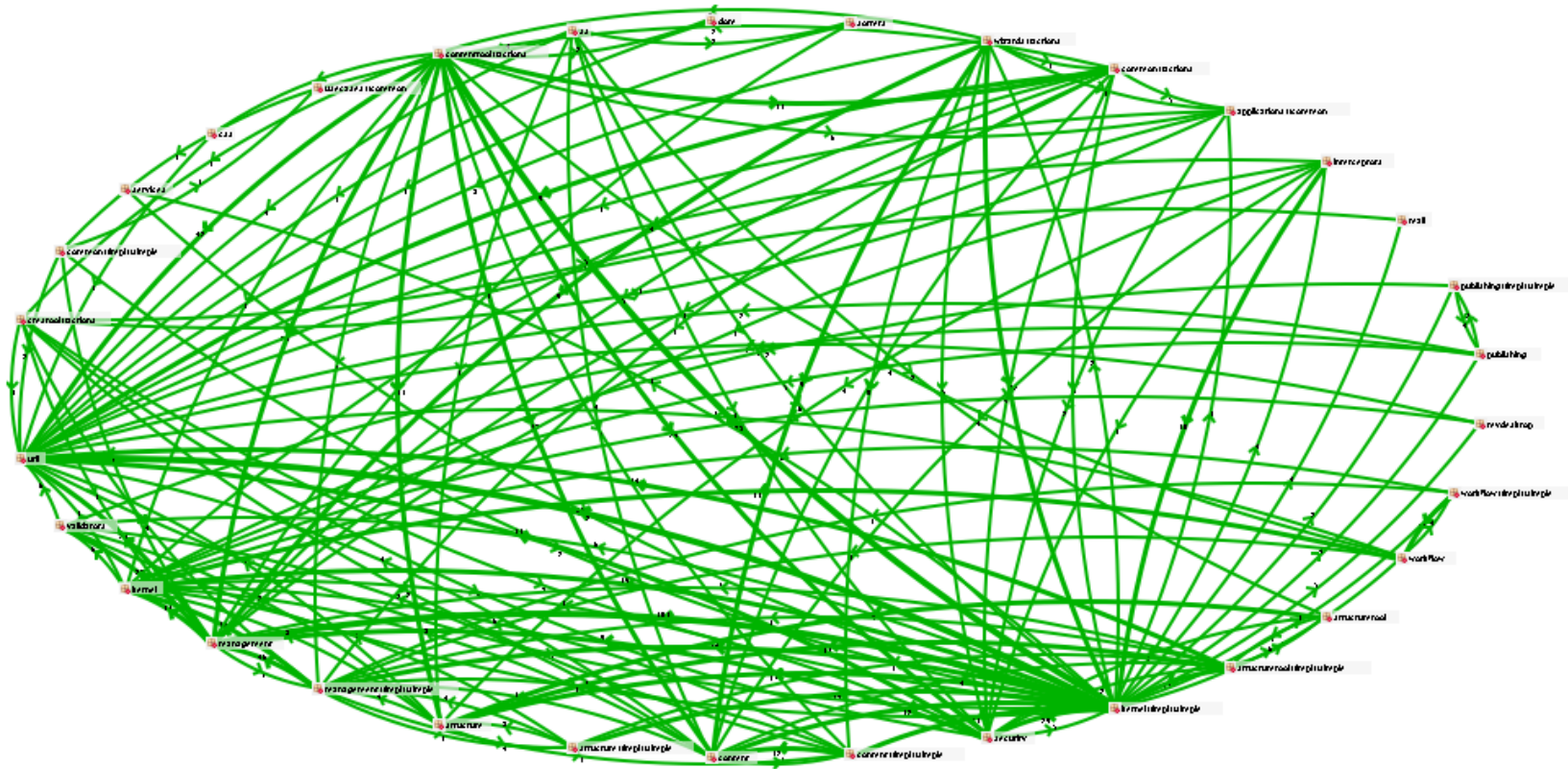
Zyklische Abhängigkeiten haben einen negativen Einfluss auf:

- ▶ Testen
- ▶ Verständlichkeit
- ▶ Wiederverwendung
- ▶ Erweiterbarkeit
- ▶ Team building

“Cyclic physical dependencies in large, low-level subsystems have the greatest capacity to increase the overall cost of maintaining a system“ [John Lakos in LSD]

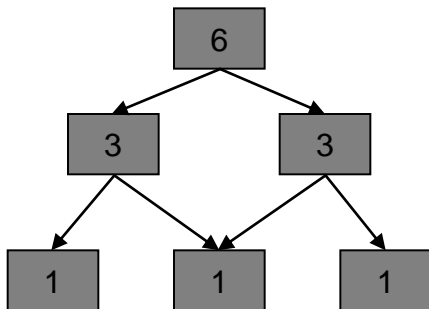


# Beispiel für eine Zyklengruppe

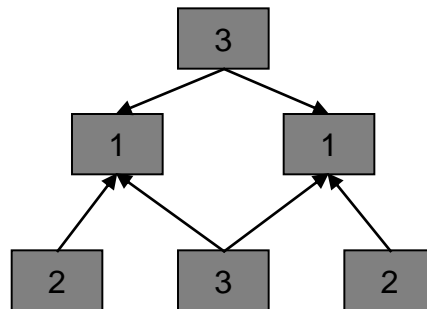


## Berechnung des Kopplungsgrades [LSD]

- ▶ Depends upon = Die Anzahl von Komponenten, von der eine Komponente abhängt (+1 für sich selbst).  
In Java gilt Source File == Komponente.
- ▶ ACD (Average Component Dependency) = Die Summe aller “depends upon” Werte, geteilt durch die Anzahl aller Komponenten

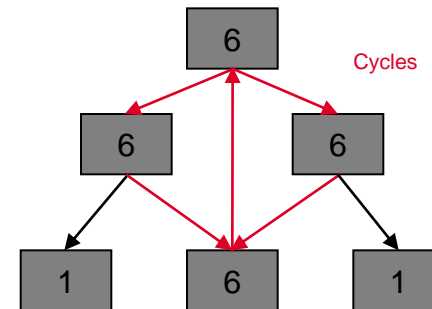


$$\text{ACD} = 15/6 = 2,5$$



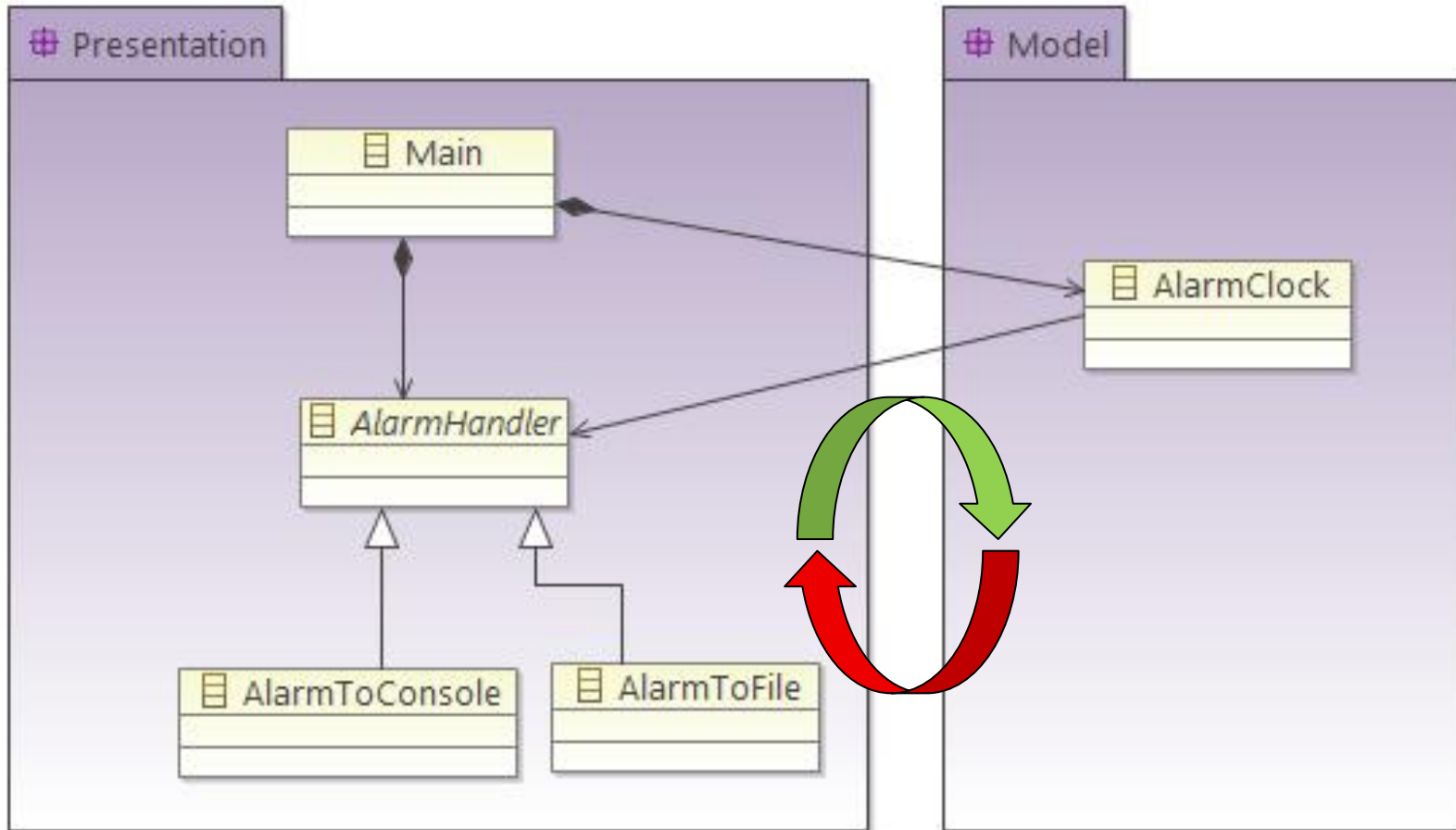
Dependency Inversion  

$$\text{ACD} = 12/6 = 2$$

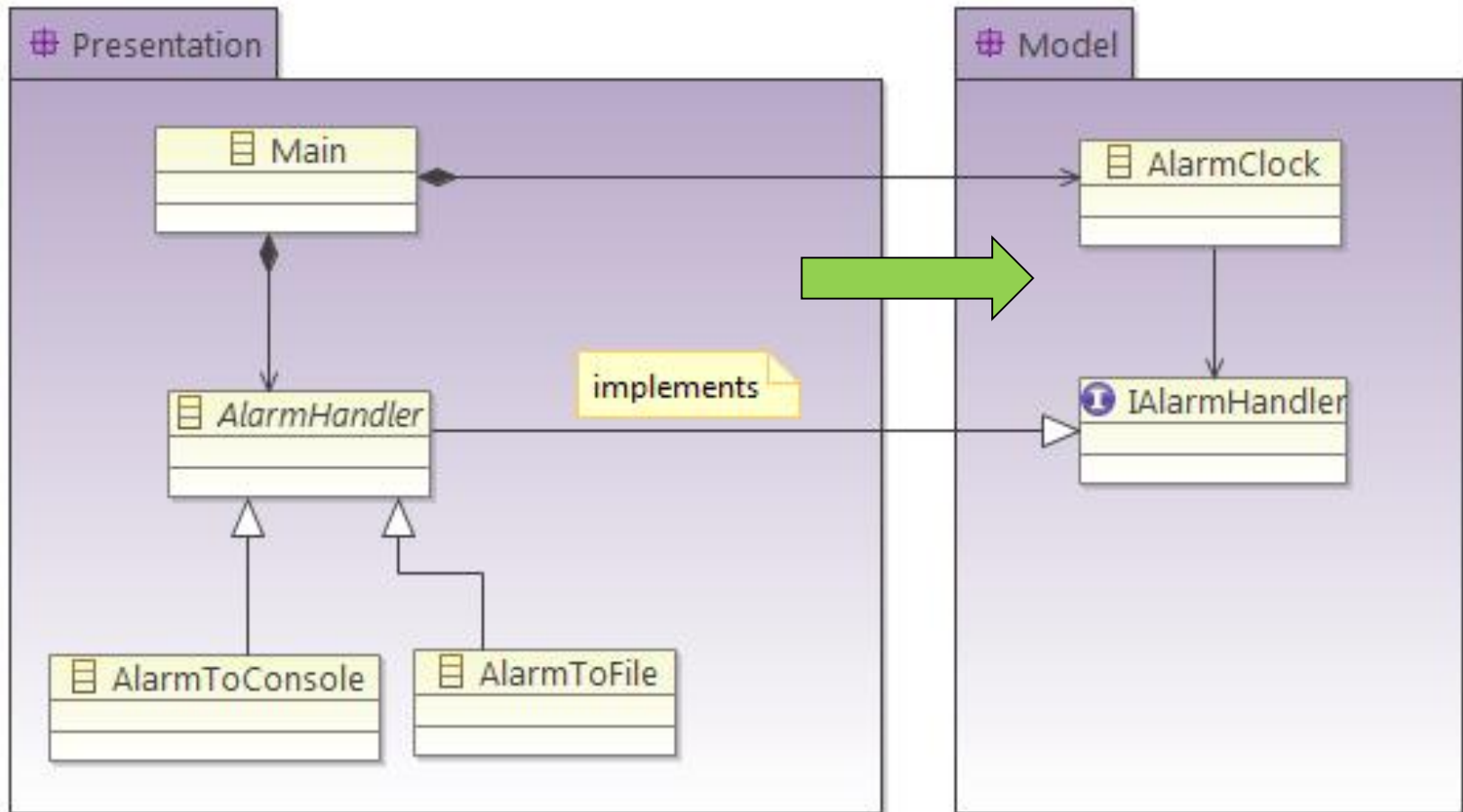


$$\text{ACD} = 26/6 = 4,33$$

## Bsp für das Auflösen eines Zyklus



# Umkehrung der Abhängigkeit durch ein Callback Interface





## Regeln für die Mikro Ebene

- ▶ Methoden implementieren Verhalten
  - ▶ Beschränkung der Komplexität (Cyclomatic Complexity)
- ▶ Typen gruppieren Methoden
  - ▶ Typen sollten klare Abstraktionen repräsentieren
  - ▶ Typen sollten klare Verantwortlichkeiten haben
- ▶ Compilation units / Komponenten / Typen
  - ▶ Zyklen sollten vermieden werden
  - ▶ Beschränkung der Größe
  - ▶ Kopplungsgrad im Auge behalten
- ▶ Packages gruppieren compilation units
  - ▶ Zyklen sollten verboten sein
  - ▶ Auch packages sollten klare Verantwortlichkeiten haben
  - ▶ Beschränkung der Anzahl enthaltener compilation units





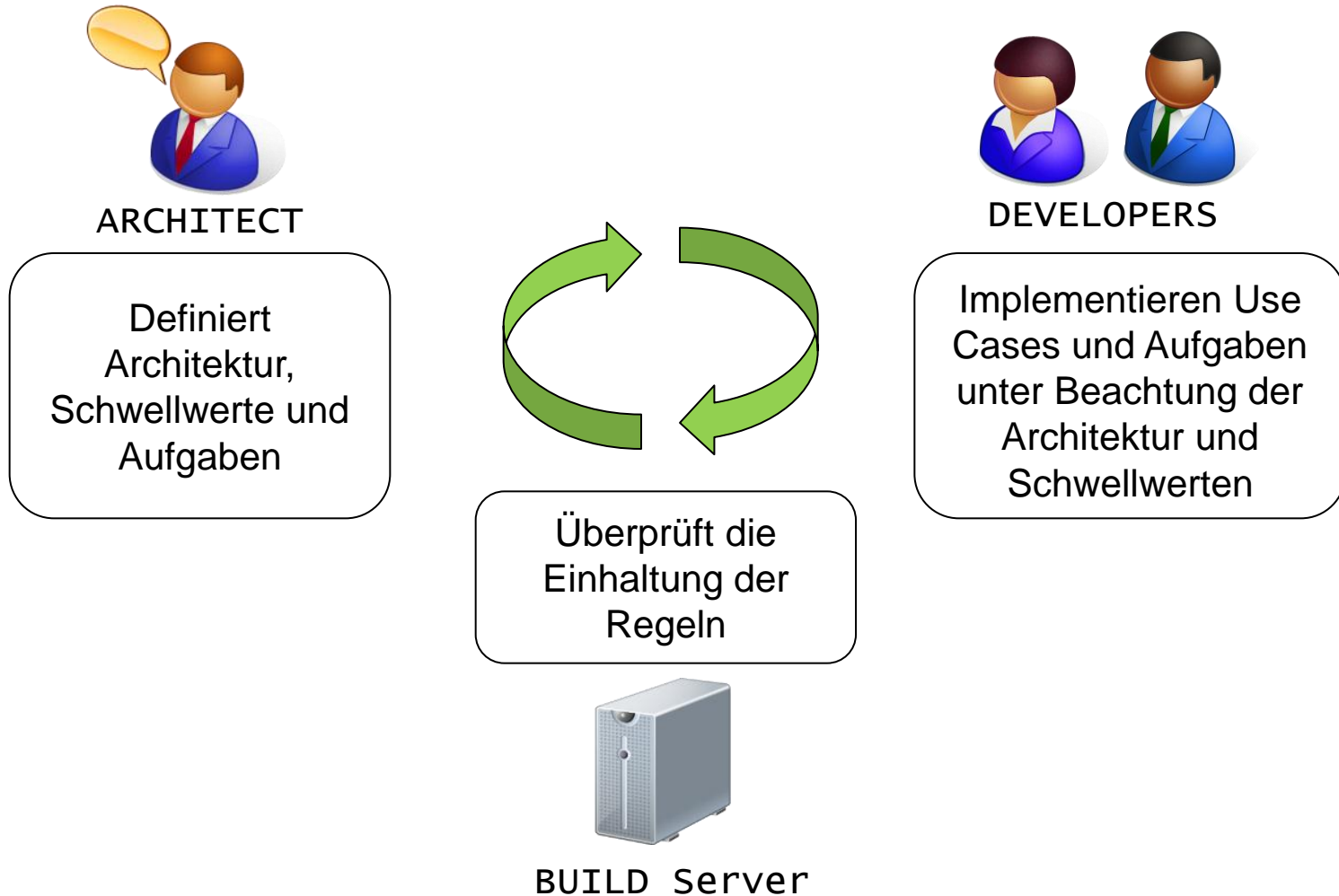
- ▶ Motivation
- ▶ Theoretischer Hintergrund
- ▶ Entwicklungsprozess
- ▶ Demonstration von Sonargraph Architect



## Workflow

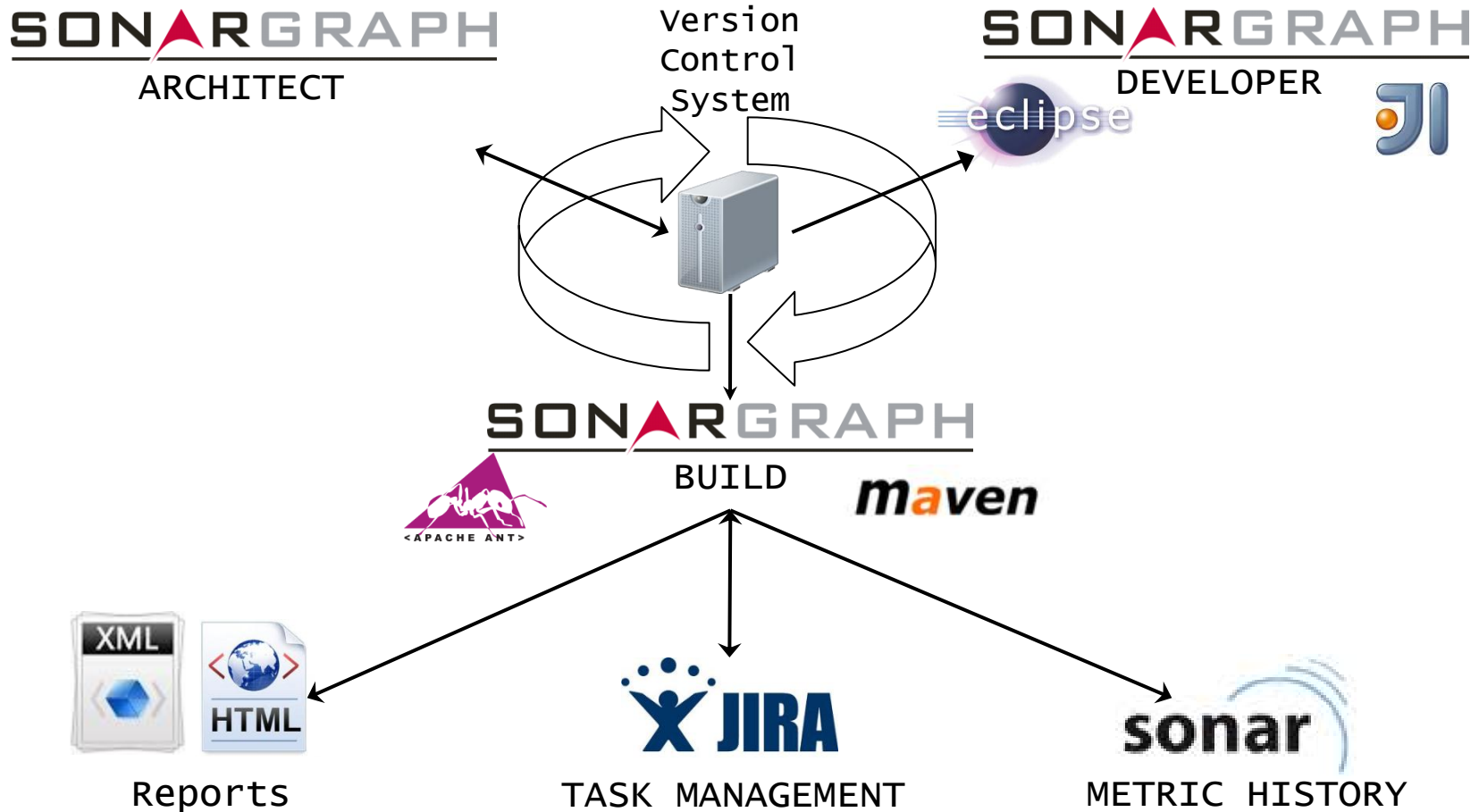
- ▶ Ein Architektur Team etabliert die strukturellen Regeln.
- ▶ Aufgaben können definiert werden, basierend auf vorgeschlagenen Refactorings oder erkannten Verletzungen.
- ▶ Entwickler können leicht die Einhaltung der Regeln in ihrer IDE überprüfen.
- ▶ Entwickler können leicht ihnen zugeordnete Aufgaben erkennen.
- ▶ Das Architekturteam bekommt Rückmeldung zu den gelösten Aufgaben.
- ▶ Die Qualität (== Einhaltung der Regeln) wird kontinuierlich überprüft.

## Architektur Workflow





# Architecture Workflow with Sonargraph



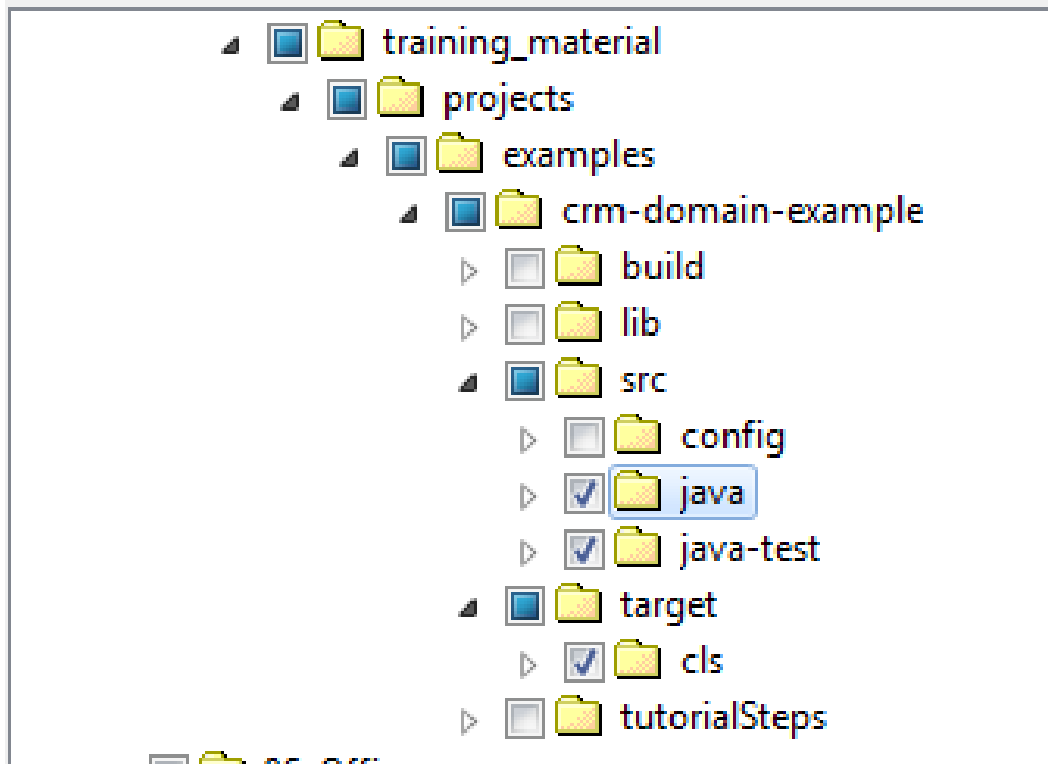
HELLO2MORROW



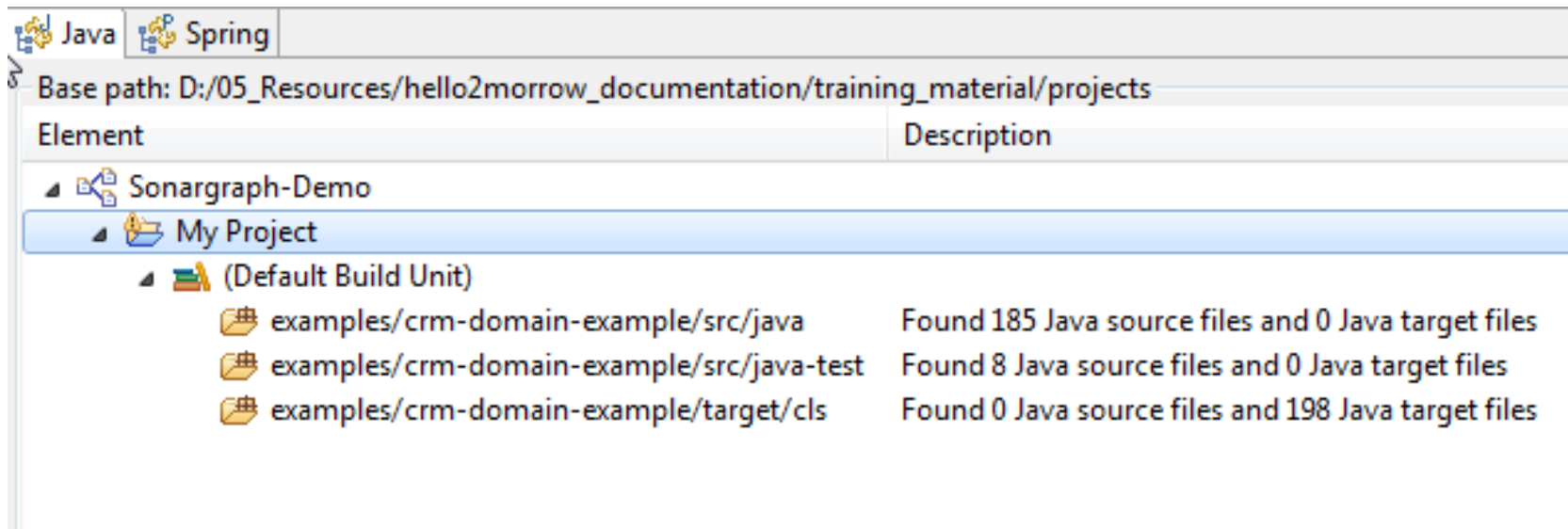
- ▶ Motivation
- ▶ Theoretischer Hintergrund
- ▶ Entwicklungsprozess
- ▶ Demonstration von Sonargraph Architect



## Step 1: Define the workspace



## Step 1: Status after initial workspace setup



The screenshot shows an IDE interface with a project structure tree on the left and a table of analysis results on the right. The project is named 'Sonargraph-Demo' and contains a sub-project 'My Project'. Under 'My Project', there is a '(Default Build Unit)' folder containing three sub-folders: 'examples/crm-domain-example/src/java', 'examples/crm-domain-example/src/java-test', and 'examples/crm-domain-example/target/cls'. The table on the right provides the following analysis results:

Element	Description
examples/crm-domain-example/src/java	Found 185 Java source files and 0 Java target files
examples/crm-domain-example/src/java-test	Found 8 Java source files and 0 Java target files
examples/crm-domain-example/target/cls	Found 0 Java source files and 198 Java target files

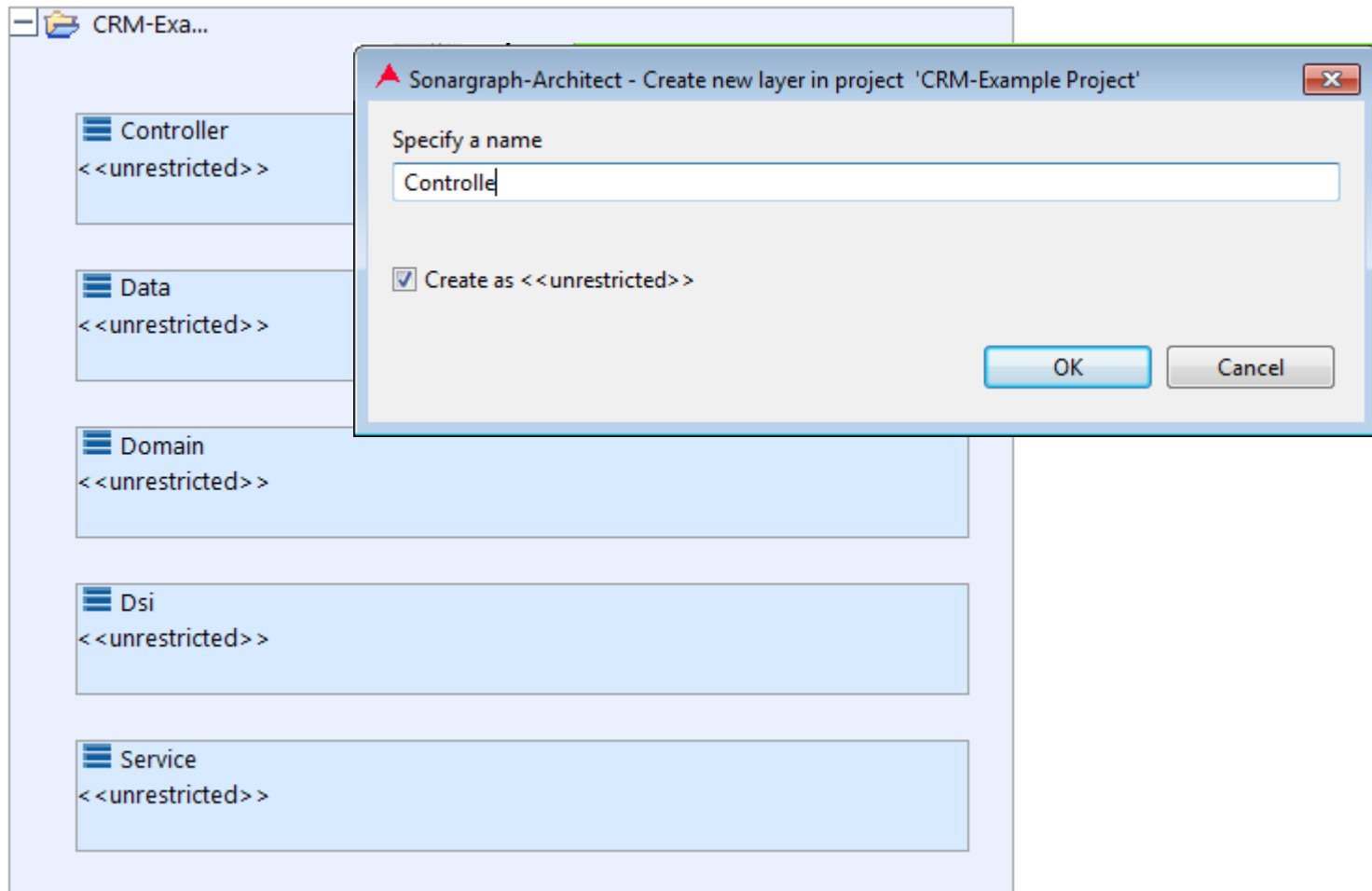


## Step 2: Definition of Layer Groups





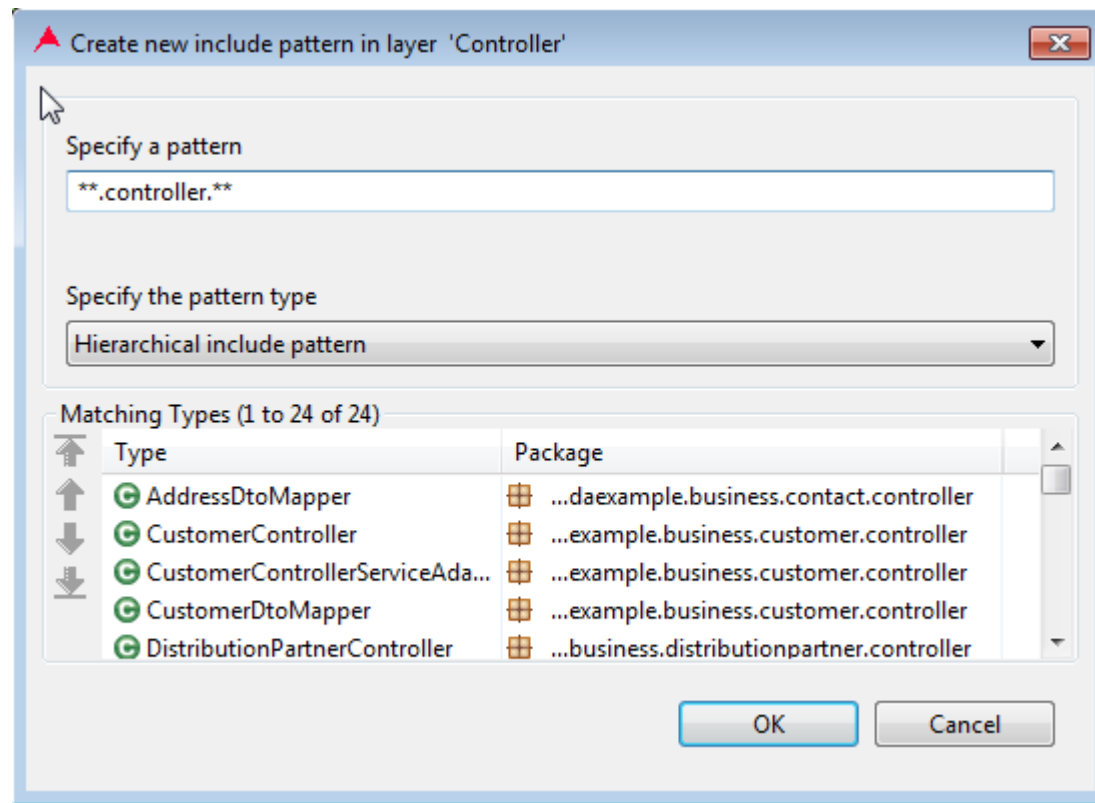
## Step 2: Definition of Layers



## Step 2: Using Stereotypes

- ▶ `<<public>>` : If an element has the stereotype public, it may be accessed by any other non-public element in the same container.
- ▶ `<<unrestricted>>` : If a non-public element has the stereotype unrestricted, it may access any other element in the same container. A public element which is unrestricted may access any other public element in the same container.
- ▶ `<<hidden>>` : If an element has the stereotype hidden, it may not be accessed from outside its parent container.
- ▶ `<<local>>` : If an element has the stereotype local, it may not access any other element outside its parent container, except external elements.

## Step 2: Assignment of types





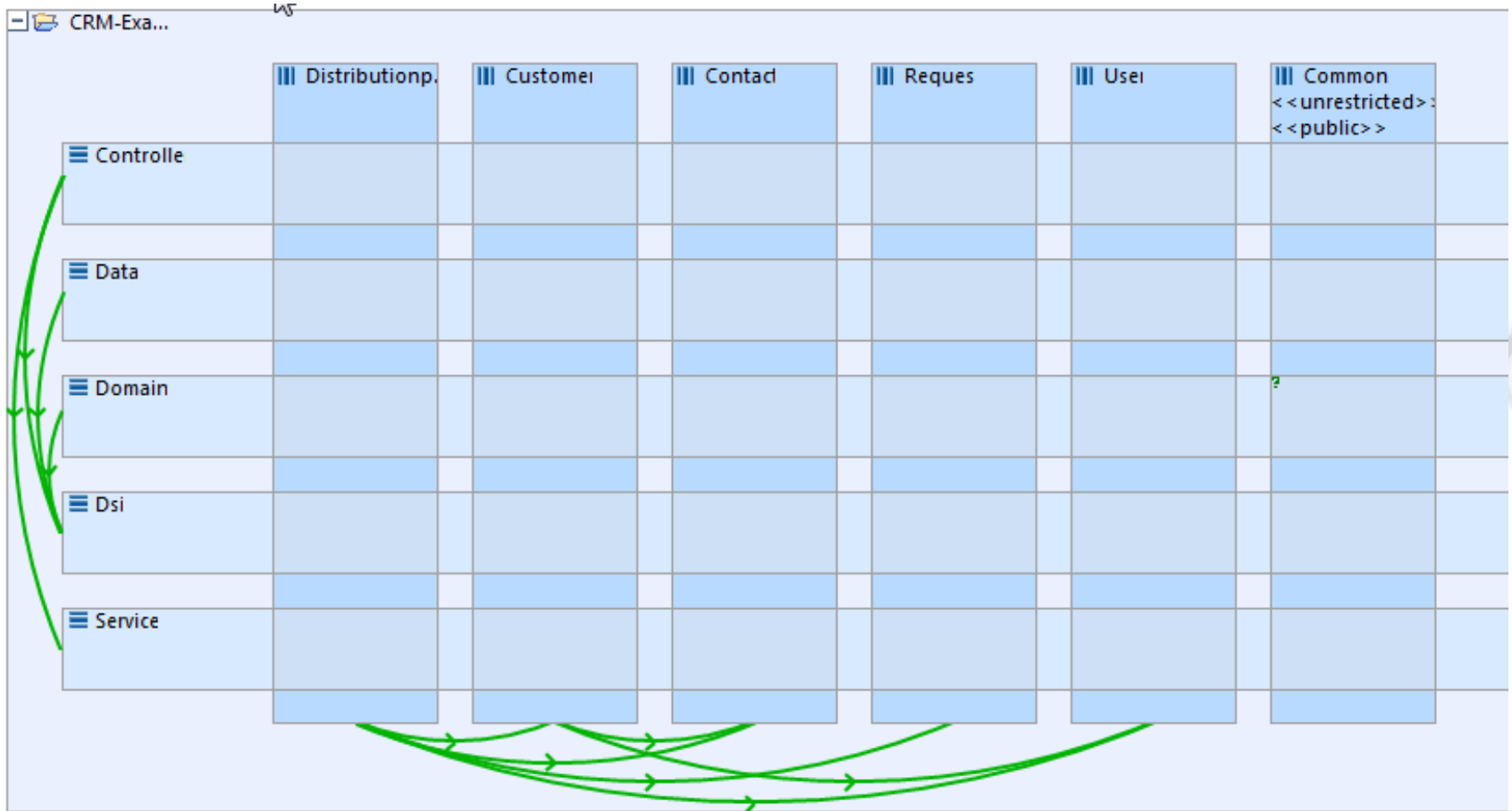
## Step 3: Definition of vertical slices

My Project

	Common <<unrestricted>>	Contact <<unrestricted>>	Customer <<unrestricted>>	Distributionp... <<unrestricted>>	Request <<unrestricted>>	User <<unrestricted>>
Controller <<unrestricted>>						
Data <<unrestricted>>						
Domain <<unrestricted>>						
Dsi <<unrestricted>>						
Service <<unrestricted>>						



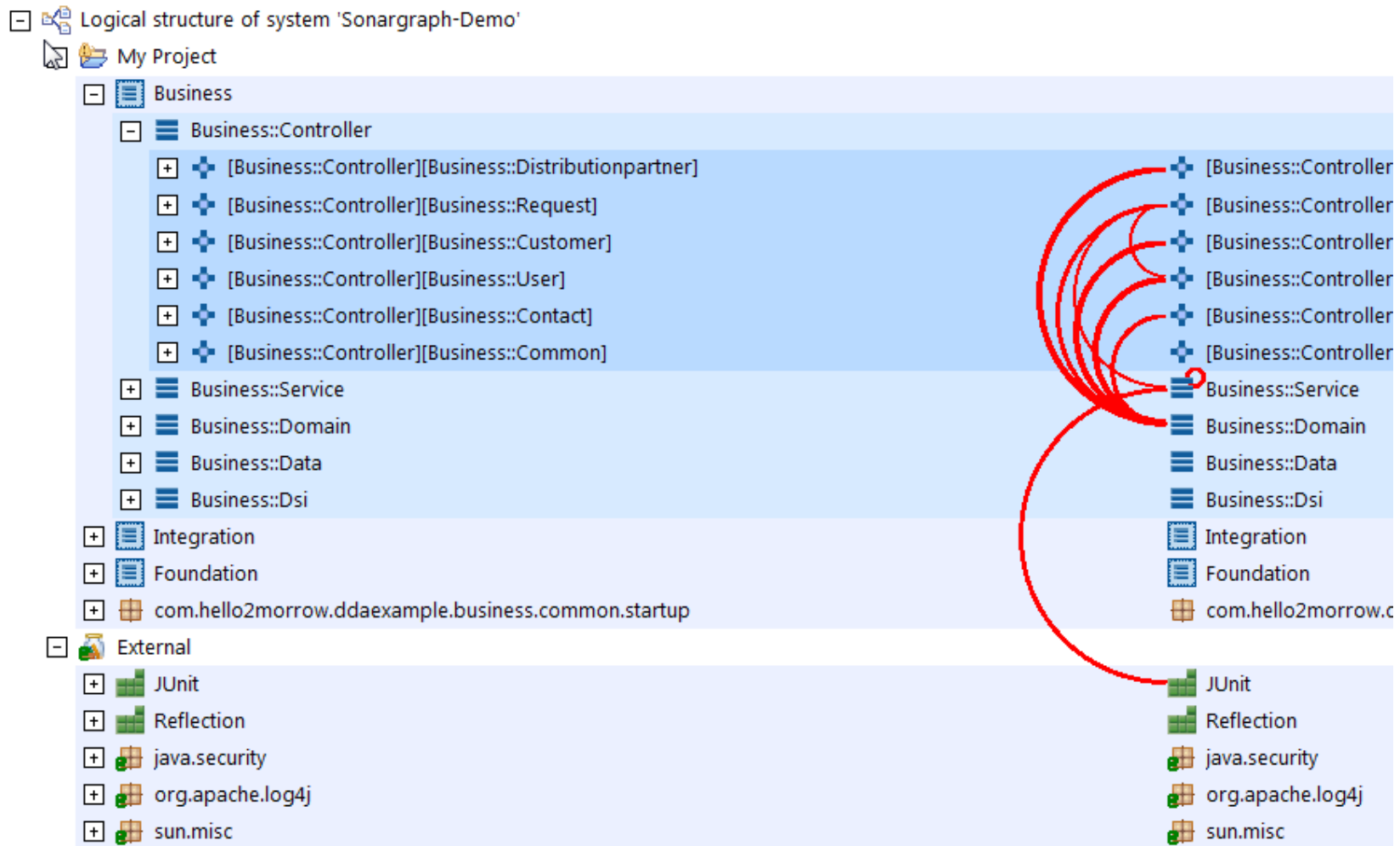
## Step 4: Definition of dependencies



HELLO2MORROW



## Step 5: Exploration of the System



## Step 6: Cut dependency and move types

The screenshot displays the package structure of a system named 'Sonargraph-Demo'. The structure is as follows:

- My Project
  - com.hello2morrow
    - ddaexample
      - business
        - request
          - controller
            - RequestController
        - user
          - controller
            - UserController

A red arrow points from the 'RequestController' class in the 'business/request/controller' package to the 'UserController' class in the 'business/user/controller' package. A context menu is open over the 'RequestController' class, with the 'Cut dependency...' option selected. The menu also includes 'Ignore violation...', 'Focus references and expand', 'Focus references', 'Focus disallowed references and expand', 'Focus disallowed references', and 'Show occurrences in'.

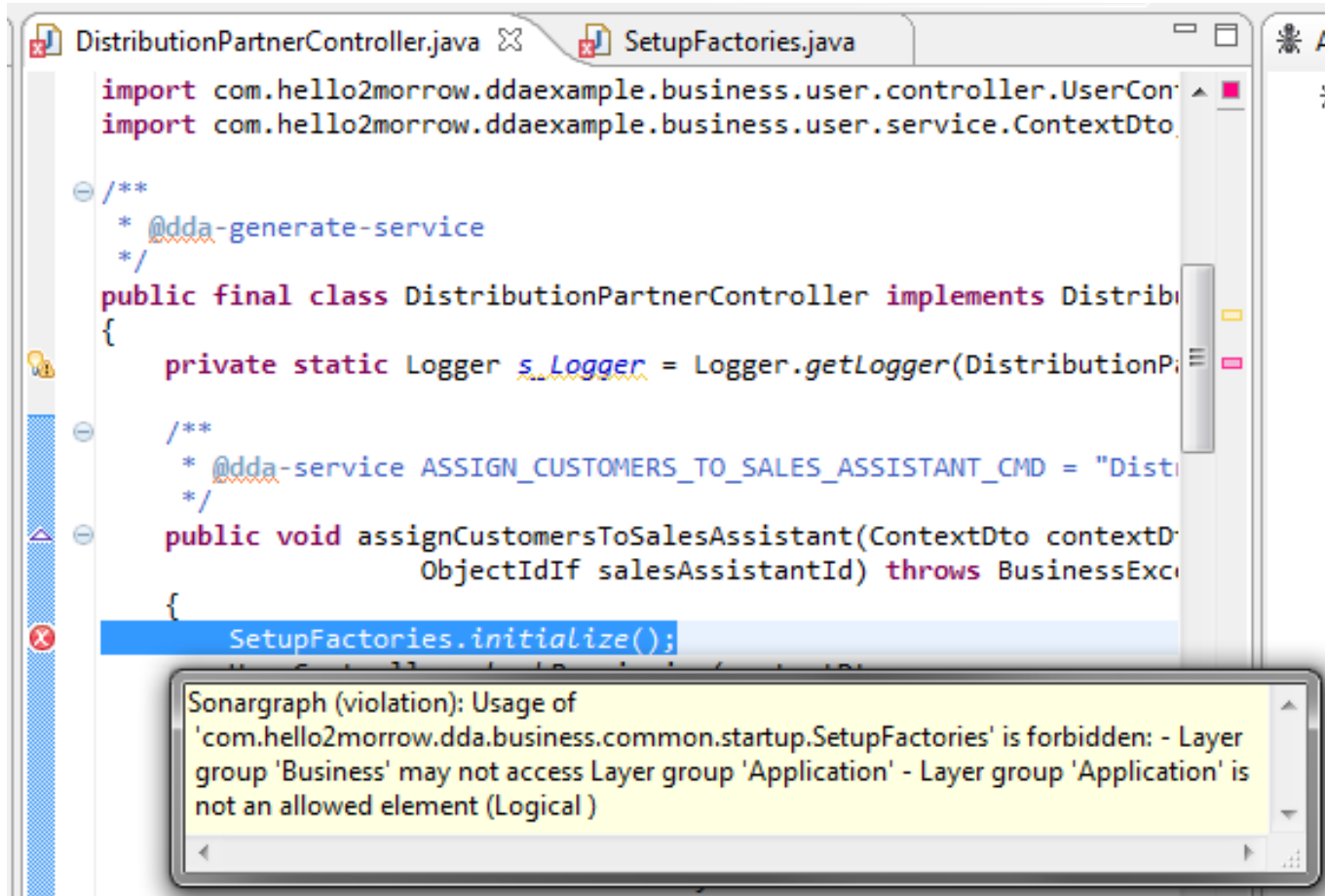
## Step 7: Overview of Tasks

The screenshot shows the IDE's task overview window. The top navigation bar includes: Dashboard, Summary (!), Workspace, Architecture, Exploration, Dependencies, Cycles, Metrics, and a double arrow icon. Below this, the active view is 'Tasks (!)', with other tabs like 'Architecture Viol...', 'Warnings', '? Architecture Cons...', 'Refactorings', and 'Fix Warning Tasks' visible. The main area displays a table of 5 tasks.

Scope	Element type	Element	Description	Priority
CRM-Exam...	Class	...s.request.controller.RestController	Cut type dependency from 'com....	Medium
CRM-Exam...	Class	...vice.test.RestControllerServiceTest	Move type 'com.hello2morrow.d...	Medium
CRM-Exam...	Class	...stributionPartnerControllerServiceTest	Move type 'com.hello2morrow.d...	Medium
CRM-Exam...	Class	...ce.test.CustomerControllerServiceTest	Move type 'com.hello2morrow.d...	Medium
CRM-Exam...	Class	....service.test.UserControllerServiceTest	Move type 'com.hello2morrow.d...	Medium



# Sonargraph Eclipse Integration into Source Editor



The screenshot shows the Eclipse IDE with two Java files open: `DistributionPartnerController.java` and `SetupFactories.java`. The `DistributionPartnerController.java` file contains the following code:

```
import com.hello2morrow.ddaexample.business.user.controller.UserCon
import com.hello2morrow.ddaexample.business.user.service.ContextDto

/**
 * @dda-generate-service
 */
public final class DistributionPartnerController implements Distribi
{
    private static Logger s_Logger = Logger.getLogger(DistributionP

    /**
     * @dda-service ASSIGN_CUSTOMERS_TO_SALES_ASSISTANT_CMD = "Disti
     */
    public void assignCustomersToSalesAssistant(ContextDto contextD
        ObjectIdIf salesAssistantId) throws BusinessExce
    {
        SetupFactories.initialize();
    }
}
```

A Sonargraph violation tooltip is displayed over the `SetupFactories.initialize();` line. The tooltip text is:

Sonargraph (violation): Usage of 'com.hello2morrow.dda.business.common.startup.SetupFactories' is forbidden: - Layer group 'Business' may not access Layer group 'Application' - Layer group 'Application' is not an allowed element (Logical)

# Sonargraph Sonar Plugin - Dashboard

Home Sonargraph - CRM Domain Example Configuration Administrator » Log out Search

Version 7.1.5 - 12. Apr 2012 14:14 Time changes... Configure widgets Edit layout Manage dashboard

**Dashboard**

- Hotspots
- Reviews
- Components
- Violations Drilldown
- Time Machine
- Clouds
- Design
- Libraries

**CONFIGURATION**

- Manual Measures
- Action Plans
- Settings
- Exclusions
- Links
- Project Roles
- History
- Project Deletion

**sonar**

**Structural Debt Index** **94**

1 open tasks  
(with 1 references in code)

**Cost of Structural Debt** **1.034 USD**

**Architecture**

8 violating type dependencies  
3 violating types (1,6%)  
14 violating references

0 ignored violations  
0 unassigned types (0,0%)

**Warnings** **10**

2 cycle groups  
8 duplicate code blocks

0 threshold violations  
0 workspace warnings  
0 ignored warnings

**Relative Package Cyclicity** **13,0%**

5 biggest cycle group size  
5 cyclic packages (12,5%)  
3 type dependencies to cut (approx.)  
4 references to remove (approx.)

**ACD (John Lakos)** **16,1**

2,4 NCCD (John Lakos)  
24.533 byte code instr.

**Lines of code** **7.608**

10.141 lines  
2.610 statements  
185 files

**Classes** **190**

40 packages  
695 methods  
116 accessors

**Violations** **887**

Rules compliance **83,1%**

Blocker	0
Critical	3
Major	248
Minor	525
Info	111

**Package tangle index** **0,6%**


> 3 cycles

**Dependencies to cut**

1 between packages  
2 between files

# Sonargraph Sonar Plugin – Violation Drilldown

- Reviews
- Components
- Violations Drilldown
- Time Machine
- Clouds
- Design
- Libraries



	Blocker	0	
	Critical	3	
	Major	238	<div style="width: 100%; height: 10px; background-color: #ccc;"></div>
	Minor	525	<div style="width: 100%; height: 10px; background-color: #ccc;"></div>
	Info	111	<div style="width: 100%; height: 10px; background-color: #ccc;"></div>

	Sonargraph Architecture Violation	3
	Simplify Boolean Return	2
	Avoid Print Stack Trace	2
	Sonargraph Threshold Violation	1
	Sonargraph Task	1
	Redundant Throws	353

	com.hello2morrow.dda.foundation.common	2	
	com.hello2morrow.ddaexample.business.distributionpartner.controller	1	

	DistributionPartnerController	1
	F2	1
	F1	1

com.hello2morrow.ddaexample.business.distributionpartner.controller.DistributionPartnerController

Coverage | Dependencies | Duplications | LCOM4 | Source | **Violations** | Raw | New window

**15** violations | Blocker: 0 | Critical: 0 | Major: 3 | Minor: 12 | Info: 0

Full source | Time changes... | Sonargraph Architecture Violation (1)

```

39     public void assignCustomersToSalesAssistant(ContextDto contextDto, ObjectIdIf[] customerIds,
40         ObjectIdIf salesAssistantId) throws BusinessException, TechnicalException
41     {
42         //     new Canvas ();
43         SetupFactories.initialize ();

```

**Sonargraph Architecture Violation** | 2 Tage

Logical architecture violation: Uses com.hello2morrow.dda.business.common.startup.SetupFactories. Usage type: Static call  
Explanation: Layer group 'Business' may not access Layer group 'Application' - Layer group 'Application' is not an allowed element (Logical)

# Sonargraph Adapter for Atlassian JIRA Task Management

**JIRA** GOT FEEDBACK? A

Dashboards ▾ Projects ▾ Issues ▾ + Create Issue Quick

crm-domain-example / CRM-1

## Sonargraph Issue in com/hello2morrow/ddaexample/business/contact/data/trans/AddressDataSupplier.java

Edit Assign Comment More Actions ▾ Start Progress Resolve Issue Workflow ▾

**Details**

Type:	Task	Status:	Open (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None <a>✎</a>		

**People**

Assignee:

Reporter:

Vote (0)

**Dates**

Created:

Updated:

**Description**

Line 17 to 61 of com/hello2morrow/ddaexample/business/contact/data/trans/AddressDataSupplier.java is a duplicate of line 13 to 57 of com/hello2morrow/ddaexample/business/contact/data/test/AddressDataSupplier.java.

**Activity**

All Comments Work Log History Aktivität Quelle Überprüfungen

There are no comments yet on this issue.



## Take away: Wenige Regeln können viel bewirken

- ▶ Definition einer Zyklen-freien logische Architektur und einer konsistente Package Namenskonvention. Alle Packages müssen logischen Architekturelementen zugeordnet werden.
- ▶ Keine Zyklen zwischen Packages
- ▶ Kontrolle des Kopplungsgrades (ACD und NCCD mit vernünftigen Schwellwerten)
- ▶ Beschränkung der Größe von Java Sourcen (700 Zeilen)
- ▶ Beschränkung der Zyklomatischen Komplexität von Methoden (z.B. 20)
- ▶ Regelmäßige Überprüfung der Regeln



## Weitere Informationen

- ▶ Whitepaper auf unserer Web-Seite:  
<http://www.hello2morrow.com>
- ▶ Community Lizenz für Projekte < 50 000 Byte Code Instructions
- ▶ 14 Tage Evaluierung ohne Einschränkung

## Referenzen

- ▶ [GOF] Design Patterns, Gamma et al., Addison-Wesley 1994
- ▶ [LSD] Large-Scale C++ Software Design, John Lakos, Addison-Wesley 1996
- ▶ [EXP] Extreme Programming, Kent Beck, Addison-Wesley 1999
- ▶ [AUP] Applying UML and Patterns, Craig Larman, Prentice Hall 2000
- ▶ [TOS] Testing Object-Oriented Systems, Beizer, Addison-Wesley 2000
- ▶ [ASD] Agile Software Development, Robert C. Martin, Prentice Hall 2003