1.– 4. September 2014
in Nürnberg

Herbstcampus

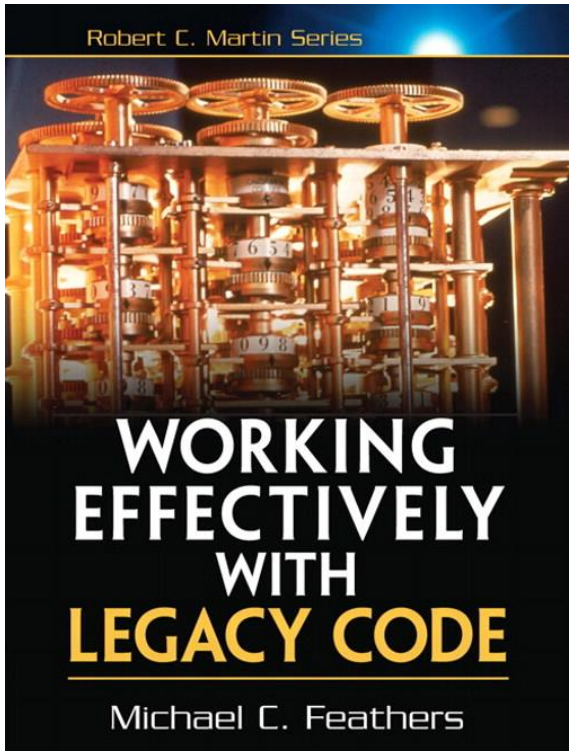
Wissenstransfer
par excellence

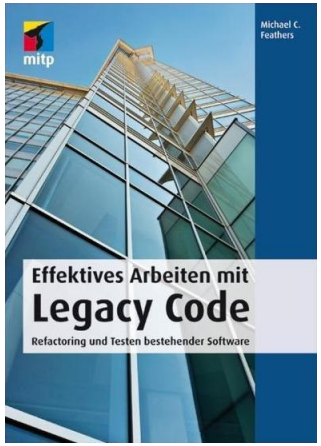
Knochenarbeit alias Königsdisziplin

Techniken und Strategien für den effektiven Umgang mit Legacy-Code

Yann Massard

BTC AG

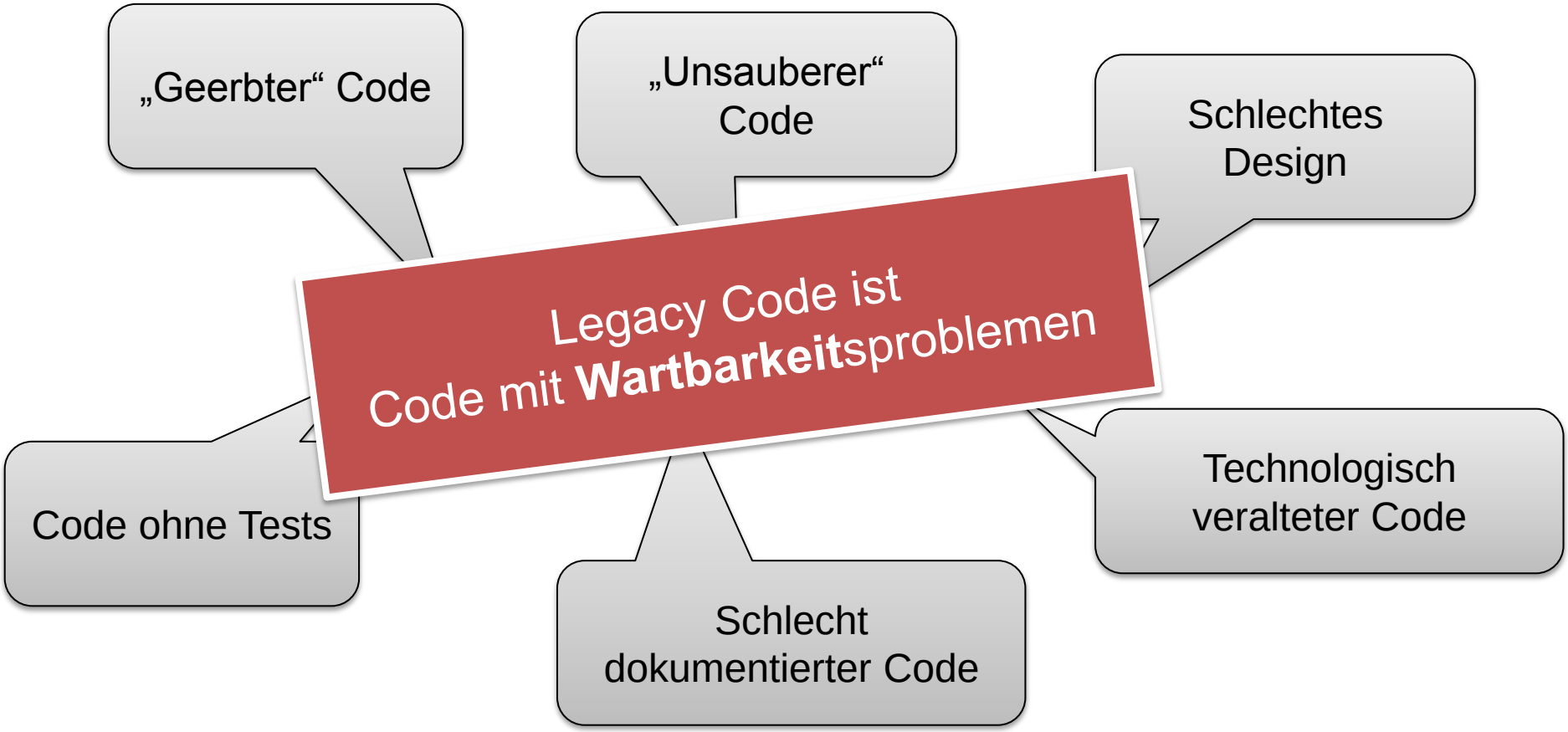




Feathers, Michael (2004). Working Effectively with Legacy Code

- Bugs fixen
- Features hinzufügen
- Performance verbessern
- Wartbarkeit verbessern
- ...

„Was ist Legacy Code?“



„Was ist Legacy Code?“

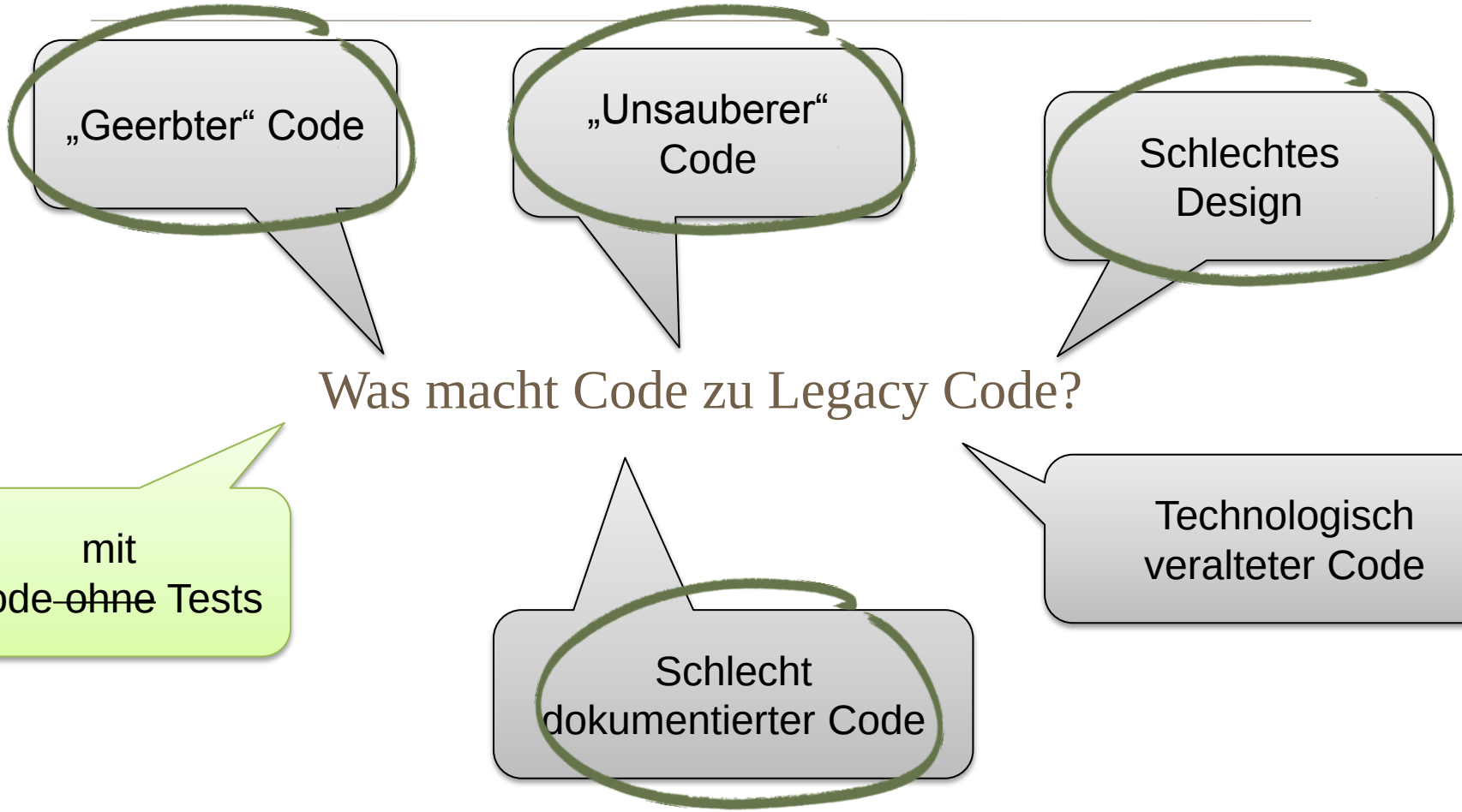
Eine(!) gängige Definition:

Legacy Code = Code without Tests

“Code without tests is bad code. It doesn't matter how well written it is; it doesn't matter how pretty or object-oriented or well-encapsulated it is. With tests, we can change the behavior of our code quickly and verifiably. Without them, we really don't know if our code is getting better or worse.”

Michael Feathers, Working Effectively with Legacy Code

„Was ist Legacy Code?“



„Was ist Legacy Code?“

Ok, dann testen wir den Code eben nachträglich.

Testen von Legacy Code

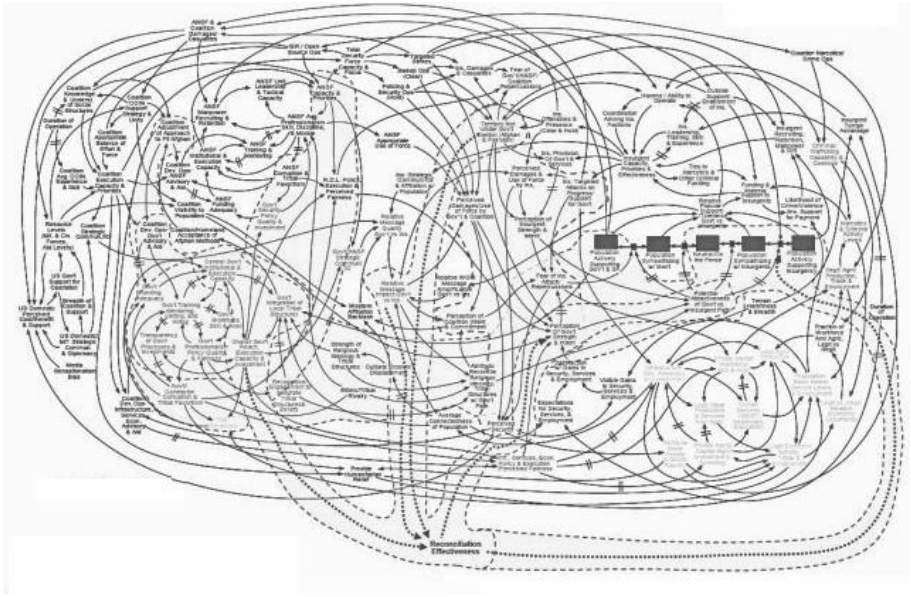
Unit Testing

- JUnit
- TestNG

Dependencies

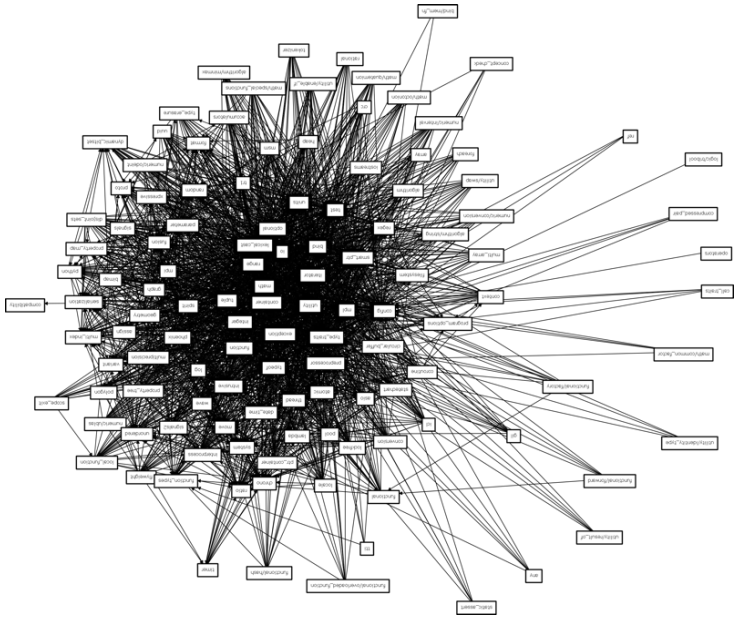
1965:

GOTO SPAGETTI CODE



2014:

DEPENDENCY SPAGETTI CODE



Testen von Legacy Code

Mock-Frameworks

Testen von Legacy Code

Probleme mit Mocks

1. Test-**Aufwand** / Test-**Wartbarkeit**
   - ... bei riesigen Schnittstellen
   - ... bei vielen Abhängigkeiten
2. **Aussagekraft** von Tests
   - ... bei vielen Mocks
3. **Technische Probleme**
   - Finale Klassen, statische Methoden
   - Probleme mit PowerMock und Jmockit
     - Java 1.8
     - Bugs / Interferenzen mit anderen Technologien
     - http://java.dzone.com/articles/why-wide-usage-powermock

Testen von Legacy Code

```
@Test
public void testNext() throws Exception {
    ProductOrderWizardContext productOrderWizardContextMock = Mockito.mock(ProductOrderWizardContext.class);
    OrderGuiState orderGuiStateMock = Mockito.mock(OrderGuiState.class);
    ProductGuiState productGuiStateMock = Mockito.mock(ProductGuiState.class);
    ChangeTicketServiceCore changeTicketServiceCoreMock = Mockito.mock(ChangeTicketServiceCore.class);
    InteractionMessageUiService interactionMessageUiServiceMock = Mockito.mock(InteractionMessageUiService.class);
    InMemoryAttachmentGuiState inMemoryAttachmentGuiStateMock = Mockito.mock(InMemoryAttachmentGuiState.class);
    InteractionAttachmentUiService interactionAttachmentUiServiceMock = Mockito.mock(InteractionAttachmentUiService.class);
    OrderServiceCore orderServiceCoreMock = Mockito.mock(OrderServiceCore.class);
    CurrentInteractionInteractionAttachmentProvider currentInteractionInteractionAttachmentProviderMock = Mockito.mock(CurrentIn
    TaskControllerPort taskControllerMock = Mockito.mock(TaskControllerPort.class);
    UiTransactionHelper uiTransactionHelperMock = Mockito.mock(UiTransactionHelper.class);
    ChangeDesiredDateAction changeDesiredDateActionMock = Mockito.mock(ChangeDesiredDateAction.class);

    ReflectionUtils.setProperty("productOrderWizardContext", productOrderSummaryAction, productOrderWizardContextMock);
    ReflectionUtils.setProperty("changeTicketServiceCore", productOrderSummaryAction, changeTicketServiceCoreMock);
    ReflectionUtils.setProperty("interactionMessageUiService", productOrderSummaryAction, interactionMessageUiServiceMock);
    ReflectionUtils.setProperty("interactionAttachmentUiService", productOrderSummaryAction, interactionAttachmentUiServiceMock)
    ReflectionUtils.setProperty("orderServiceCore", productOrderSummaryAction, orderServiceCoreMock);
    ReflectionUtils.setProperty("changeDesiredDateAction", productOrderSummaryAction, changeDesiredDateActionMock);
    ReflectionUtils.setProperty("taskController", productOrderSummaryAction, taskControllerMock);
    ReflectionUtils.setProperty("uiTransactionHelper", productOrderSummaryAction, uiTransactionHelperMock);
```

```
LanPortProduct product = new LanPortProduct();
List<AbstractBaseProduct> articles = getArticles(PricingMethodType.NONRECURRING, PricingMethodType.NONRECURRING, PricingMethodType.NONRECURRING);
ProductPrice productPrice = new ProductPrice();
productPrice.setAccountingRule(PricingMethodType.LSKN);
product.setProductPrice(productPrice);
AccountEntity accountEntity = new AccountEntity();
accountEntity.setId(1L);
final InteractionChangeTicket changeTicket = new InteractionChangeTicket();
final Calendar calendar = Calendar.getInstance();
calendar.set(2014, 5, 4);
changeTicket.setDesiredDate(calendar.getTime());
final InstallationDataGuiState installationDataGuiState = new InstallationDataGuiState(new InstallationData(), null);
final ShipmentDataGuiState shipmentDataGuiState = new ShipmentDataGuiState(new Place());
final ChangeTicketEntity changeTicketEntity = new ChangeTicketEntity();
changeTicketEntity.setId(1L);
final InMemoryCommentGuiState inMemoryCommentGuiState = new InMemoryCommentGuiState(null, null);
inMemoryCommentGuiState.doAddComment("TESTNACHRICHT", MessageType.PRIVATE);
List<AttachmentItem> attachmentMetadataList = new ArrayList<>();
final InteractionAttachment attachmentMetadata = new InteractionAttachment();
attachmentMetadataList.add(new AttachmentItem(attachmentMetadata));

Mockito.when(productOrderWizardContextMock.getOrderGuiState()).thenReturn(orderGuiStateMock);
Mockito.when(productOrderWizardContextMock.getChangeTicket()).thenReturn(changeTicket);
Mockito.when(orderGuiStateMock.getInstallationGuiState()).thenReturn(installationDataGuiState);
Mockito.when(orderGuiStateMock.getShipmentGuiState()).thenReturn(shipmentDataGuiState);
Mockito.when(orderGuiStateMock.getProductGuiState()).thenReturn(productGuiStateMock);
Mockito.when(productGuiStateMock.getMainProduct()).thenReturn(product);
Mockito.when(productGuiStateMock.extractArticlesFromGuiStates()).thenReturn(articles);
Mockito.when(productGuiStateMock.getAccount()).thenReturn(accountEntity);
Mockito.when(changeTicketServiceCoreMock.createChangeTicket(changeTicket, false)).thenReturn(changeTicketEntity);
Mockito.when(productOrderWizardContextMock.getInMemoryCommentGuiState()).thenReturn(inMemoryCommentGuiState);
Mockito.when(productOrderWizardContextMock.getInMemoryAttachmentGuiState()).thenReturn( inMemoryAttachmentGuiStateMock);
Mockito.when(inMemoryAttachmentGuiStateMock.getAttachments()).thenReturn(attachmentMetadataList);
Mockito.when(currentInteractionInteractionAttachmentProviderMock.getInteractionAttachmentBag()).thenReturn( new InteractionAttachmentBag());
Mockito.when(taskControllerMock.createTask(Mockito.any(Task.class))).thenReturn(new Task());
```

Testen von Legacy Code

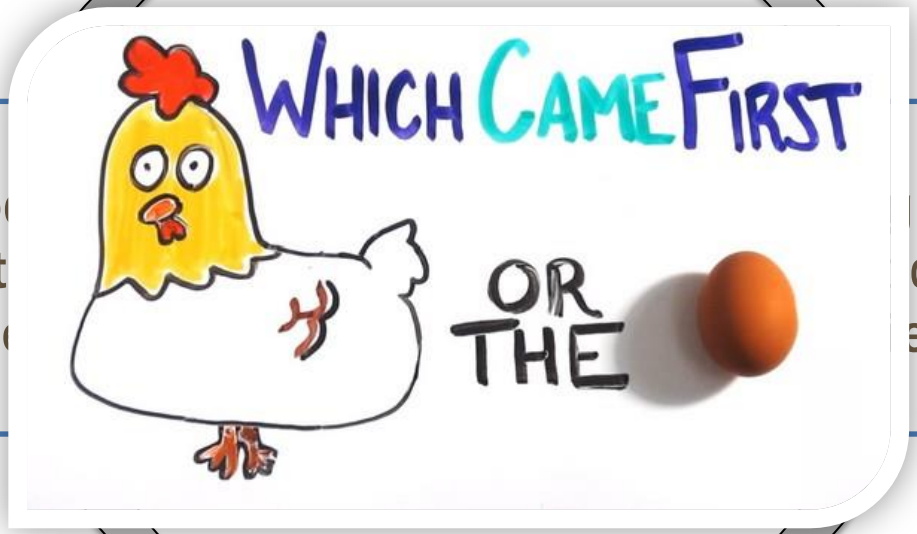
```
final String url = productOrderSummaryAction.next();

Assert.assertEquals(url, "/ui/pageflow/changes/productOrderCreated.seam");
Assert.assertEquals(changeTicket.getId(), Long.valueOf(1L));
Assert.assertEquals(productPrice.getAccountingRule(), PricingMethodType.NONRECURRING);
```

Grundproblem

Man müsste…

dazu…

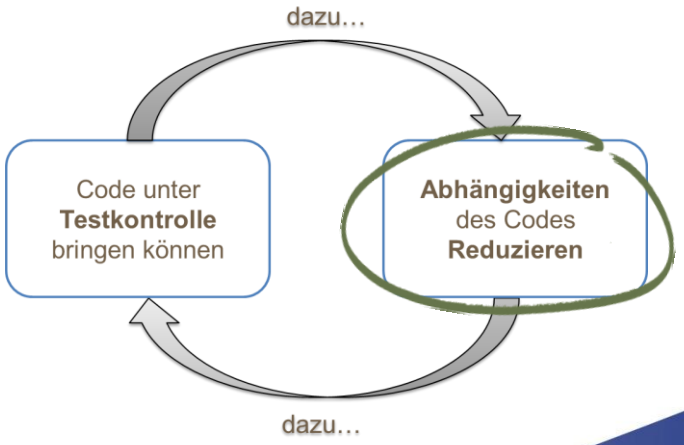
Co
Test
bring



keiten
des
eren

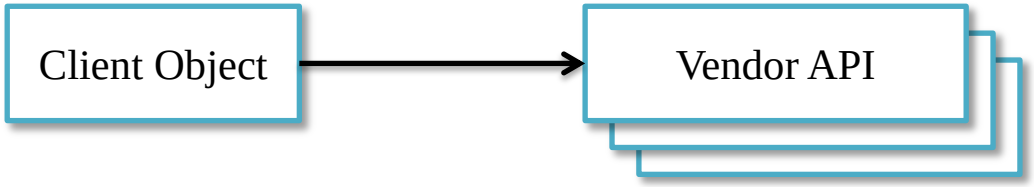
dazu…

Techniken zum Aufheben von Dependencies



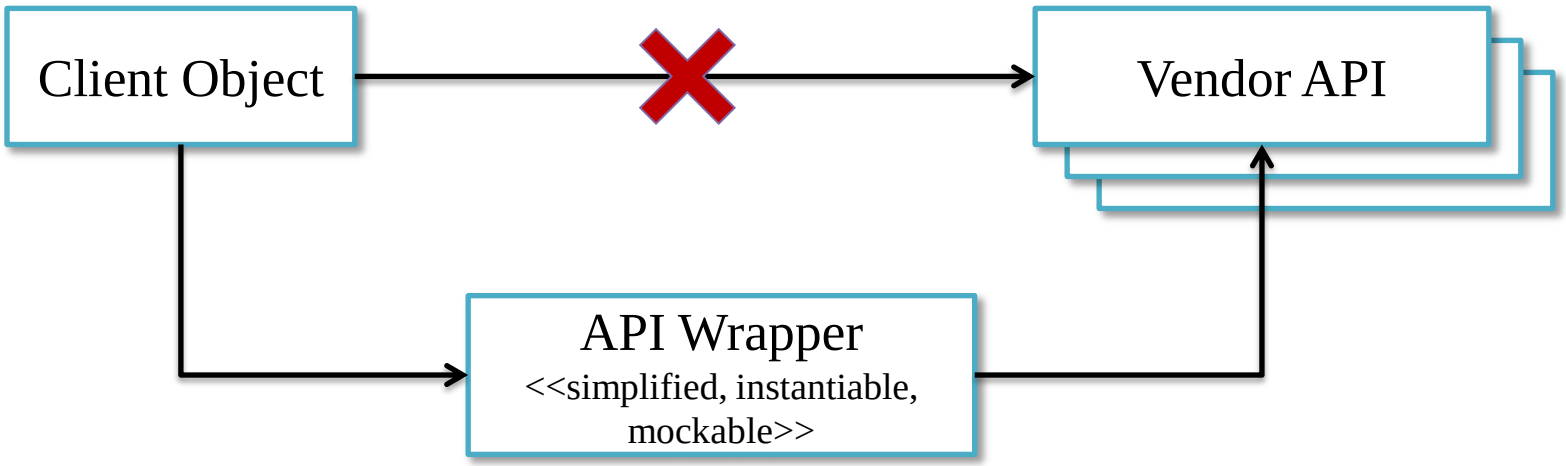
Techniken zum Aufheben von Dependencies

- **Problem**:
  - Abhängigkeit zu...
    - komplexer Schnittstelle
    - finalen Klassen
    - statischen Methoden



Techniken zum Aufheben von Dependencies

- Skin and Wrap API



Techniken zum Aufheben von Dependencies

- Skin and Wrap API

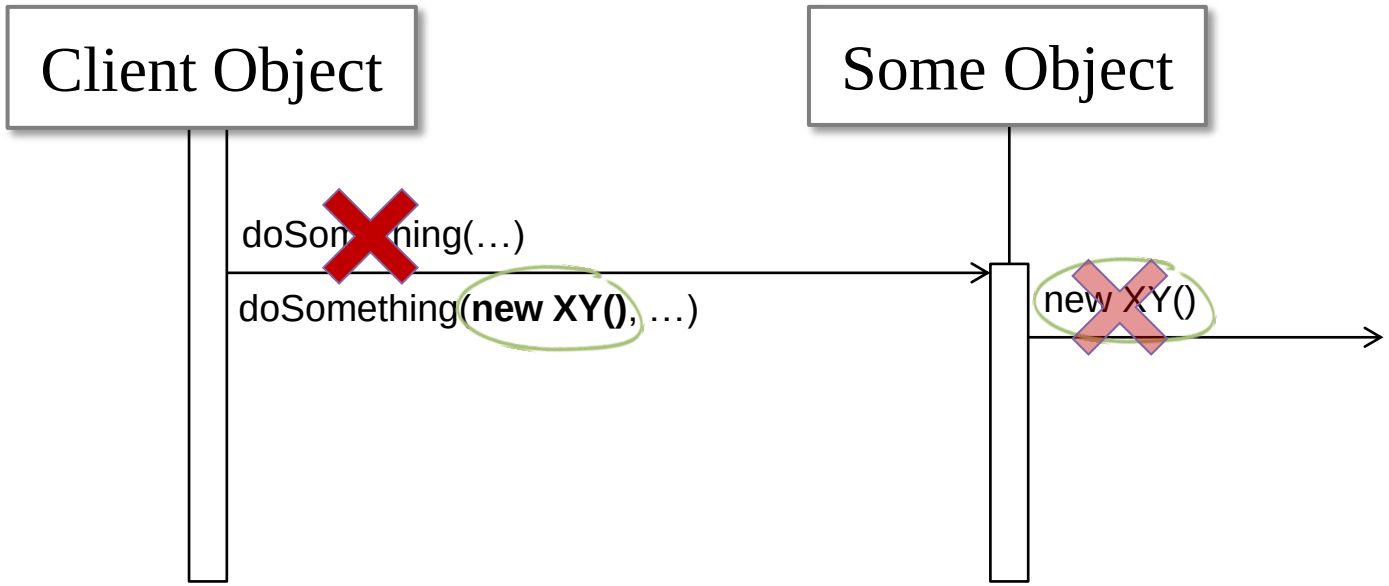
```
public class WrappedFoo {

    public String doSomething() {

        return VendorFoo.doSomething();

    }

}
```

```
public class VendorFoo {

    public native static String doSomething();

}
```

Techniken zum Aufheben von Dependencies

Extract Parameter

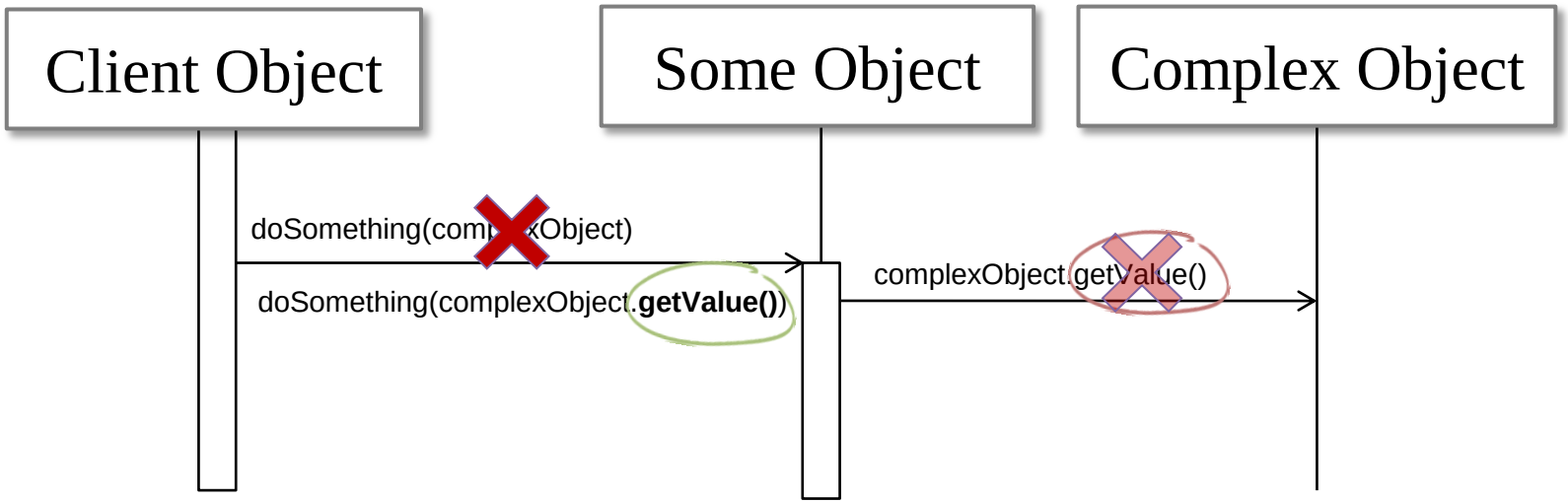
- **Problem:**
  - **new** XY() in einer Methode



Techniken zum Aufheben von Dependencies

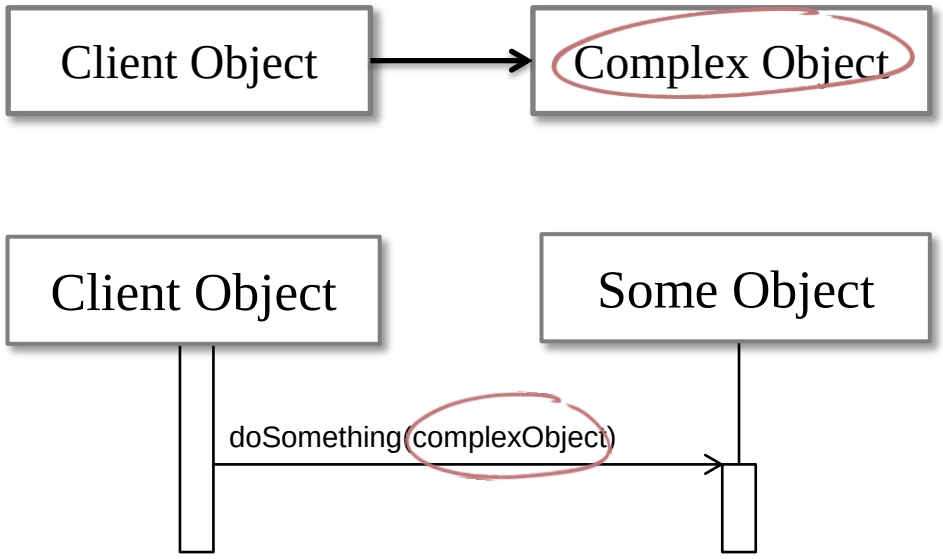
Extract Parameter

- **Problem:**
  - Komplexes Objekt als Übergabe-Parameter, worauf eigentlich nur ein Getter aufgerufen wird



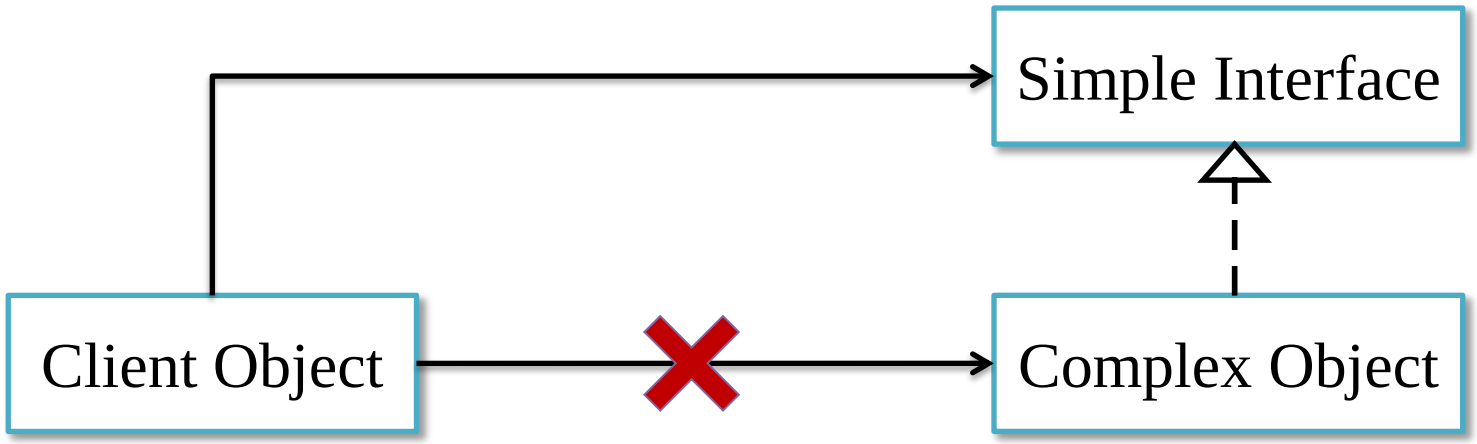
Techniken zum Aufheben von Dependencies

- **Problem:**
  - Abhängigkeit zu komplexem Objekt



Techniken zum Aufheben von Dependencies

- Extract Interface



Generelles Konzept: *Interface Segregation*

Techniken zum Aufheben von Dependencies

- Extract Interface

```
public class Foo {

    private VeryComplexObject veryComplexObject;

    private void bar(String name) {
        // ...
        veryComplexObject.persistString(name);
        // ...
    }

}
```

```
public class VeryComplexObject {

    // ... many many methods with terrible code!

    public void persistString(String s) {
        // ...
    }

    // ...
}
```

Techniken zum Aufheben von Dependencies

Extract Interface

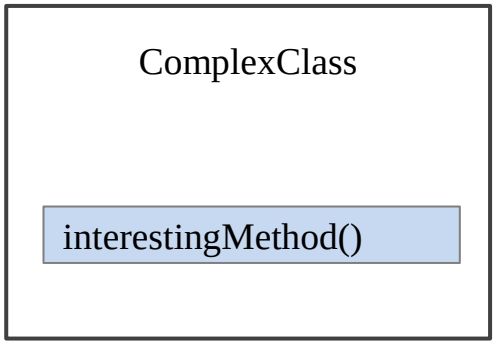
```
public class Foo {

    private StringPersister stringPersister;

    private void bar(String name) {
        // ...
        stringPersister.persistString(name);
        // ...
    }

}
```

```
public interface StringPersister {

    void persistString(String s);

}
```

```
public class VeryComplexObject implements StringPersister {

    // ... many many methods with terrible code!

    public void persistString(String s) {
        // ...
    }

    // ...
}
```

Techniken zum Aufheben von Dependencies

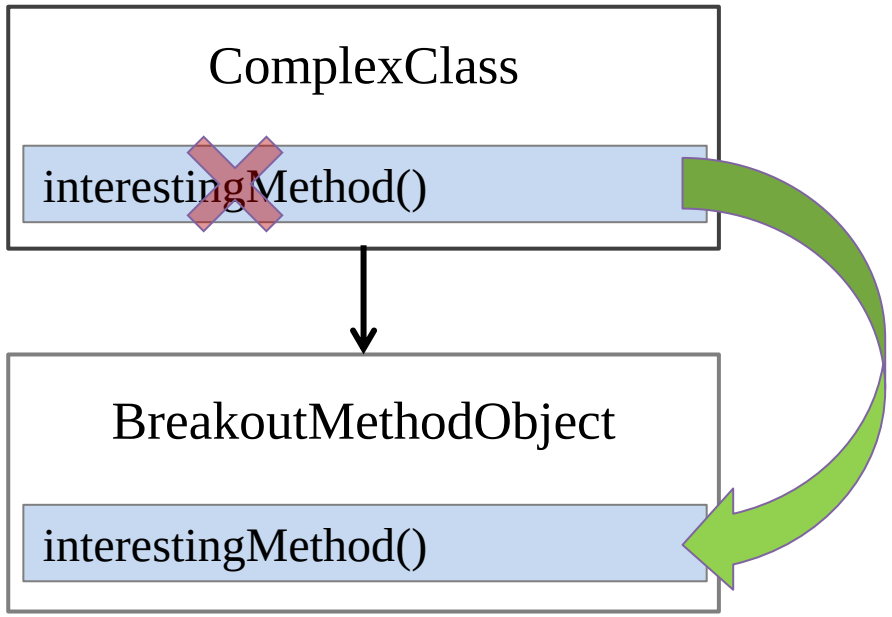
- **Problem:**
  - Eine Methode in einer komplexen Klasse soll getestet werden
  - Die Klasse kann nicht instanziert werden



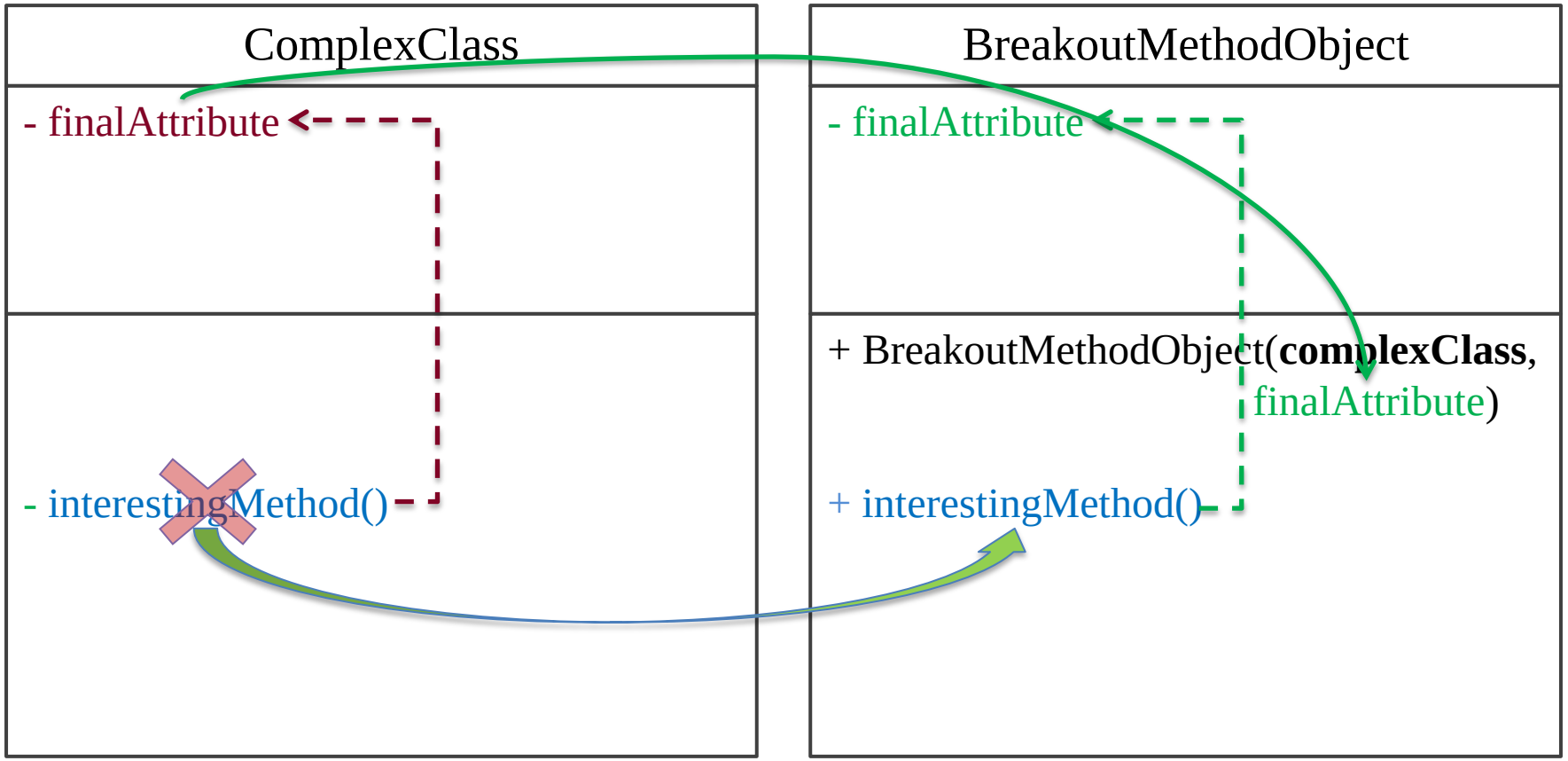
- **Möglichkeiten**:
  - Methode `static` und sichtbar machen
  - Klasse **mock**en, außer zu testende Methode
  - **Breakout Method Object** extrahieren

Techniken zum Aufheben von Dependencies

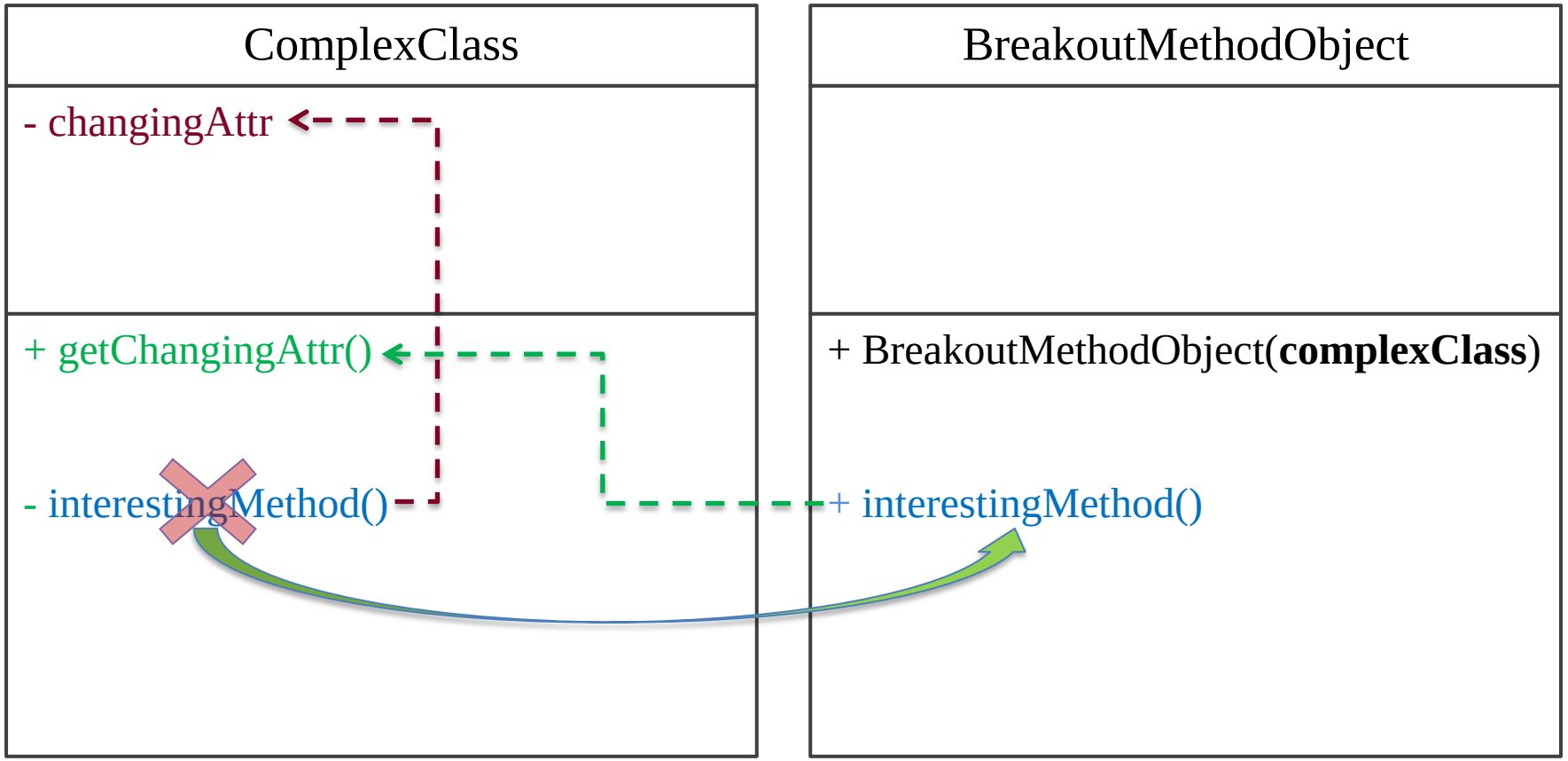
- Breakout Method Object



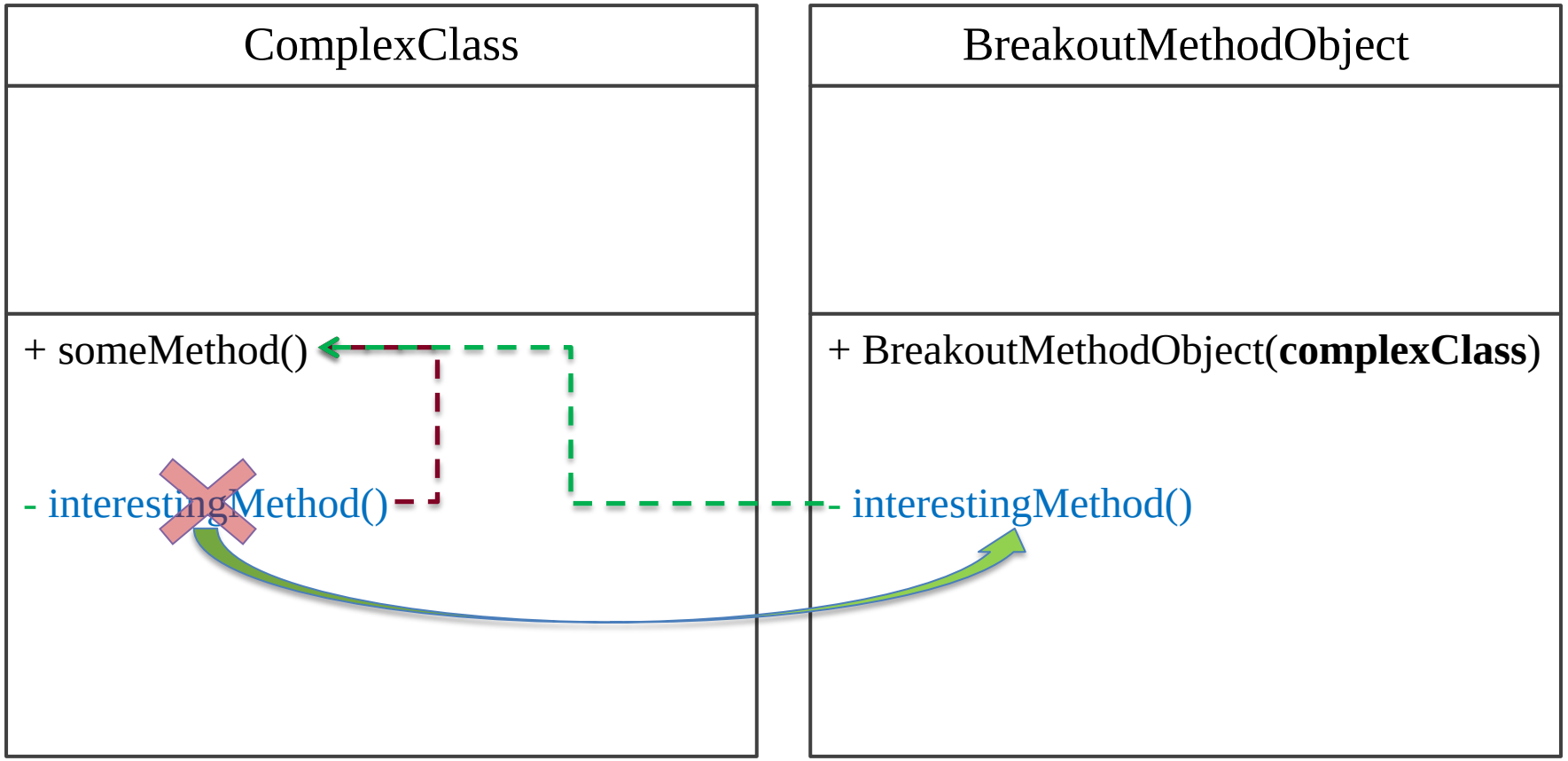
Techniken zum Aufheben von Dependencies



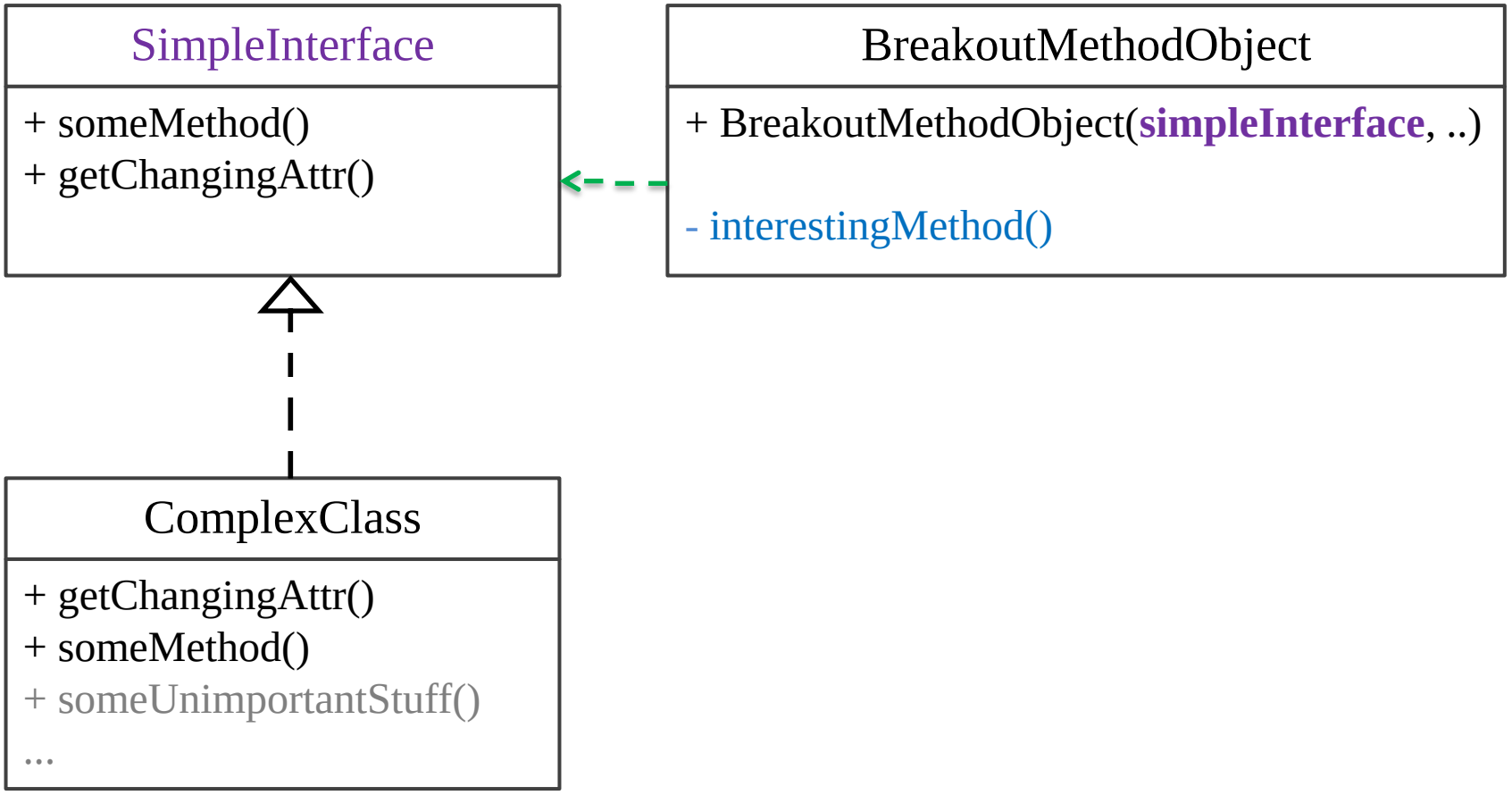
Techniken zum Aufheben von Dependencies



Techniken zum Aufheben von Dependencies



Techniken zum Aufheben von Dependencies



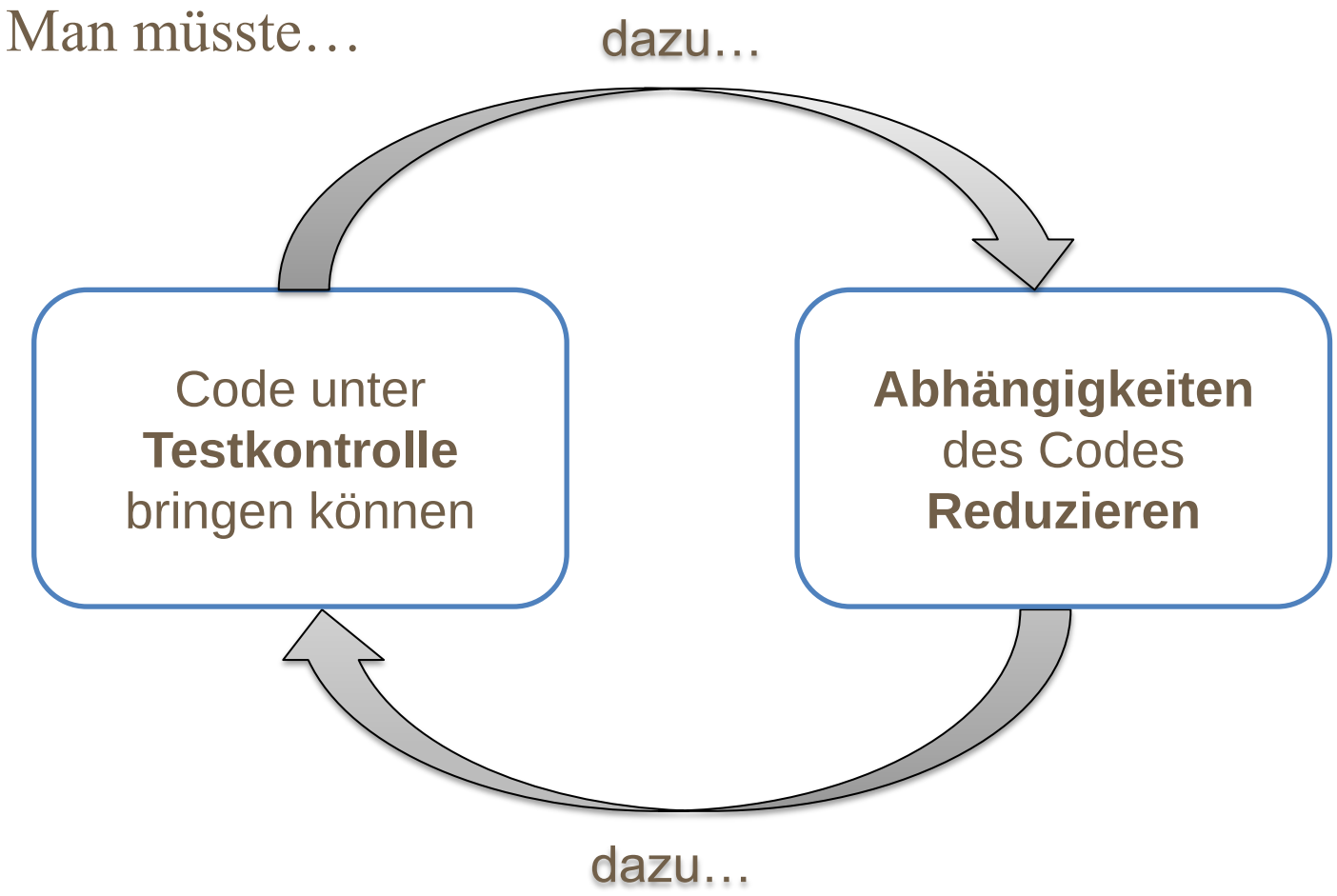
Techniken zum Aufheben von Dependencies

```
public class ArticleGuiStateList {

   private final ListMultimap<Long, AbstractArticleGuiState<AbstractBaseProduct>> guiStatesByOfferId;

   public ArticleGuiStateList(
      ListMultimap<Long, AbstractArticleGuiState<AbstractBaseProduct>> guiStatesByOfferId) {...}
   public ArticleGuiStateList(List<AbstractArticleGuiState> articleGuiStates) {...}

   public ListMultimap<Long, AbstractArticleGuiState<AbstractBaseProduct>> getRecurringGuiStatesByOfferId() {...}
   public ListMultimap<Long, AbstractArticleGuiState<AbstractBaseProduct>> getNonRecurringGuiStatesByOfferId() {...}

}
```

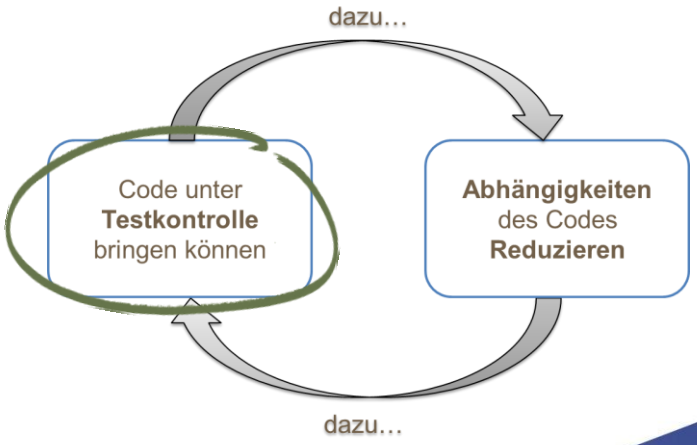
Techniken zum Aufheben von Dependencies

- Weitere Refactorings
  - **Extract** (method / variable / field / delegate / superclass / interface ...)
  - **Inline** (variable / method / class)

Grundproblem



Techniken zum Testen von Legacy Code



Techniken zum Testen von Legacy Code

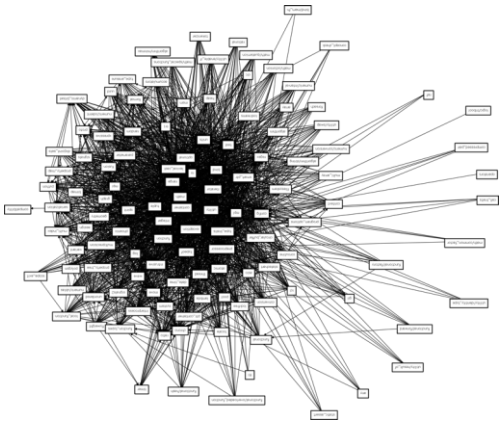
Wie schreibt man Tests für bestehenden Code?
(„Vorhandenes Verhalten fixieren“)

... wenn man noch nicht einmal weiß, was er tut.

Techniken zum Testen von Legacy Code

Generell:

- Wo mit vertretbarem Aufwand möglich: Unit Tests!
- Sonst: gröbere Strukturen testen



Techniken zum Testen von Legacy Code

Charakterisierungstest

- **Problem:**
  - Unit Test für eine oder mehrere Klassen schreiben
  - Verhalten der Implementierung unklar
- **Vorgehen:**
  - Test ohne Asserts schreiben
  - Asserts den realen Ergebnissen entsprechend schreiben

Techniken zum Testen von Legacy Code

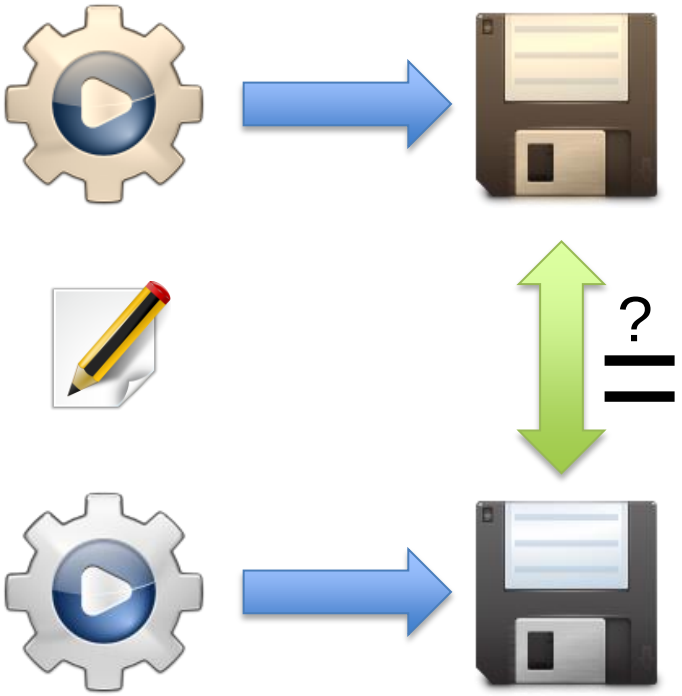
Golden Master Technique

- *Sonderfall des Charakterisierungstests*
  - Getestet wird eine komplexe Klasse oder eine Menge von Klassen
  - Die Ergebnisse sind **serialisierbar**

Techniken zum Testen von Legacy Code

Vorgehen

1. Test ausführen
2. Ergebnisse (Text) speichern
3. Änderungen durchführen
4. Test erneut ausführen
5. Ergebnisse alt/neu vergleichen



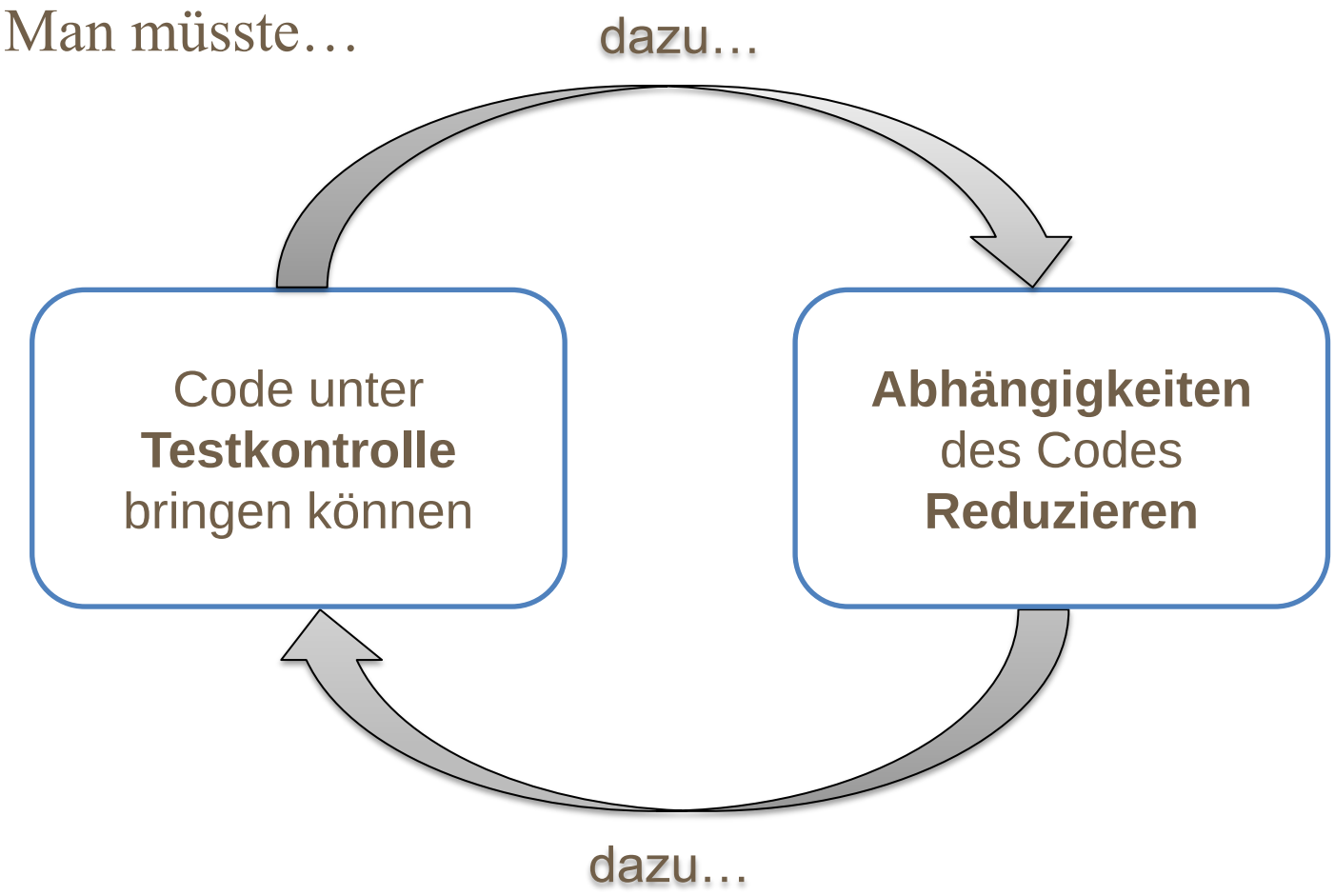
Techniken zum Testen von Legacy Code

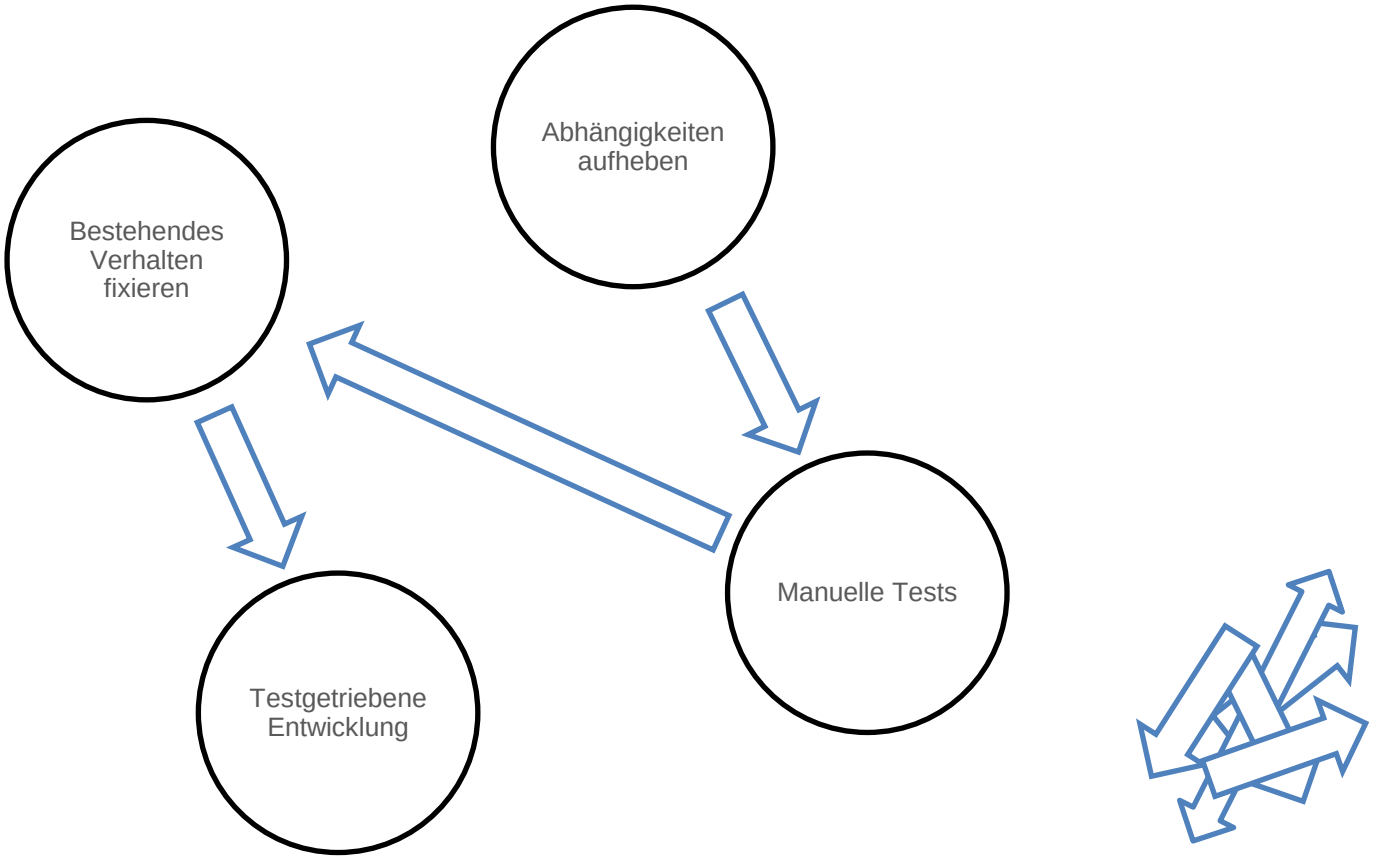
Besonders geeignete Einsatzgebiete:

- **DB-Persistierung**
  - DB-Inhalt vorher → Aufruf → DB-Inhalt nachher
  - Tools: 

- **WebServices**
  - Request → Response

- …und überall sonst, wo Systemzustände / Ergebnisse serialisierbar sind


Man müsste…
dazu…
Code unter
Testkontrolle
bringen können
Abhängigkeiten
des Codes
Reduzieren
dazu…

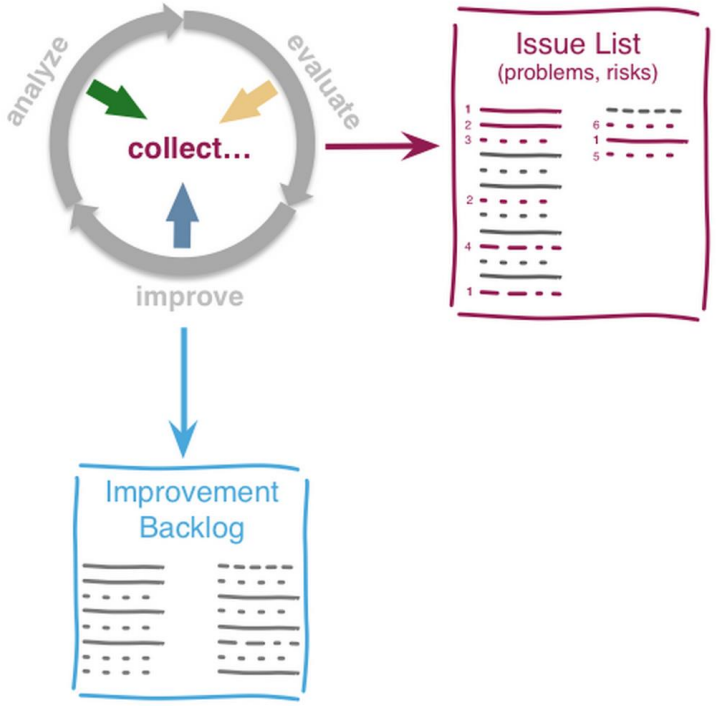

Abhängigkeiten aufheben
Bestehendes Verhalten fixieren
Manuelle Tests
Testgetriebene Entwicklung

Vorgehensempfehlungen und Strategien

Vorgehensempfehlungen und Strategien

Behandlung von „Legacy“-Themen

- Gleichwertige Behandlung von „technical debt“-Themen wie fachliche Themen



Graphik: http://aim42.org/
Siehe auch: Capgemini Agile Legacy Lifecycle

Vorgehensempfehlungen und Strategien

Generelle Empfehlungen

BIG BANG

Evolutionäres Vorgehen

Vorgehensempfehlungen und Strategien

Vorsicht vor BIG BANG!

- Unproduktiver Code...
  - kann schnell obsolet werden
  - kann häufig nur schwer integriert werden
  - Konkurrierender produktiver Code entwickelt sich weiter!

BIG
(and unfortunately obsolete)
BANG

Evolutionäres Vorgehen

Evolutionäres Vorgehen

- Kleine Schritte
- Alle Änderungen sind produktiv
- U.U. höhere Kosten durch Kleinschrittigkeit
- Niedriges Investitionsrisiko durch mehrwertgenerierende Zwischenschritte
- Temporäre Unschönheiten werden akzeptiert (Kulturfrage)
- Ideal-Ziel wird u.U. nie erreicht

Vorgehensempfehlungen und Strategien

Generelle Empfehlung

- Aufwand und Risiko bewerten

BIG BANG, wenn...	**Evolutionäres Vorgehen, wenn...**
Überschaubare Änderungen	
Sehr kurze Umsetzungsdauer	**Sonst.**
Anforderungslage und derzeitige (tatsächliche) Funktionalität klar	

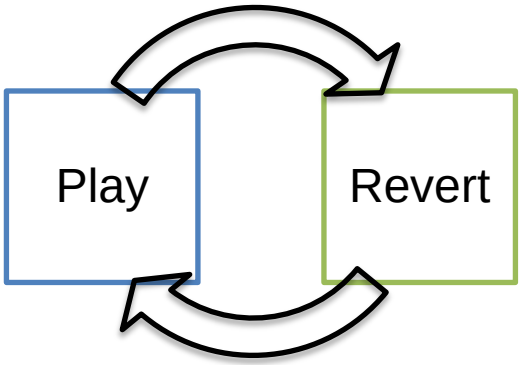
Vorgehensempfehlungen und Strategien

Pair Programming

Vorgehensempfehlungen und Strategien

Scratch Refactoring

1. Änderungen schnell und unsauber durchführen
2. Revert



- **Ziele**
  - Code kennen lernen
  - Probleme und Abhängigkeiten vorab erkennen
  - Sicherheit gewinnen

Vorgehensempfehlungen und Strategien

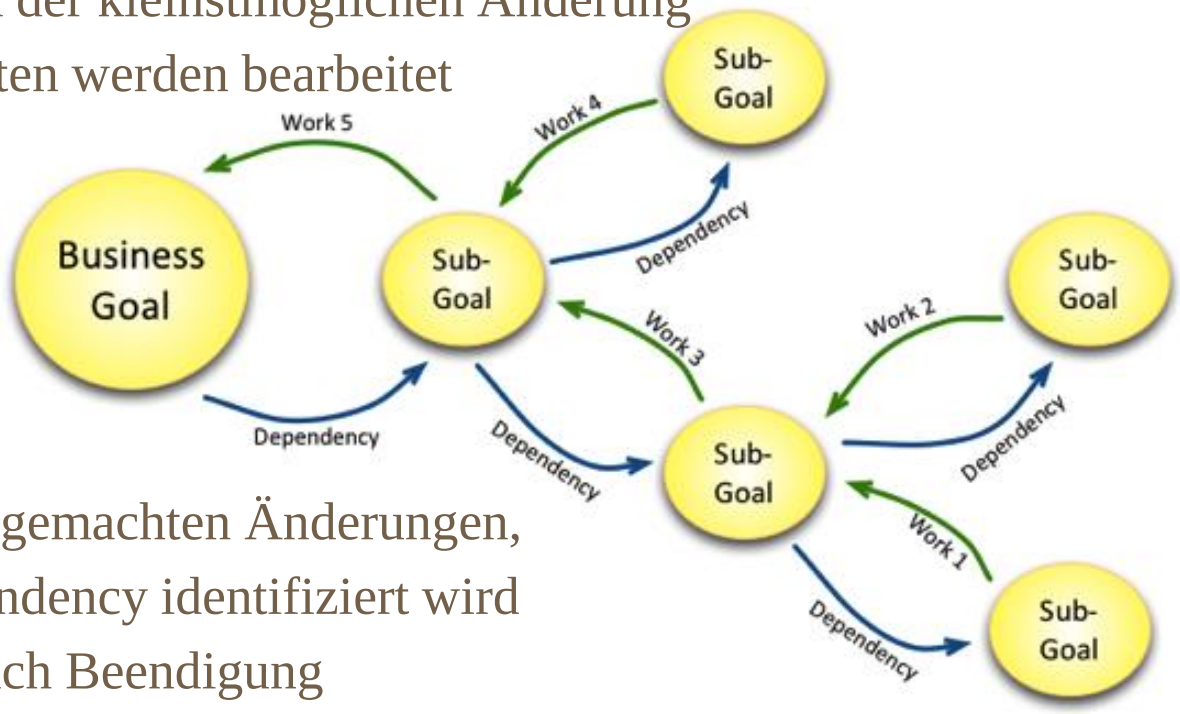
… schön, aber…

wo anfangen?

Vorgehensempfehlungen und Strategien

Mikado-Methode

- Identifikation der kleinstmöglichen Änderung
- Nur Blattknoten werden bearbeitet

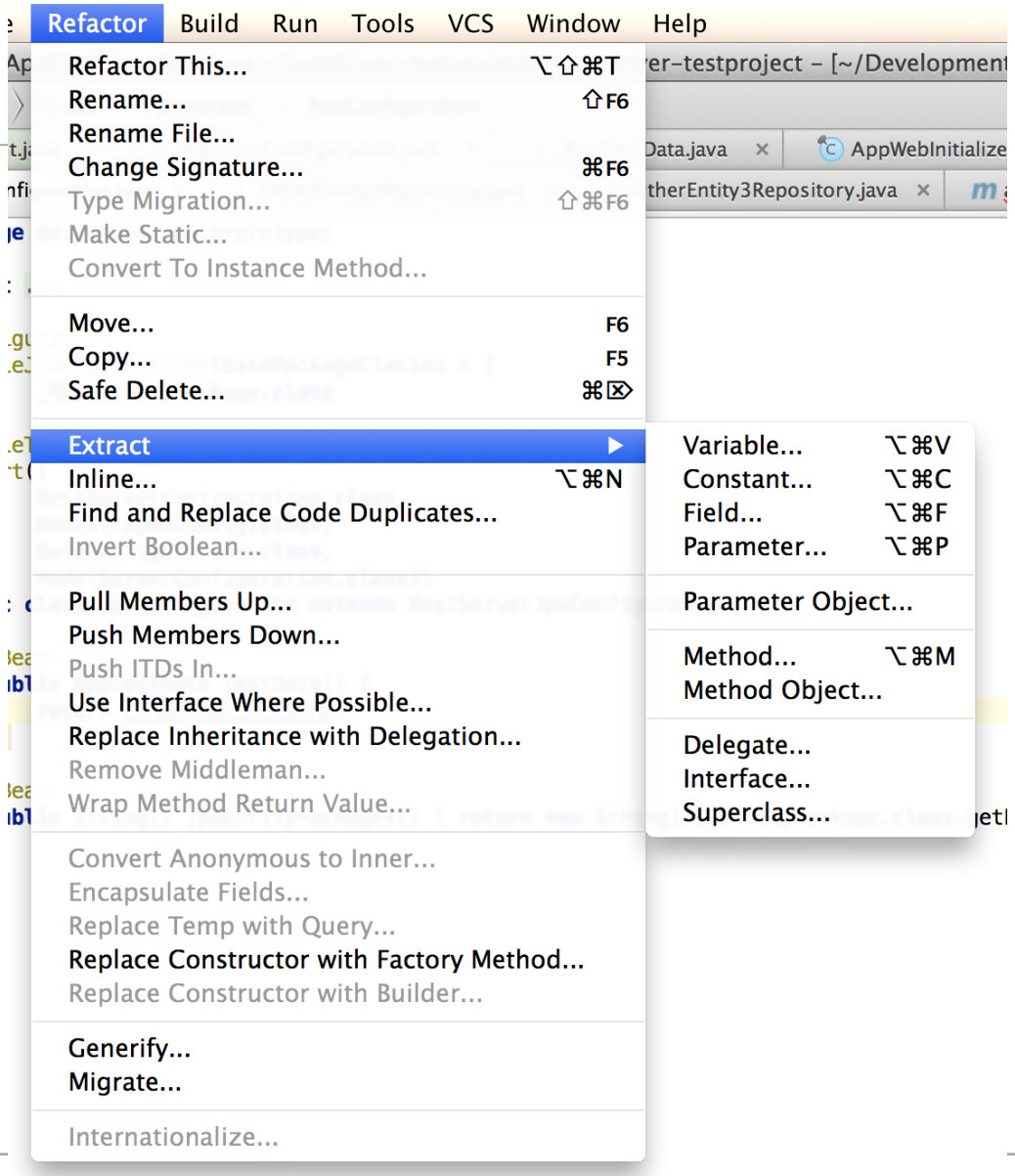


- **Wichtig**:
  - **Revert** der gemachten Änderungen, wenn Dependency identifiziert wird
  - **Commit** nach Beendigung von Unteraufgaben

Bild: http://nomad8.com/organisational-change-with-mikado/

Sonstiges

Know your IDE!



Knochenarbeit

Königsdisziplin

Vielen Dank!

Yann Massard
BTC AG

Weitere Aspekte und Schritte

Weitere Aspekte und Schritte

Handwerkszeug

- Clean Code
- Test-driven Development
- Refactorings
- Tooling

Weitere Aspekte und Schritte

Weitere Schritte

- Legacy Code Retreat
- Coaching
- Literatur
  - Working Effectively with Legacy Code *(Michael Feathers)*