



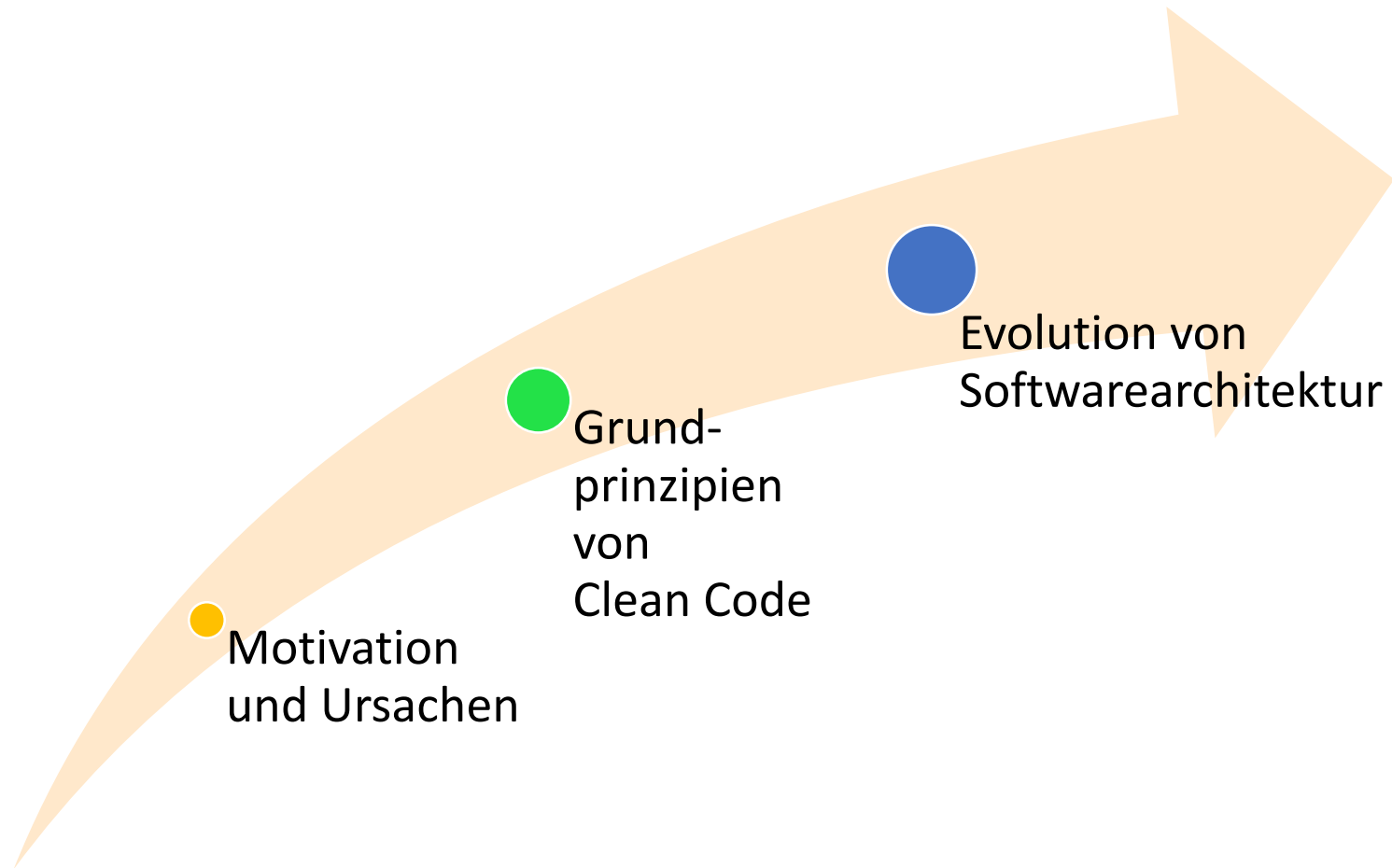
TECHNISCHE UNIVERSITÄT
CHEMNITZ

Clean Code – Clean Coder
Bearbeiter: Tobias Lang
Professur Softwaretechnik

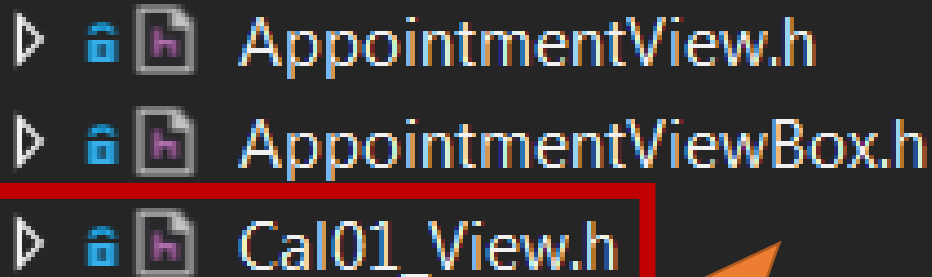
Clean Code_r

Tobias Lang

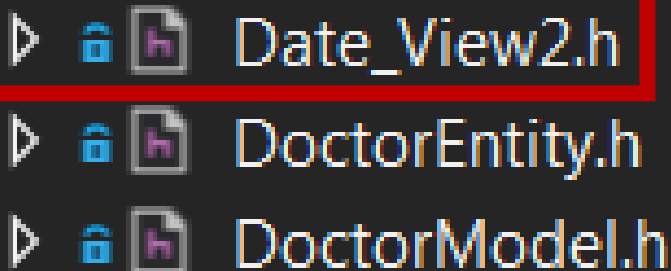
Übersicht



Motivation



▶ AppointmentView.h
▶ AppointmentViewBox.h
▶ Cal01_View.h

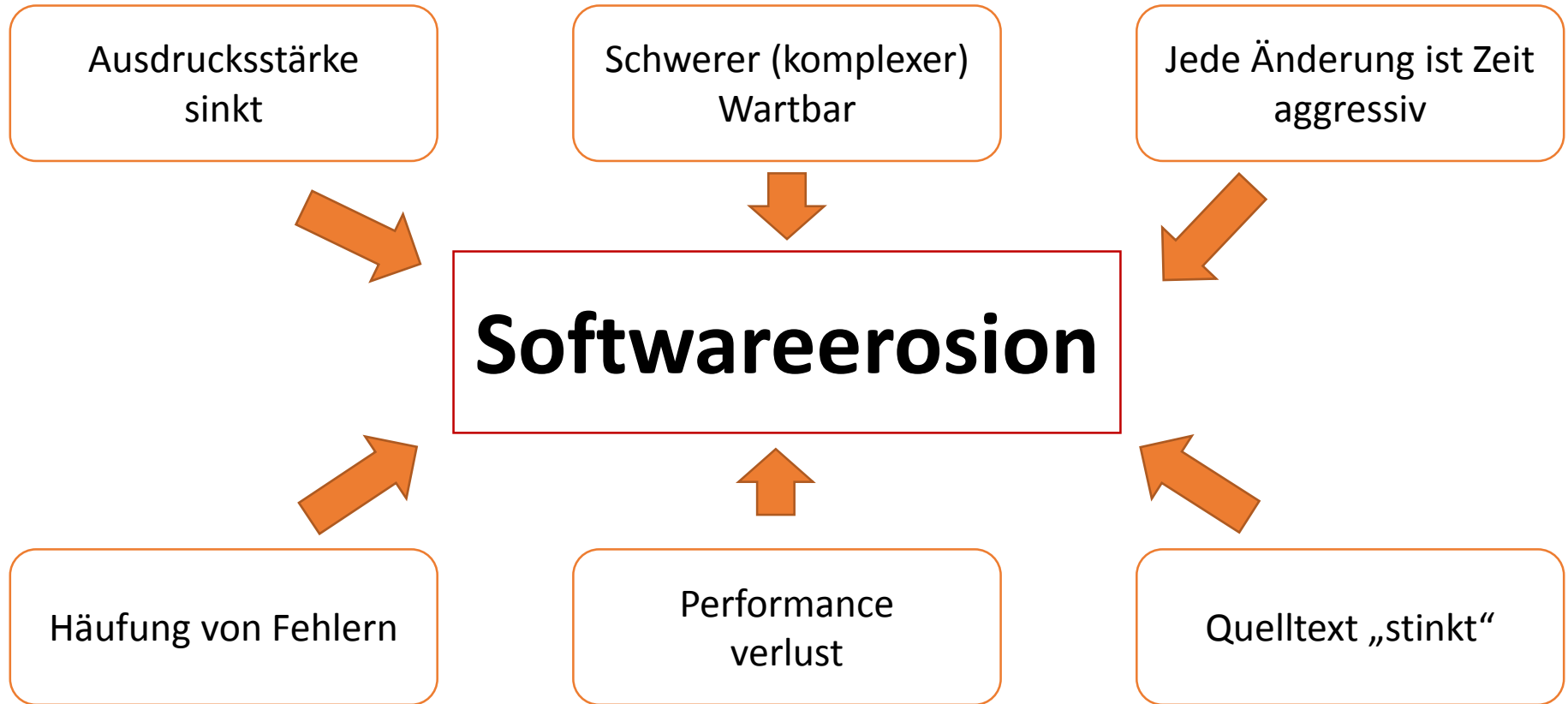


▶ Date_View2.h
▶ DoctorEntity.h
▶ DoctorModel.h

Was enthalten diese
Dateien?

Projekt von einem Studierenden
der TU Chemnitz

Motivation



„Ursachen“ Beispiele



„Zeitmangel“



Später „richtig
machen“



Unvermögen



Unkonzentriertheit



Verantwortungslos

Professionell Handeln



Pläne anpassen



Sofort refactorn



Üben!



Bereit sein



Verantwortsvoll

Clean Code „Definition“

In der Fachliteratur gibt es keine einheitliche Definition von „Clean Code“. Die folgende Definition ist daher „nur“ als weitere Perspektive zu sehen. Auf den folgenden Folien werden die „Best Practises“ und anerkannte Prinzipien von erfahrenen Programmierern vorgestellt.

Quelltext gilt dann als „sauber“, wenn er dem (inneren) Wertesystem entspricht.

Die innere Haltung besteht aus Werten, die in Ihrer Gesamtheit immer der Verantwortung

- gegenüber sich selbst
- dem Auftraggeber
- und dem Kunden

gerecht wird.

Aussagekräftige Namen

Namen sollten immer drei Fragen beantworten:



Existenzgrund?



Was leistet es?



Nutzungsart?

```
int d;
```



```
public List<int[]> getThem()
```



```
int elapsedTimeInDays;
```



```
List<int[]> getAllRoomNumbers()
```



Aussagekräftige Namen

Aussprechbare Namen verwenden

```
getAddrAttr(String add);
```

Codierung vermeiden

```
stAddress = "TU Chemnitz";  
mDb = db;  
mMessage = message;
```

Keine Mental Mappings

```
mDb = db;  
mFields.put(Constants.Mission.MESS
```

Ein Wort pro Konzept

```
getAllUsers();  
fetchAllUsers();  
retrieveAllUsers();
```

Funktionen

Funktionen sollten immer eine Aufgabe erfüllen

```
719 public String getAddressForGoogleMap(final String id) {  
720     final String[] arguments = {id};  
721     final String sql = "SELECT street, city FROM missions WHERE _id = ? LIMIT 1;";  
722     Cursor cursor = mReadableDatabase.rawQuery(sql, arguments);  
723  
724     String street = cursor.getString(0);  
725     street = (street == null) ? "" : street.replace(",", " ");  
726  
727     String city = cursor.getString(1);  
728     city = (city == null) ? "" : city.replace(",", " ");  
729  
730     return street + ", " + city;  
731 }
```

1 ⚡

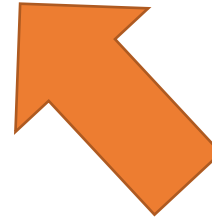
2 ⚡

3 ⚡

Funktionen

Eine Funktion sollte keine Seiteneffekte haben

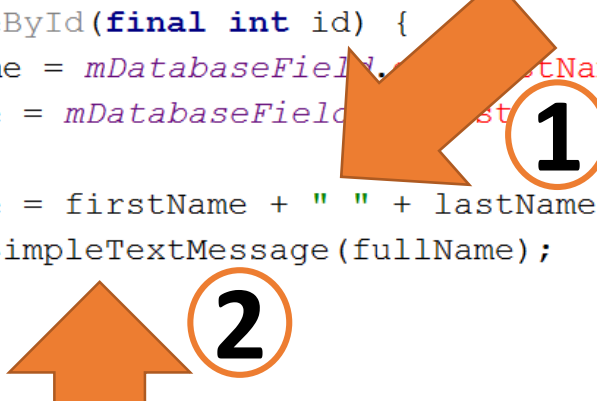
```
109 public String getFullNameById(final int id) {  
110     final String firstName = mDatabaseField.getFirstNameById(id);  
111     final String lastName = mDatabaseField.getLastNameById(id);  
112  
113     final String fullName = firstName + " " + lastName;  
114     calendarService.sendSimpleTextMessage(fullName);  
115  
116     return fullName;  
117 }
```



Funktionen

Eine Abstraktionsebene pro Funktion

```
109 public String getFullNameById(final int id) {  
110     final String firstName = mDatabaseField.firstNameById(id);  
111     final String lastName = mDatabaseField.lastNameById(id);  
112  
113     final String fullName = firstName + " " + lastName;  
114     calendarService.sendSimpleTextMessage(fullName);  
115  
116     return fullName;  
117 }
```



Funktionen

Eine Funktion sollte wenig Funktionsargumente haben

Nur eine Aufgabe?

```
final String description = getKeywordDescription("F19", true);
```

```
public String getKeywordDescription(String keyword, boolean isForce) {
```

```
getKeywordDescription(String keyword) | getForceDescription(String keyword)
```

```
public paintCircle(int x, int y, int radius, Color surface)
```

```
public paintCircle(Circle currentCircle, Color surface)
```

Fehlerbehandlung

```
private void tearDown() {  
    Device deviceHandler = new Device();  
  
    if (deviceHandler != null) {  
        deviceHandler.shutdown();  
  
        Status deviceStatus = deviceHandler.getStatus();  
  
        if (deviceStatus != null) {  
            if (deviceStatus == INVALID) {  
                log.error("Device is still up and running");  
                quit();  
            }  
  
            log.info("The device has been successfully shut down");  
        }  
    }  
}
```

Geschäftslogik

Fehlerbehandlung

Fehlerbehandlung

Fehlerbehandlung an sich ist eine Aufgabe

```
private Database(Context context) {  
    super(context, Constants.DATABASE_NAME, null, Constants.DATABASE_VERSION);  
    mContext = context;  
    mReadableDatabase = getReadableDatabase();  
  
    if (isNewDatabase) {  
        mReadableDatabase.close();  
  
        try {  
            copyAssetToDatabase();  
        } catch (Exception error) {  
            Toast.makeText(mContext, error.toString(), Toast.LENGTH_LONG).show();  
        }  
  
        mReadableDatabase = this.getReadableDatabase();  
        isNewDatabase = false;  
    }  
}
```

Sollte eine eigene
Funktion sein

Fehlerbehandlung

Statt Fehlercodes oder NULL zurückzugeben, sollten Fehler geworfen werden

```
private void shutDown() {  
    try {  
        Device deviceHandler = new Device();  
        deviceHandler.shutDown();  
        log.info("The device has been successfully shut down");  
    }  
    catch (Exception error) {  
        log.error("The device is still up and running");  
    }  
}  
  
public void shutDown() {  
    ...  
    throw new DeviceShutDownError("Invalid device handler");  
    ...  
}
```

Geschäftslogik

Fehlerbehandlung

Fehlerbehandlung

Special Case Objects können eine Alternative für Ausnahmen sein

```
public List<Member> getMemberList() {  
    List<Member> memberList = Collections.emptyList();  
    return memberList;  
}
```

```
public Member getMember() {  
    Member member = new Member();  
    member.setName("John Doe");  
  
    return member;  
}
```



Standardwert

Kommentare

Schlechter Code sollte nicht kommentiert werden, sondern korrigiert!



Mehrwert



Ausdrucksstärke erhöhen

```
/**
 * TODO: Refactor this method
 *
 // Pattern for hh:mm:ss
Pattern.compile("\\d{2}:\\d{2}:\\d{2}");

// This method takes at least 3 seconds
Cursor fileCursor = Cursor.openFile();
```

Eine Konstante wäre hier besser

Kommentare

```
// This method is very expensive  
// Cursor fileCursor = openFile();  
// Cursor fileCursor = openFile(filePath);  
Cursor fileCursor = openFile(filePath + fileExtension);
```

Müll

Geschwätz

```
141 /**  
142  * Open the database  
143  */  
144 public void openDatabase() {  
145     if (!mReadableDatabase.isOpen()) {  
146         mReadableDatabase = getReadableDatabase();  
147     }  
148 }
```

Klassen

Klassen sollten die „Step-Down-Regel“ erfüllen (UM-ZU-Absätze)

```
class Database {
```

```
    public static final String username = "root";  
    private static final String password = "123456";
```

```
    public static void setUsername(String name) {...}
```

```
    public void connectToDatabase() {
```

Warum public zuerst?

```
        private static void buildConnectionSQL() {...}
```

```
        private void closeConnection() {}  
}
```

Step-down-Regel

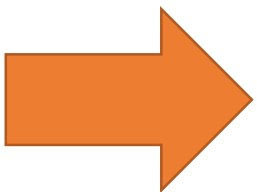
Klassen

Klassen sollten dem Single-Responsibility-Prinzip folgen (eine Verantwortlichkeit)

```
public class TextParser {  
    private static final String databaseUser = "root";  
    private static final String databasePassword = "123456";
```

Niedrige Kohäsion

```
    private static final Pattern time = Pattern.compile("\\d{2}:\\d{2}:\\d{2}");  
    private static final Pattern date = Pattern.compile("\\d{2}\\.\\d{2}\\.\\d{4}");  
  
    public List<String> parseText(String text) {...}  
    public void saveTextInDatabase(String text) {...}  
}
```



Klassen sollten eine lose Kopplung und eine starke Kohäsion aufweisen

Wartbarkeit (Modifiability)

Gesetz von Demeter („Prinzip der Verschwiegenheit“)

```
string path = context.options.path.absolute;  
File* logFile = new File(path);
```

Train Wreck



```
string absolutePath = context.getAbsolutePath();
```



```
context.createLog();
```

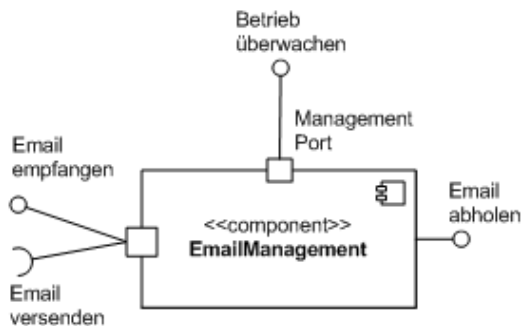
Metrik

Dauer der Änderungen
in Personenstunden



Wartbarkeit

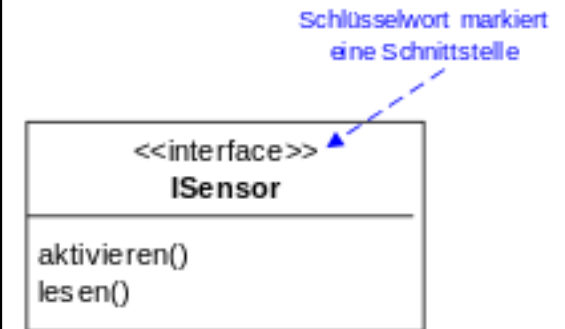
Modularer- oder Komponentenbasierter Aufbau



Automatisierte und ausführbare Tests

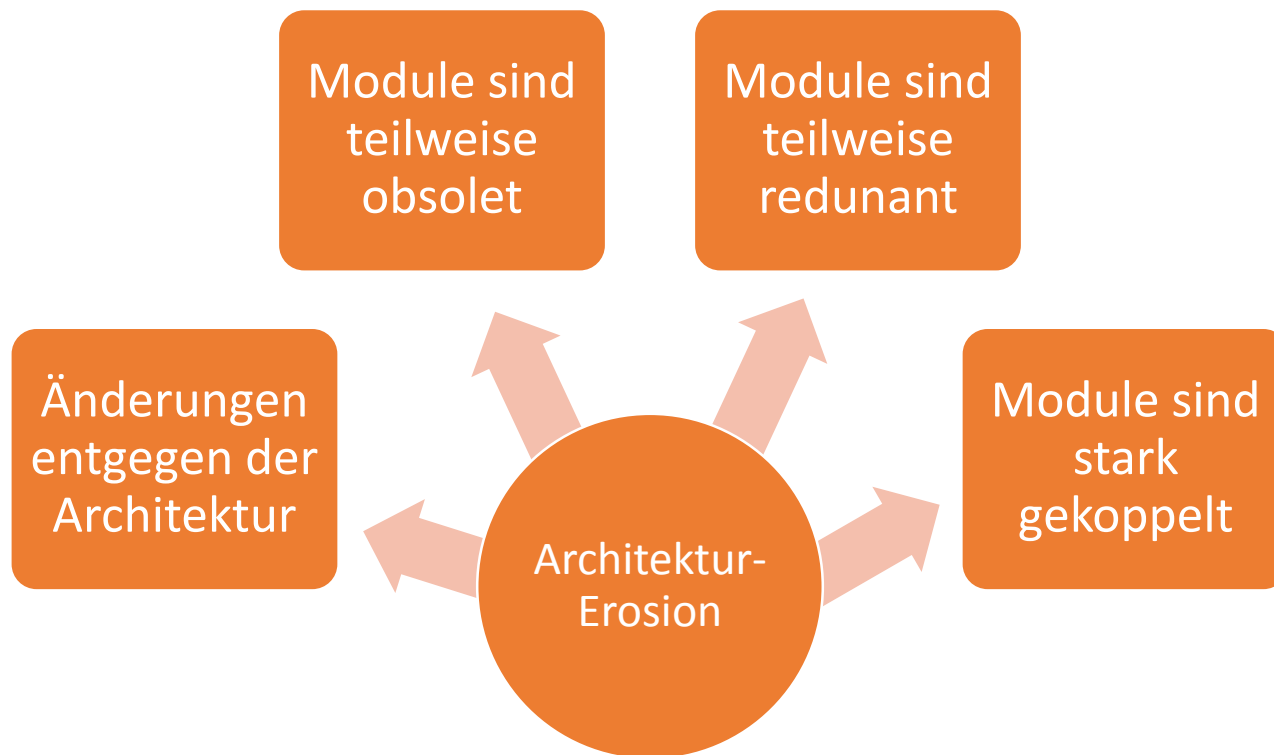


Klar definierte Interfaces

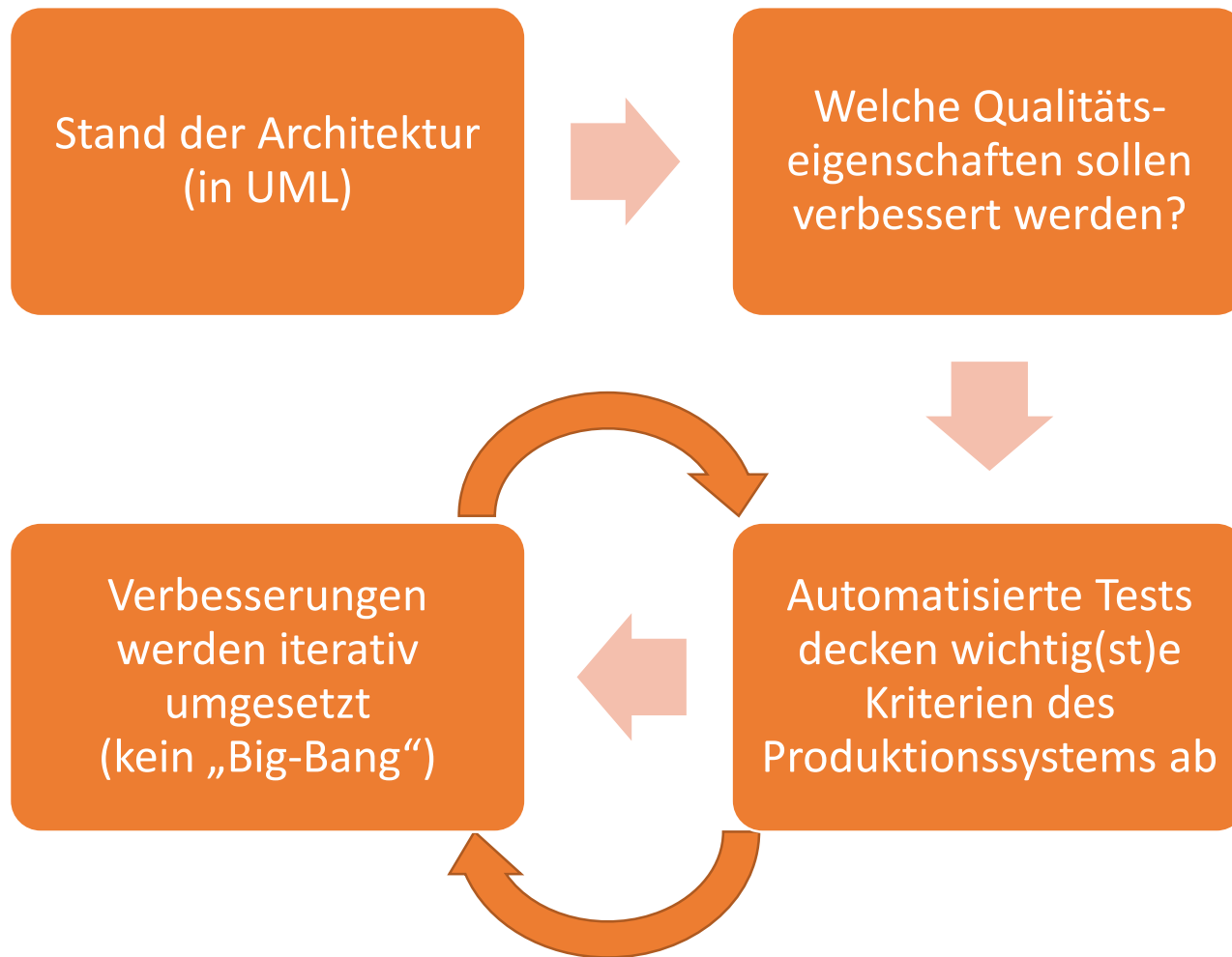


Erosion von Softwarearchitektur

Starke Kohäsion und lose Kopplung sind auf Architekturebene unentbehrlich

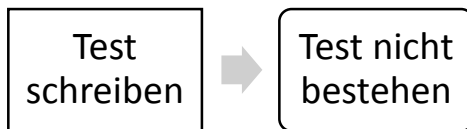


Evolution von Softwarearchitektur



Testgetriebene Entwicklung

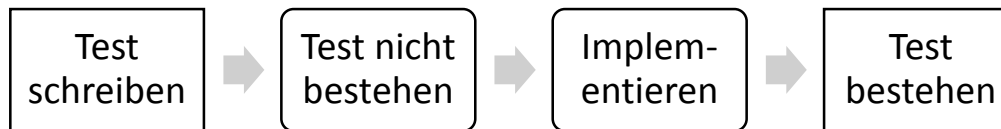
Test Driven Development (TDD)



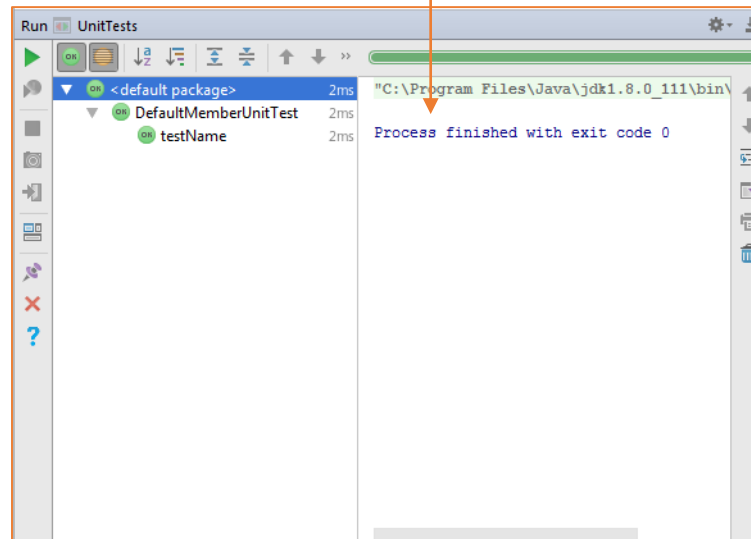
```
public class DefaultMemberUnitTest {  
    private Member defaultMember;  
  
    @Before  
    public void setUp() {  
        defaultMember = new Member();  
    }  
  
    @Test  
    public void testName() {  
        String memberName = defaultMember.getName();  
        assertEquals("Default", memberName);  
    }  
}
```

```
String memberName = defaultMember.getName();  
                                ^  
symbol:   method getName()  
location: variable defaultMember of type Member  
1 error  
FAILED  
  
FAILURE: Build failed with an exception.  
  
* What went wrong:  
Execution failed for task ':app:compileDebugUnitTestJavaWithJavac'.  
> Compilation failed; see the compiler error output for details.  
  
* Try:  
Run with --stacktrace option to get the stack trace. Run with --info or  
BUILD FAILED  
  
Total time: 1.48 secs
```

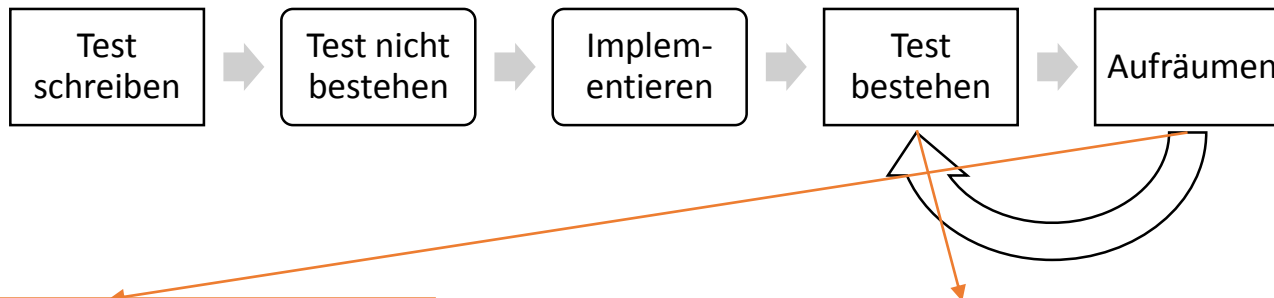
Testgetriebene Entwicklung



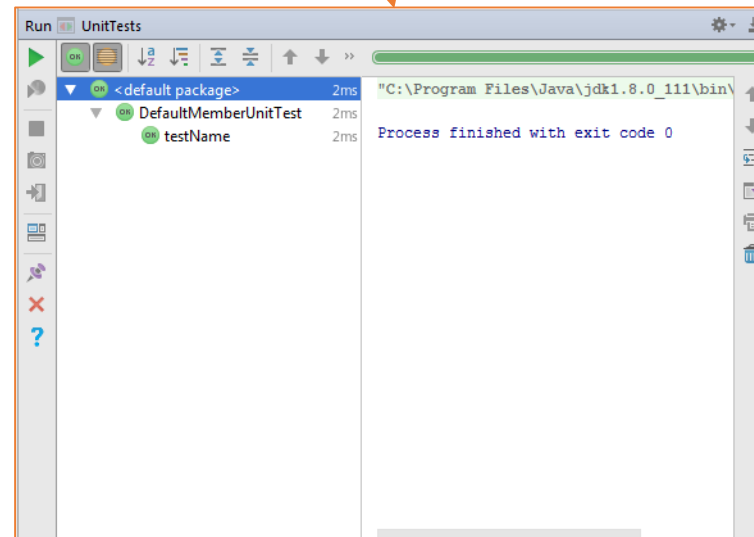
```
class Member {  
    public String getName() {  
        return "Default";  
    }  
}
```



Testgetriebene Entwicklung

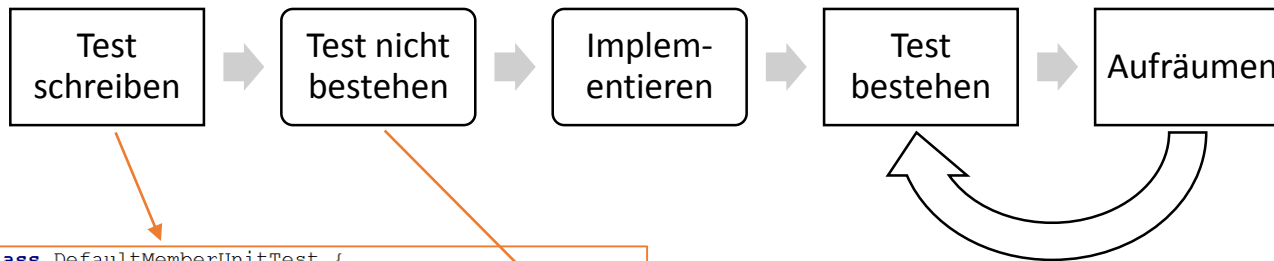


```
class Member {  
    private String name = "Default";  
  
    public String getName() {  
        return name;  
    }  
}
```



Testgetriebene Entwicklung

Test Driven Development (TDD)



```

public class DefaultMemberUnitTest {
    private Member defaultMember;

    @Before
    public void setUp() {
        defaultMember = new Member();
    }

    @Test
    public void testName() {
        String memberName = defaultMember.getName();
        assertEquals("Default", memberName);
    }

    @Test
    public void testFullName() {
        String memberFullName = defaultMember.getFullName();
        assertEquals("John Default", memberFullName);
    }
}
  
```

```

        String memberFullName = defaultMember.getFullName();
                                           ^
        symbol:   method getFullName()
        location: variable defaultMember of type Member
        1 error

FAILED

FAILURE: Build failed with an exception.

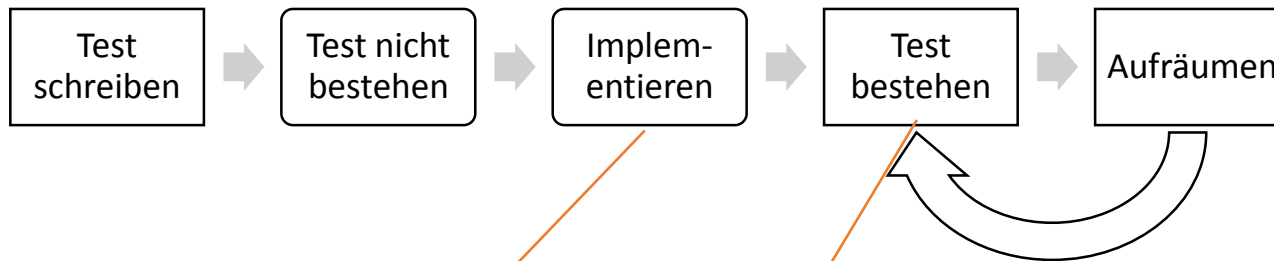
* What went wrong:
Execution failed for task ':app:compileDebugUnitTestJavaWithJavac'.
> Compilation failed; see the compiler error output for details.

* Try:
Run with --stacktrace option to get the stack trace. Run with --info

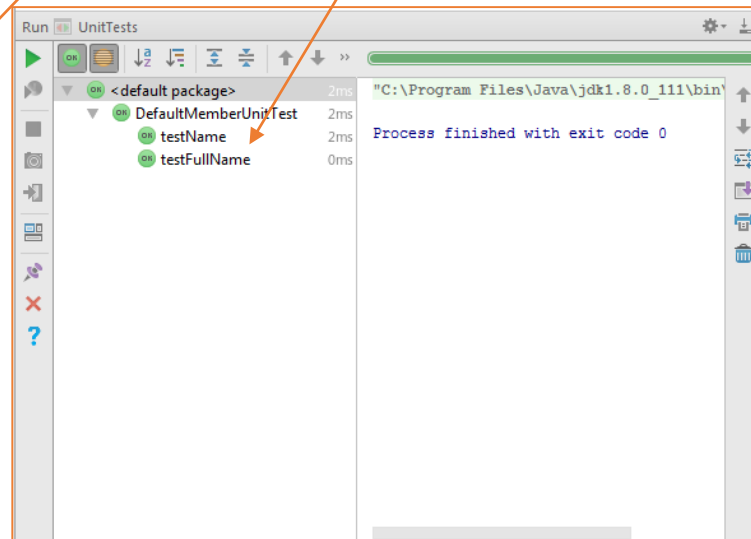
BUILD FAILED

Total time: 0.972 secs
  
```

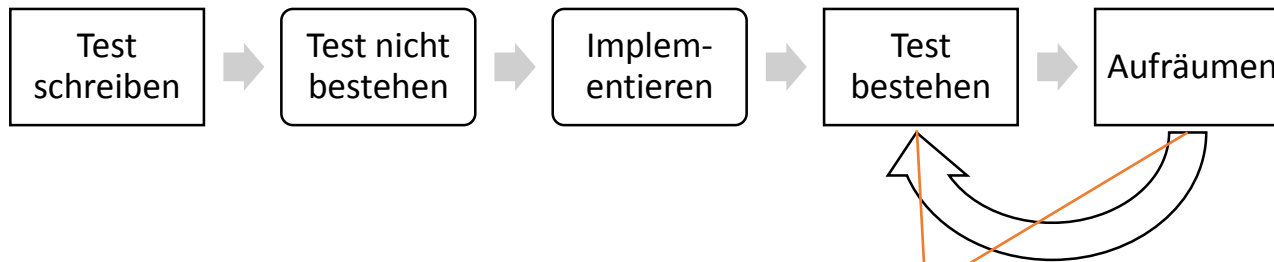
Testgetriebene Entwicklung



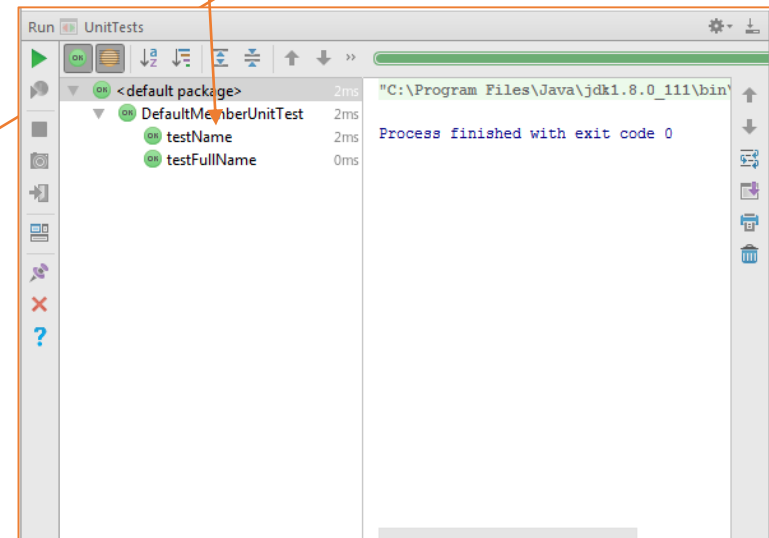
```
class Member {  
    private String name = "Default";  
  
    public String getName() {  
        return name;  
    }  
  
    String getFullName() {  
        return "John " + name;  
    }  
}
```



Testgetriebene Entwicklung



```
class Member {  
    private static final String DELIMITER = " ";  
    private String firstName = "John";  
    private String name = "Default";  
  
    public String getName() {  
        return name;  
    }  
  
    String getFullName() {  
        return firstName + DELIMITER + name;  
    }  
}
```



Testgetriebene Entwicklung

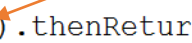
Abhängigkeiten werden durch Mocks/Spys/Stubs ersetzt

Stub → Ersetzt ein Objekt, Methode oder Attribut mit einer Konstanten (ohne Bedingungen)

Mock → Ersetzt ein Objekt, Methode oder Attribut mit einer Konstanten (mit Bedingungen)

```
@Test
public void testSalary() {
    Salary salary = mock(Salary.class);
    when(salary.calculateMonthlyBonus(1.0)).thenReturn(10.0);

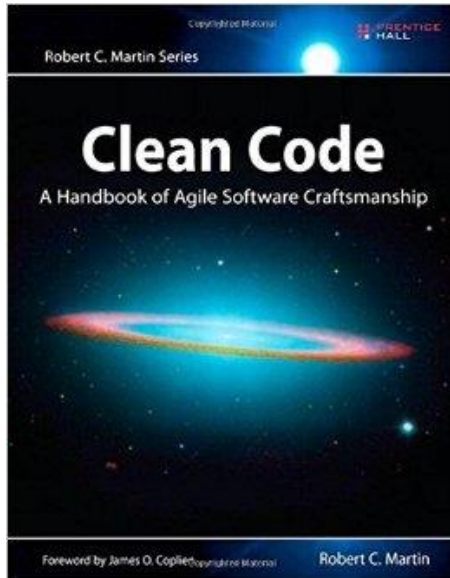
    Double memberSalary = defaultMember.getSalary();
    assertRange(10.0, memberSalary);
}
```



Nur bei dem Argumentwert 1.0
wird 10.0 zurückgegeben

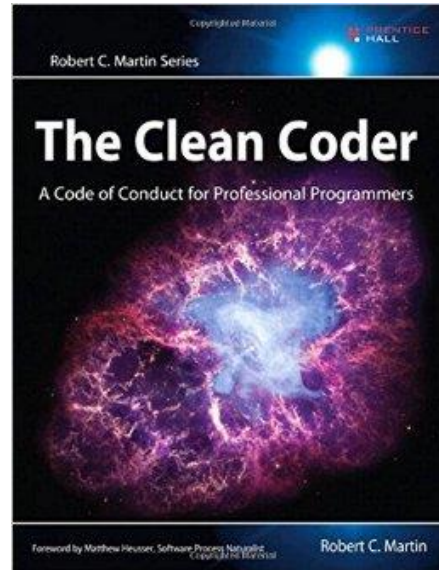
Spy → Überwacht die Verwendung des Ersatzes und kann als Stub/Mock dienen

Ressourcen



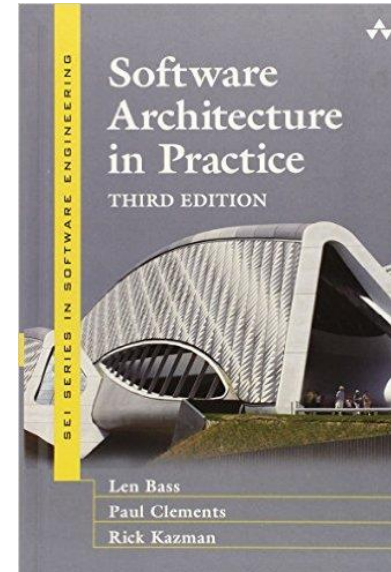
Clean Code

Refactoring, Patterns,
Testen und Techniken für
sauberen Code



Clean Coder

Verhaltensregeln für
professionelle
Programmierer



Software Architecture in Practice



Handbuch der Software- Architektur

Werkzeuge

IntelliJ IDEA (IDE)



YouTrack (PM)



Jira (PM)



Bitbucket (git)



Upsource (Code review)



Junit (Test Framework)



Gradle (Build Management)



Draw.io (Zeichnungen)

