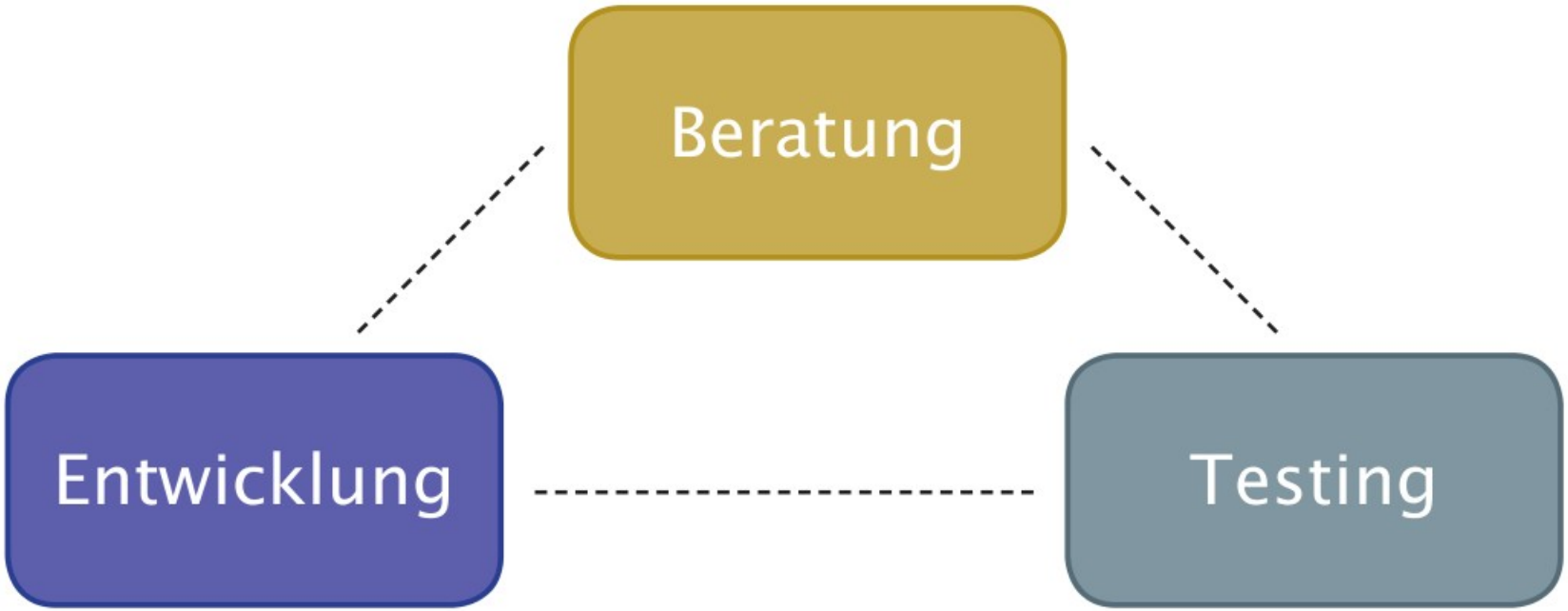
Wie wird mein Code testbar?

Referent: David Völkel

4A Solutions

Any Application
Anytime
Anywhere

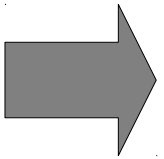
http://commons.wikimedia.org/wiki/File:Pit_crew_Hudson_Valley.JPG
http://commons.wikimedia.org/wiki/File:CarService.JPG



David Völkel

4A Solutions

Any Application
Anytime
Anywhere



http://commons.wikimedia.org/wiki/File:Pit_crew_Hudson_Valley.JPG
http://commons.wikimedia.org/wiki/File:CarService.JPG

Überblick

* Testbarkeit
* TDD vs. Legacy
* Isolation
* Ciao „*accidental complexity*“!

Testbarkeit

Kriterien

* operability
* decomposability
* observability
* controllability

Design for Testability (DfT)

Designziel

* wenig Testaufwand

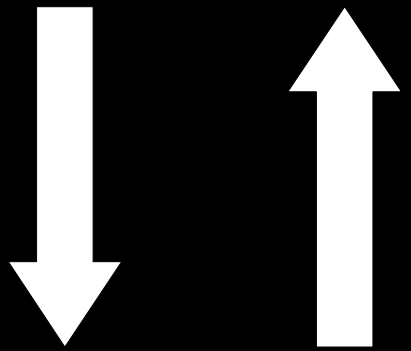
Ursprung E-Technik

* Testschnittstellen

Test Driven Development
vs.
Legacy

Test Driven Development

write failing test



make the test

pass

(refactor)

http://commons.wikimedia.org/wiki/File:Bluebird_land_speed_record_car_1935_rc10413.jpg

Test Driven Development

write failing test

(refactor)

make the test pass

(refactor)

http://commons.wikimedia.org/wiki/File:Bluebird_land_speed_record_car_1935_rc10413.jpg

Test Driven Development

write failing test

(refactor)

make the test pass

(refactor)

Externe Qualität
Funktional

http://commons.wikimedia.org/wiki/File:Bluebird_land_speed_record_car_1935_rc10413.jpg

Test Driven Development

write failing test

(refactor)

make the test pass

(refactor)

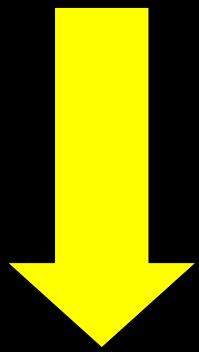
Interne Qualität
=> Test Driven Design

Externe Qualität
Funktional

Test Driven Design

* **Drive** Design
  - kontinuierlich
  - minimalistisch (YAGNI)
  - **testbar** <=> gutes Design

Test



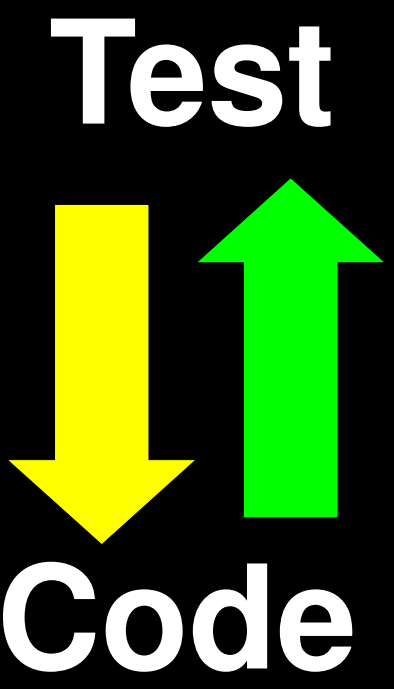
Code

http://commons.wikimedia.org/wiki/File:Bluebird_land_speed_record_car_1935_rc10413.jpg

Test Driven Design

* **Drive** Design
  - kontinuierlich
  - minimalistisch (YAGNI)
  - testbar <=> gutes Design
* **Feedback** auf Design
  - „Listen to your tests“!

(Zitat Johansen, 2010)



http://commons.wikimedia.org/wiki/File:Bluebird_land_speed_record_car_1935_rc10413.jpg

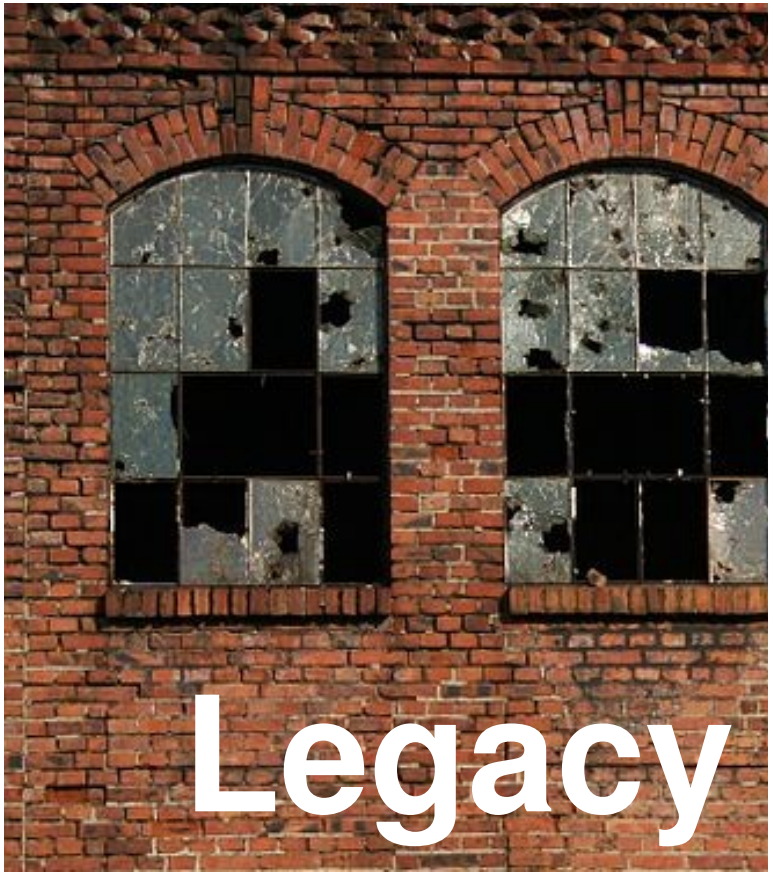
Legacy Code

= keine Tests

schlecht testbares Design

* eigener Code
* 3rd Party Code (Frameworks)

http://commons.wikimedia.org/wiki/File:Kopalnia_Wieczorek_rozbite_okna_12.08.jpg?uselang=de

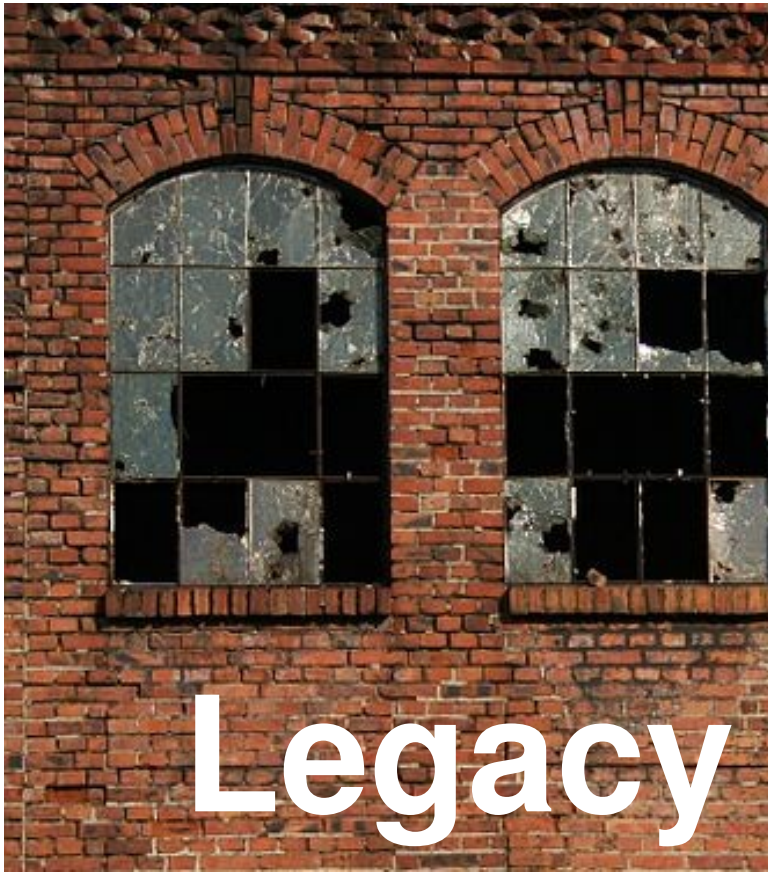


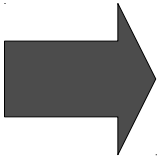
Renovierung

Doppelt schwierig

* Noch wenig KnowHow im Team
* Schwer testbarer Code

http://commons.wikimedia.org/wiki/File:Kopalnia_Wieczorek_rozbite_okna_12.08.jpg?uselang=de
http://commons.wikimedia.org/wiki/File:Bluebird_land_speed_record_car_1935_rc10413.jpg





Renovierung

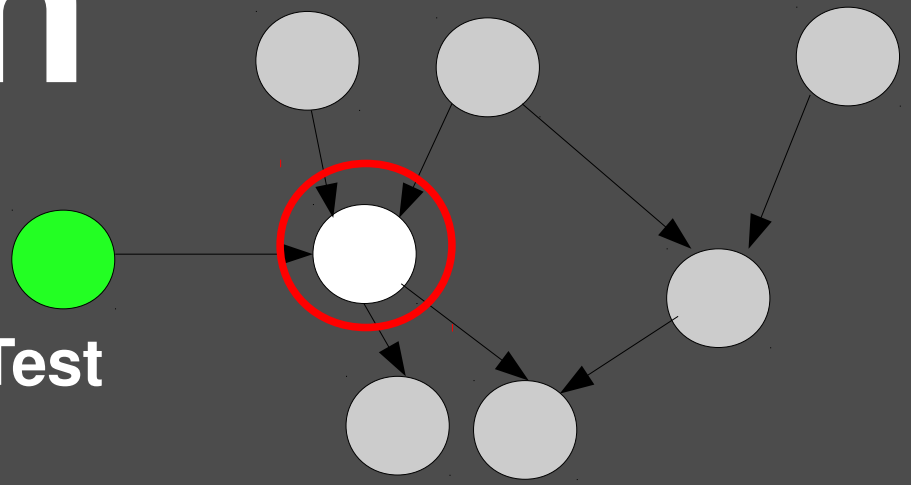
* „Lazy“
* Fixierung mit Systemtests
* Refactoring
* Unittests für neue Features

Nach Feathers 2004

http://commons.wikimedia.org/wiki/File:Kopalnia_Wieczorek_rozbite_okna_12.08.jpg?uselang=de
http://commons.wikimedia.org/wiki/File:Bluebird_land_speed_record_car_1935_rc10413.jpg

Isolation

Isolation

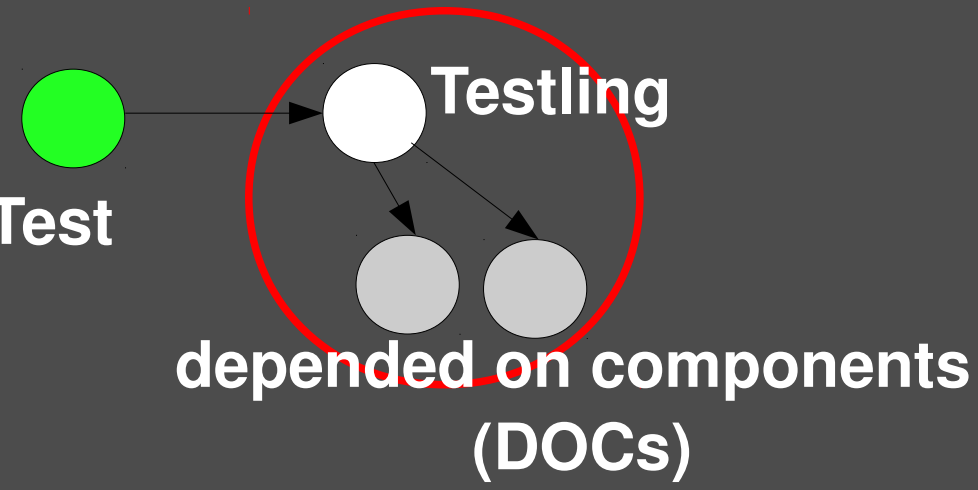


Ziele

* Klarer Fokus
  - Testaufwand sinkt
  - Fehlerlokalisierung einfacher
* Performanz

front door

* KEINE Isolation
* round trip tests
* state verification



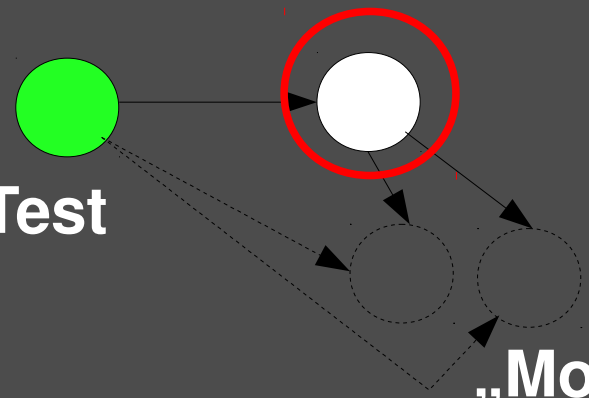
Begriffe nach Meszaros 2007

http://commons.wikimedia.org/wiki/File:Stork_Hotel_door.jpg

back door

Unittests gegen

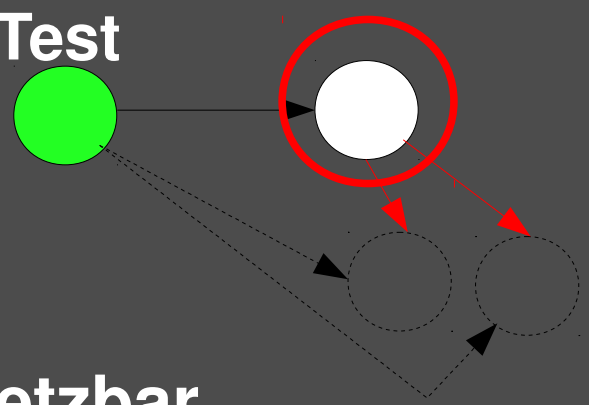
- **Blätter**
- **„Units“ mit DOCs**



Begriffe nach Meszaros 2007

http://commons.wikimedia.org/wiki/File:Dummy_of_a_police_car.jpg

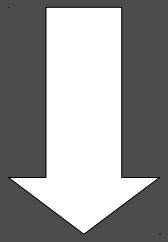
Isolierbarkeit



Test Doubles

Problem:

Test Doubles nicht setzbar

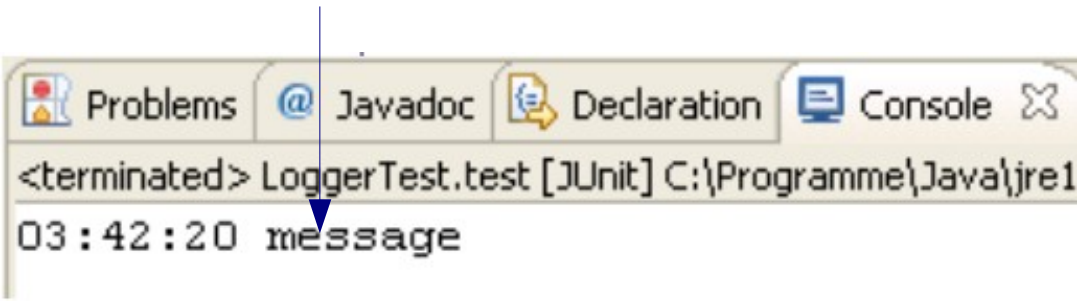


Refactorings

Isolierbar

Beispiel: Logger

```
public class Logger {

    public void log(String message) {
        Date timeStamp = new Date();

        String formatedTimeStamp = new SimpleDateFormat("hh:mm:ss")
                .format(timeStamp);

        System.out.println(formatedTimeStamp + " " + message);
    }

}
public class LoggerTest {
    @Test
    public void test() throws Exception {
        Logger logger = new Logger();
        logger.log("message");
    }
}
```

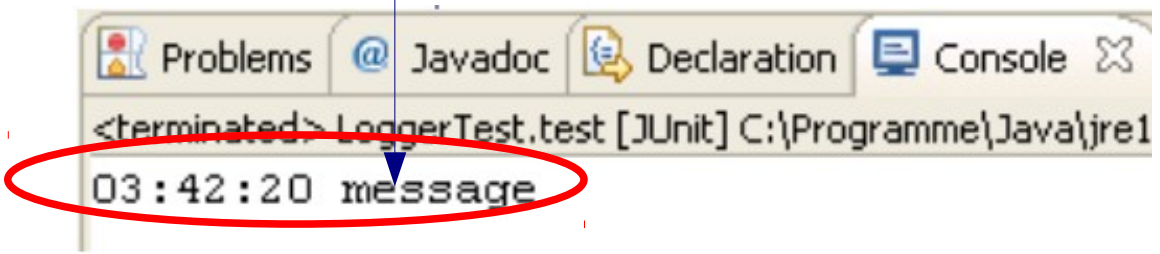


Probleme

```
public class Logger {

    public void log(String message) {
        Date timeStamp = new Date();

        String formatedTimeStamp = new SimpleDateFormat("hh:mm:ss")
                .format(timeStamp);

        System.out.println(formatedTimeStamp + " " + message);
    }

}
public class LoggerTest {
    @Test
    public void test() throws Exception {
        Logger logger = new Logger();
        logger.log("message");
    }
}
```

Controllability (INPUT)

Observability (OUTPUT)



Ursachen

```
public class Logger {

    public void log(String message) {
        Date timeStamp = new Date();

        String formatedTimeStamp = new SimpleDateFormat("hh:mm:ss")
                .format(timeStamp);

        System.out.println(formatedTimeStamp + " " + message);
    }

}
```

statische Abhängigkeiten

* Konstruktoraufrufe
* Singletons

Statischer Aufruf => Objekte

```
public class Logger {

    public void log(String message) {
        Date timeStamp = timeSource.getTime();

        String formatedTimeStamp = new SimpleDateFormat("hh:mm:ss")
                .format(timeStamp);

        String line = formatedTimeStamp + " " + message;
        lineAppender.append(line);
    }
```

Statischer Aufruf => Objekte

```
public class Logger {

    public void log(String message) {
        Date timeStamp = timeSource.getTime();

        String formatedTimeStamp = new SimpleDateFormat("hh:mm:ss")
                .format(timeStamp);

        String line = formatedTimeStamp + " " + message;
        lineAppender.append(line);
    }
```

Klasse <=> Interface

* weniger Aufwand
* „don't mock concrete classes“

* Abhängigkeiten expliziter
  geringer (ISP)
* Semantik durch Rollen

Zitat aus Freeman / Pryce 2009

„Setzbarkeit“

```
public class Logger {

    public void log(String message) {..

    private TimeSource timeSource;
    private LineAppender lineAppender;

    public Logger(TimeSource timeSource, LineAppender lineAppender) {
        this.timeSource = timeSource;
        this.lineAppender = lineAppender;
    }

}
```

Dependency Injection

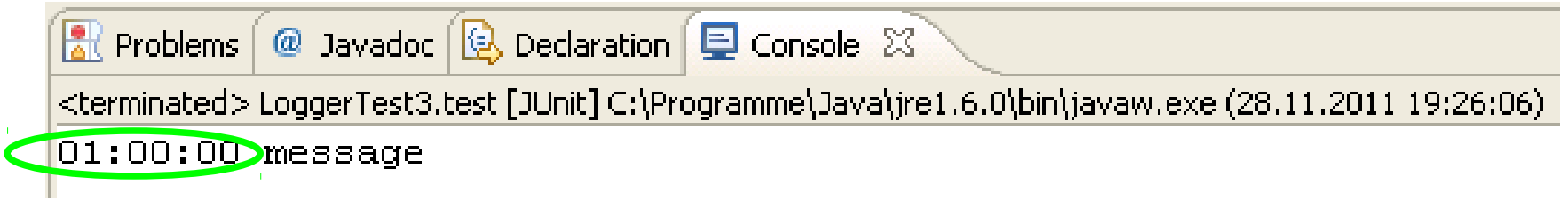
* DOCs setzbar
* Konstruktor vs. Setter
* kein `static`

http://commons.wikimedia.org/wiki/File:Syringe_Glove_01.jpg

Stub mit Bordmittel

```
@Test
public void test() throws Exception {
    TimeSource timeSourceStub = new TimeSource() {
        public Date getTime() {
            return TIME_01_00_00;
        }
    };
    Logger logger = new Logger(timeSourceStub, CONSOLE_LINE_APPENDER);

    logger.log("message");
}
```

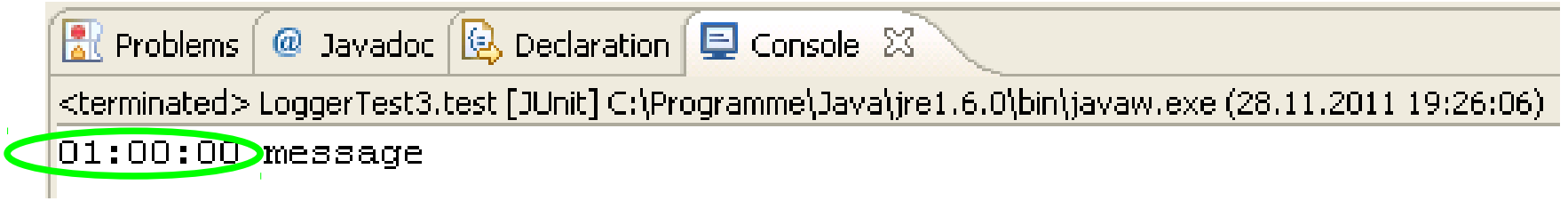
Controllability (INPUT)



* Input kontrollierbar
* etwas sperrig

Stub mit Mockframework

```
@Test
public void test() throws Exception {
    TimeSource timeSourceStub = mock(TimeSource.class);
    stub(timeSourceStub.getTime()).toReturn(TIME_01_00_00);

    Logger logger = new Logger(timeSourceStub, CONSOLE_LINE_APPENDER);

    logger.log("message");
}
```



einfacher bei breiteren Interfaces

Test Spy mit Bordmitteln

```
@Test
public void test() throws Exception {
    // setup
    TimeSource timeSourceStub = mock(TimeSource.class);
    stub(timeSourceStub.getTime()).toReturn(TIME_01_00_00);

    LineAppenderTestSpy lineAppenderTestSpy = new LineAppenderTestSpy();

    Logger logger = new Logger(timeSourceStub, lineAppenderTestSpy);

    // execute
    logger.log("message");

    // verify
    assertEquals("01:00:00 message", lineAppenderTestSpy.line);
}

public class LineAppenderTestSpy implements LineAppender {

    public String line;

    @Override
    public void append(String line) {
        this.line = line;
    }

}
```

Observability (OUTPUT)

Test Spy mit Mockframework

```
@Test
public void test() throws Exception {
    // setup
    TimeSource timeSourceStub = mock(TimeSource.class);
    stub(timeSourceStub.getTime()).toReturn(TIME_01_00_00);

    LineAppender lineAppenderTestSpy = mock(LineAppender.class);

    Logger logger = new Logger(timeSourceStub, lineAppenderTestSpy);

    // execute
    logger.log("message");

    // verify
    verify(lineAppenderTestSpy).append("01:00:00 message");
}
```

* knapp
* „behavior verification“

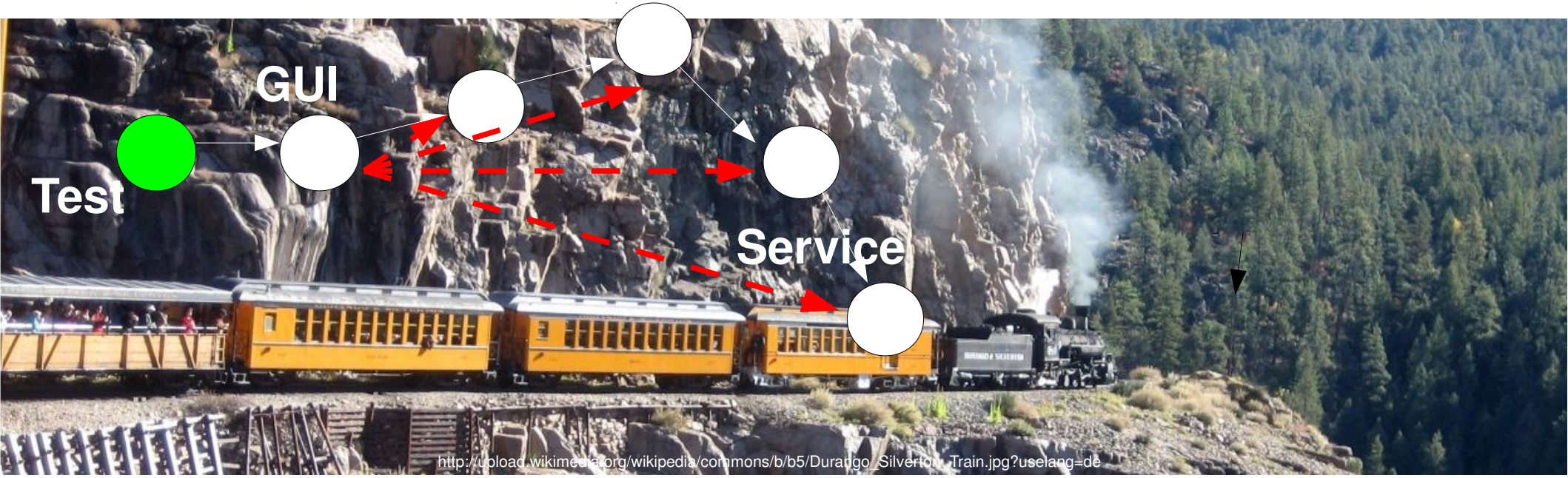
Transitive Abhängigkeiten

```
public class CustomerGUI {

    public void renderCustomers() {

        service = application.getCustomerModule().getBusinessLayer().getCustomerSearchService();

        customers = service.searchAllCustomers();
                ...
    }
```

„trainwreck code“

Begriff von Freeman / Price 2009

Problem: Implizite Abhängigkeiten

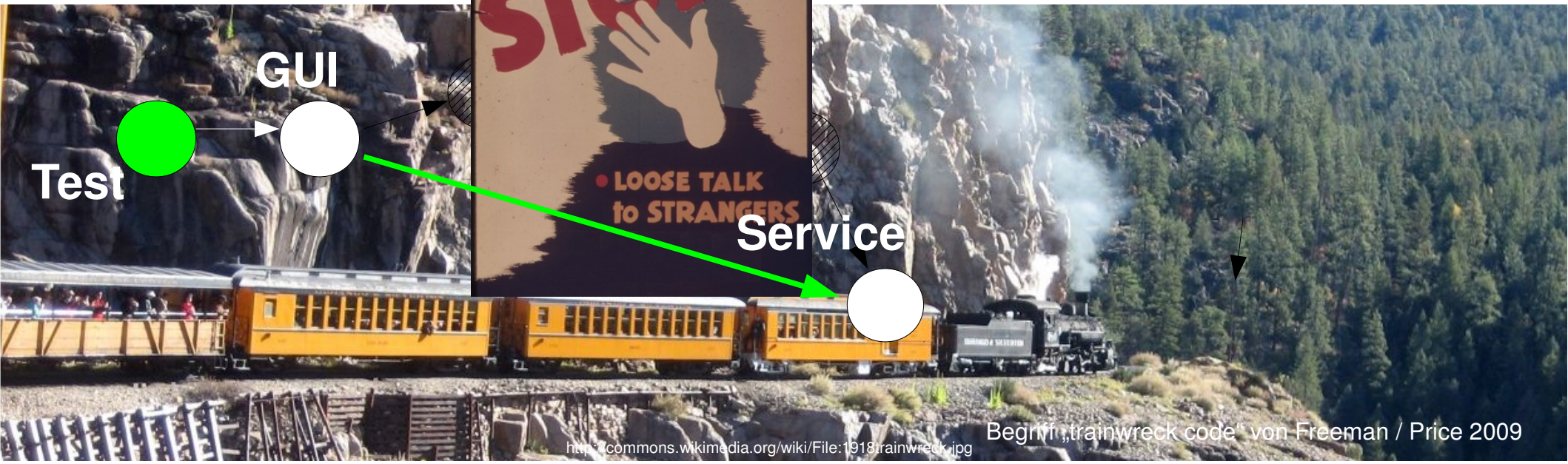


http://upload.wikimedia.org/wikipedia/commons/b/b5/Durango_Silverton_Train.jpg?uselang=de

Transitive Abhängigkeiten

```
public class CustomerGUI {

    private CustomerSearchService service;

    public CustomerGUI(CustomerSearchService service) {
        this.service = service;
    }

    public void renderCustomers() {
        customers = service.searchAllCustomers();
    }
```

* Abhängigkeit explizit
* nur zu „Nachbarn“



http://commons.wikimedia.org/wiki/File:1918trainwreck.jpg

Begriff „trainwreck code“ von Freeman / Price 2009

http://commons.wikimedia.org/wiki/File:Syringe_Glove_01.jpg

Ciao „*accidental complexity*“!

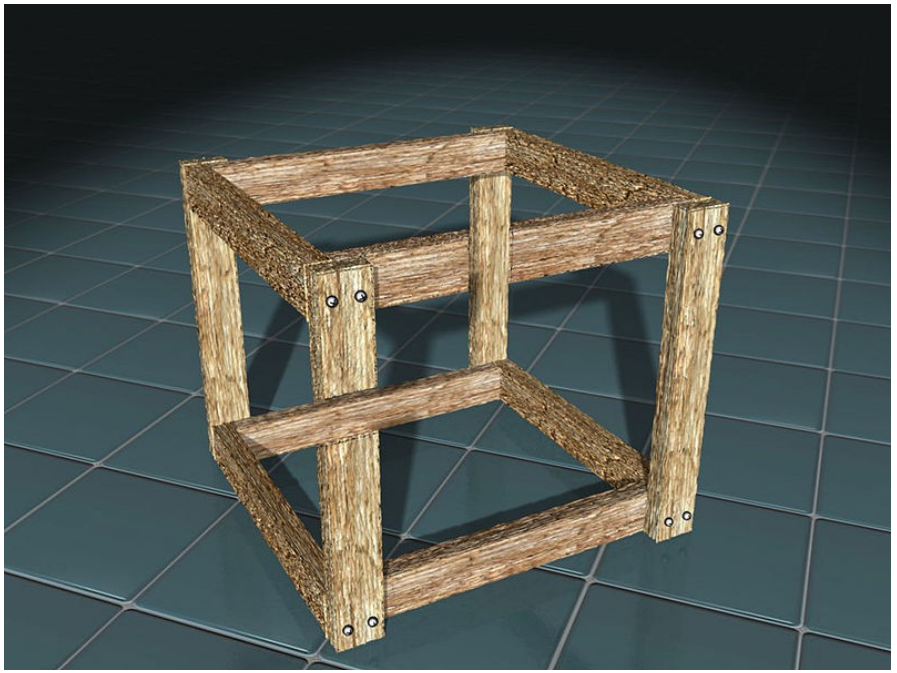
Ciao „*accidental complexity*“!

Problem:

* komplexe Setups
* schlechter Fokus

http://commons.wikimedia.org/wiki/File:US_Navy_040619-N-8704K-001_Boatswain_Mate_3rd_Class_Joanna_Saldana_cuts_the_rope_off_a_span_wire_aboard_the_aircraft_carrier_USS_John_F._Kennedy_%28CV_67%29_during_an_underway_replenishment_.jpg?uselang=de

Logische Fehler



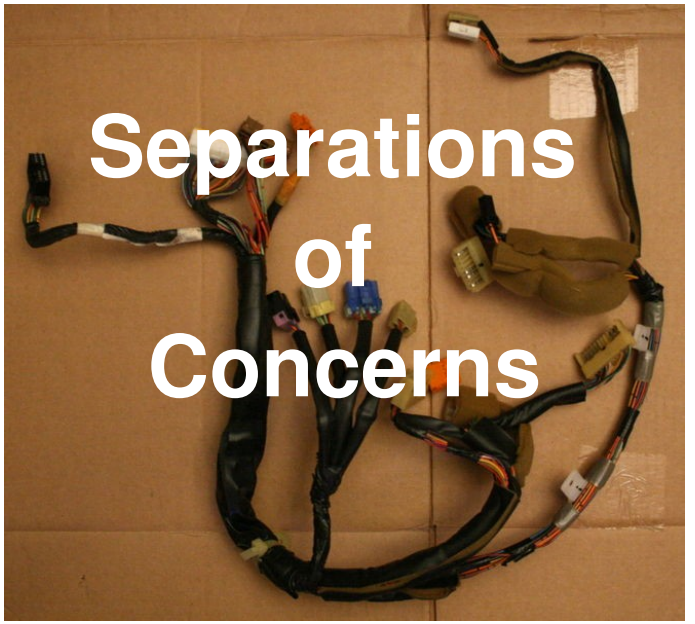
* Schichten, z.B.
  Prod-Code => Test-Code
* Zyklen

… und viele andere mehr!

http://de.wikipedia.org/w/index.php?title=Datei:Lattenkiste_a.jpg&filetimestamp=20060723120820

decomposability

Alles hängt von allem ab



System kleinschneiden

http://commons.wikimedia.org/wiki/File:Crossroads.jpg

Unfokussiert

```
public void log(String message) {
    Date timeStamp = timeSource.getTime();

    String formatedTimeStamp = new SimpleDateFormat("hh:mm:ss")
            .format(timeStamp);

    String line = formatedTimeStamp + " " + message;
    lineAppender.append(line);
}

@Test
public void testLog() throws Exception {
    ...

    // verify
    verify(lineAppenderTestSpy).append("01:00:00 message");
}
```

2 Aspekte auf einmal

Extract Class Refactoring

```
public void log(String message) {
    Date timeStamp = timeSource.getTime();

    String formatedTimeStamp = timeStampFormatter.format(timeStamp);

    String line = formatedTimeStamp + " " + message;
    lineAppender.append(line);
}
```

fokussierter

```
@Test
public void testTimeStampFormat() throws Exception {
    assertEquals("01:00:00", formatter.format(TIME_01_00_00));
}
```

einfacher testbar

```
@Test
public void testTimeStampHourLimit() throws Exception {
    assertEquals("01:59:59", formatter.format(TIME_01_59_59));
}

...
```

DfT Strategie

Quellen

- http://www.testbarkeit.de

- http://en.wikipedia.org/wiki/Design_for_testing

- Peter Zimmer, 2012: Testability – A Lever to Build Sustaining Systems, OOP Conference 2012

- http://secs.ceas.uc.edu/~cpurdy/sefall11/testing_payneetal_1997.pdf

- Steve Freeman, Nat Pryce, 2009: „Growing Object-Oriented Software guided by Tests“

- Micheal Feathers, 2004: „Working effectively with legacy systems“

- Robert C. Martin, 2009: „Clean Code“

- Gerard Meszaros, 2007: „xUnit Test Patterns“ bzw. http://xunitpatterns.com/

- Christian Johansen, 2010: „Test-Driven JavaScript Development “

- http://de.wikipedia.org/wiki/Gesetz_von_Demeter

- Eric Evans, 2003: „Domain Driven Design“

Fragen und Diskussion

4A Solutions

Any Application
Anytime
Anywhere

http://commons.wikimedia.org/wiki/File:CarService.JPG