Aspektorientierung

Gibt es ein Leben nach Java und OO?

oliver.boehm@agentes.de

© agentes AG

Inhalt

agentes

- von Assember bis AspectJ
- Das Architektur-Dilemma
- Freie Sicht auf die Businesslogik
- Do you speak Aschbegt-Dschei?
- AOP by Examples
- AOP in der Praxis
- QS mit AOP
- Der Blick nach vorn
- Zusammenfassung

© agentes AG

Wir über uns

agentes

Gründung
- 01.04.2004

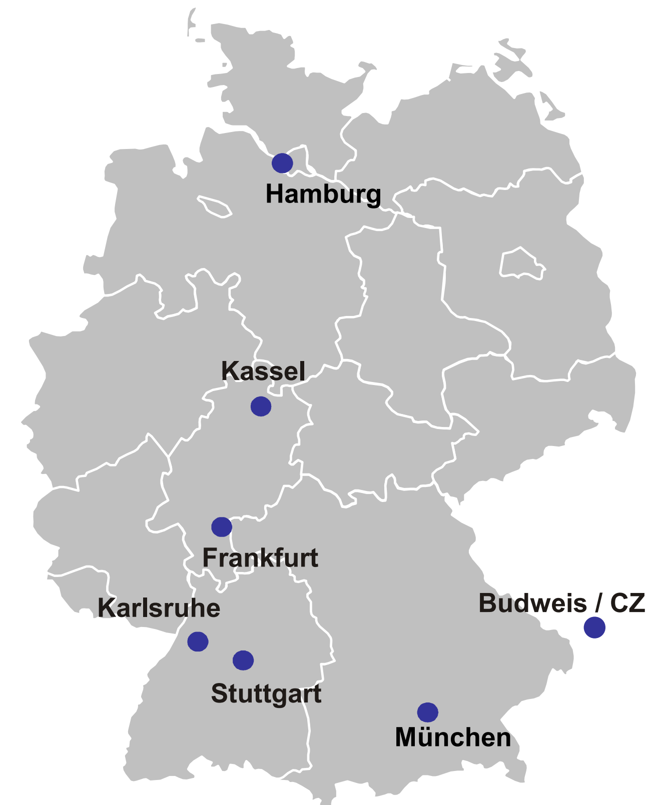
Vorstand
- Olaf Ahl, Wolfgang Clauss, Dr. Raimund Wiedemann

Beteiligungen
- agentes industries GmbH, 100 %-Tochterbeteiligung
- zelect GmbH, Karlsruhe, Mehrheitsbeteiligung
- agentes IT s.r.o., Budweis (CZ), Mehrheitsbeteiligung

Aktionäre
- PIRONET NDH AG, Köln (börsennotiert)
- agentes - Management

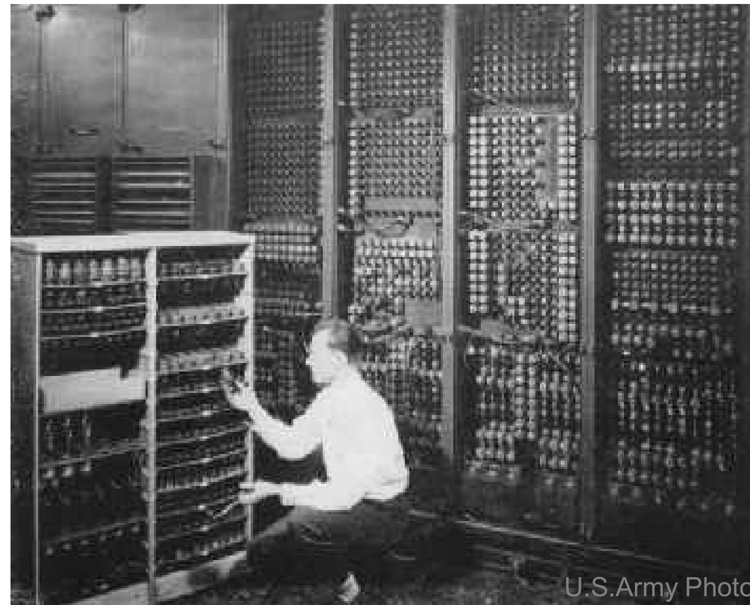
Mitarbeiter
- 120

Ergebnis
- 2004: Umsatz 8,6 Mio. EUR
- 2005: Umsatz 8,7 Mio. EUR
- 2006: Umsatz 10,0 Mio. EUR
- 2007: Umsatz 12,2 Mio. EUR

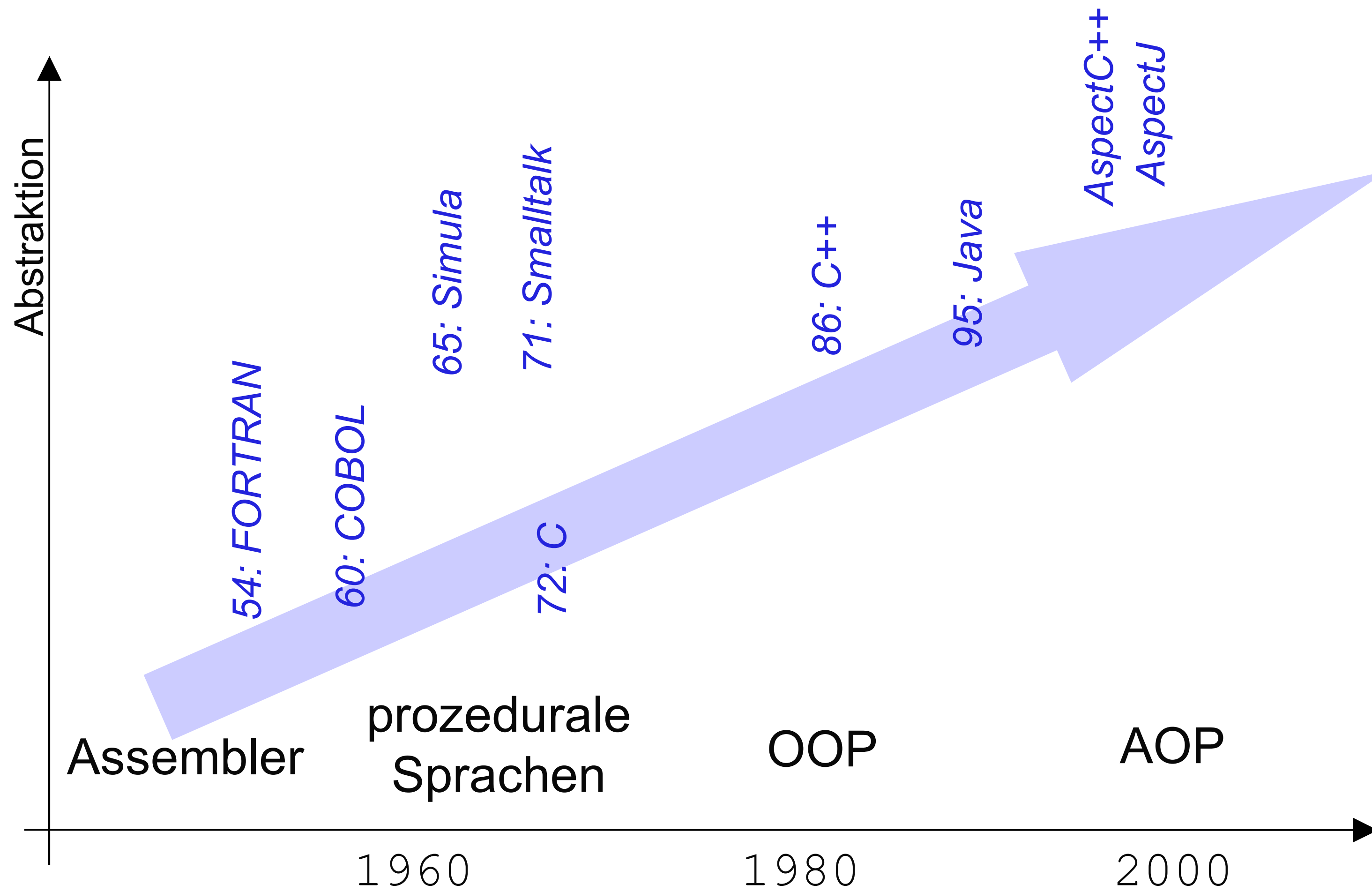


© agentes AG

von Assembler bis AspectJ



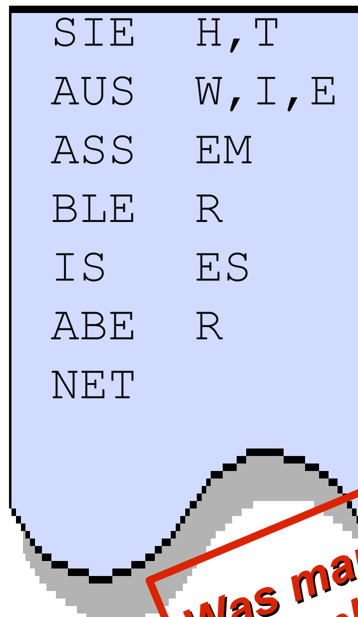
Die Geschichte der Programmiersprachen



© agentes AG

Assembler / Maschinensprache

agentes



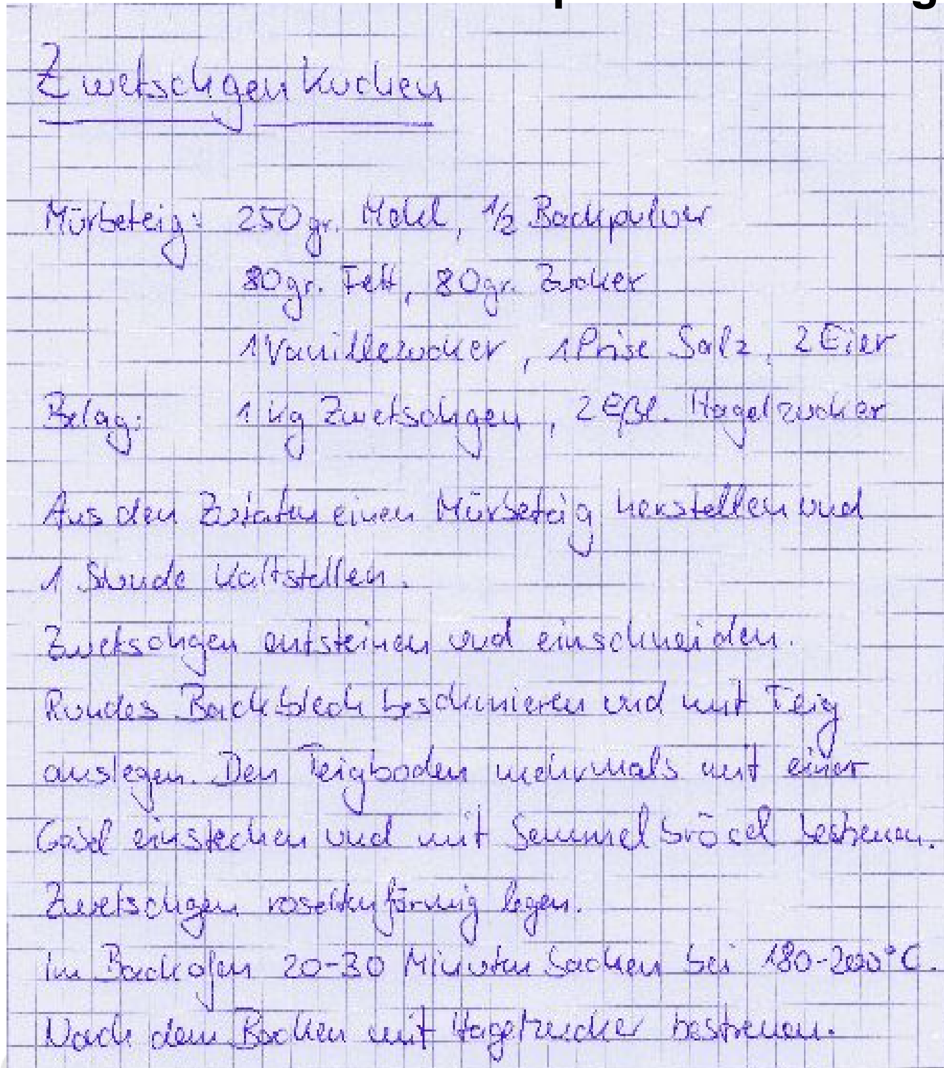
- **viele Dialekte**
  - Prozessor-abhängig
  - HW-abhängig
- **schnell**
- **nur von Spezialisten zu verstehen**

Was man nicht in Assembler machen kann, muss man löten!

© agentes AG

prozedurale Sprachen

- **Bsp: Backanleitung**



Zwetschgenkuchen

Mürbeteig: 250 gr. Mehl, 1/2 Backpulver
80 gr. Fett, 80 gr. Zucker
1 Vanillezucker, 1 Prise Salz, 2 Eier

Belag: 1 kg Zwetschgen, 2 EßL. Hagelzucker

Aus den Zutaten einen Mürbeteig herstellen und 1 Stunde kaltstellen.
Zwetschgen entsteinen und einschneiden.
Rundes Backblech beschmieren und mit Teig auslegen. Den Teigboden mehrmals mit einer Gabel einstechen und mit Semmelbrösel bestreuen.
Zwetschgen rosettenförmig legen.
Im Backofen 20-30 Minuten backen bei 180-200°C.
Nach dem Backen mit Hagelzucker bestreuen.

PASCAL

COBOL

C

FORTRAN

```
kuchen = backen(butter, mehl,
   zucker, eier, ...)
```


© agentes AG

OO-Sprachen

agentes

Ruby

SIMULA

Smalltalk

C++

Python

Eiffel

Java

```
kuchen = new Kuche(butter, mehl,
 eier);
kuchen.backe();
```

Kuchen
Zutaten Größe
add backe

© agentes AG

Aspektorientierte Sprachen

agentes

= OOPS, um Aspekte angereichert, z.B.

- Logging-Aspekt
- Security-Aspekt
- Transaktion-Aspekt
- ...

AspectS

AspectJ

AspectC++

© agentes AG

Das Architektur-Dilemma

© agentes AG

Die Quadratur der Architektur

- **gute Architekturen =**
  - einfach
  - erweiterbar
  - offen für künftige Anforderungen
  - ballastfrei
- **Dilemma**
  - wie sehen künftige Anforderungen aus?
  - welche muss ich berücksichtigen?
  - wieviel Design muss sein? was ist zuviel Design?

Design auf Vorrat

oder

das “San Francisco”-Syndrom

© agentes AG

Ziel der SW-Entwicklung seit > 30 Jahren

agentes

- "Divide and Conquer"
- Separations of Concern
  - Concern = Ding, Anforderung, ...
  - es gibt fachliche Concerns (Geschäftslogik)
  - und nichtfachliche Concerns (Logging, Sicherheit, ...)
- Jeder Concern soll in einem eigenen Modul, Klasse, ... gekapselt werden

© agentes AG

fachliche Concerns

nichtfachliche Concerns

Logging

Authentifizierung

Performance

Autorisierung

Persistenz

Fehlerbehandlung

Datenhaltung

Transaktionen

Lastverteilung

Client-Server

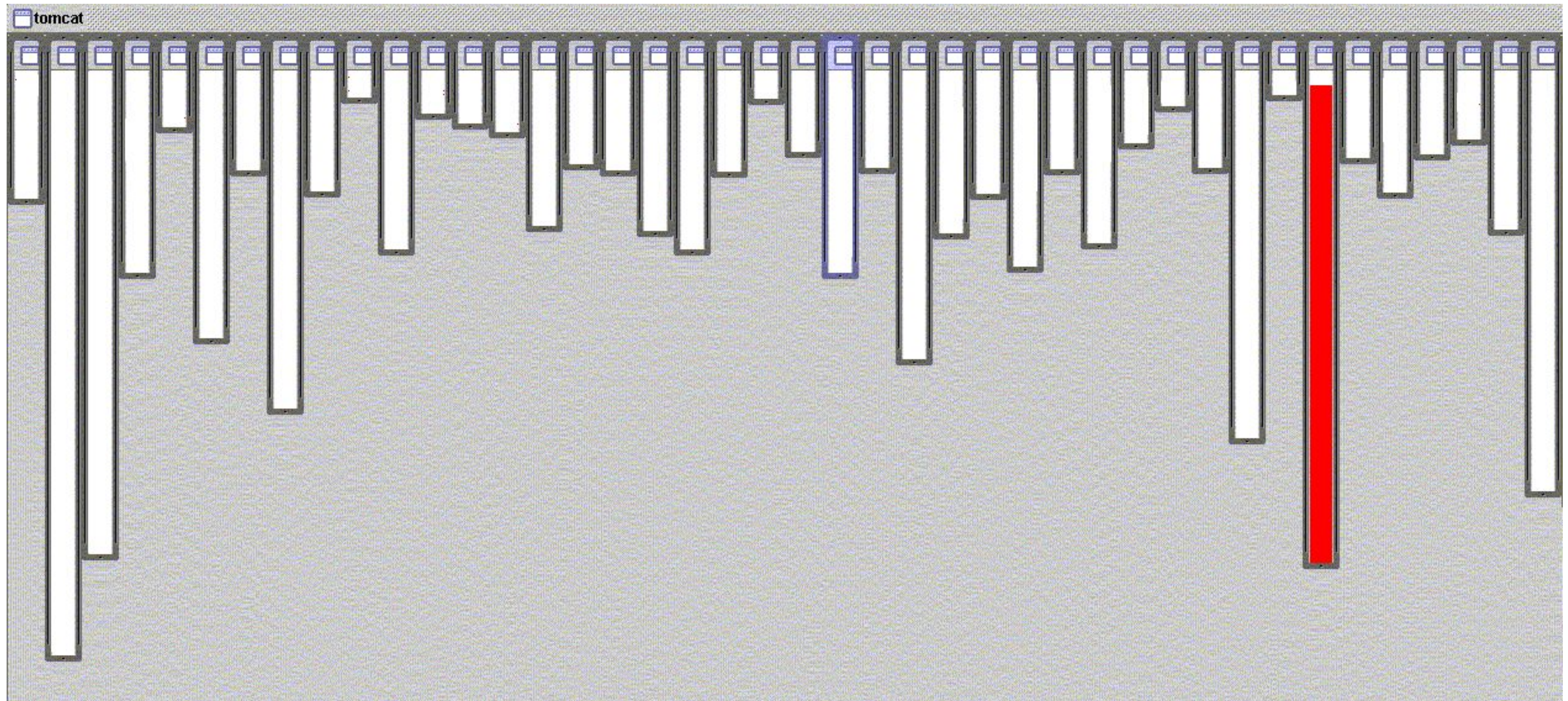
GUI

Sicherheit

Exception-Handling

agentes

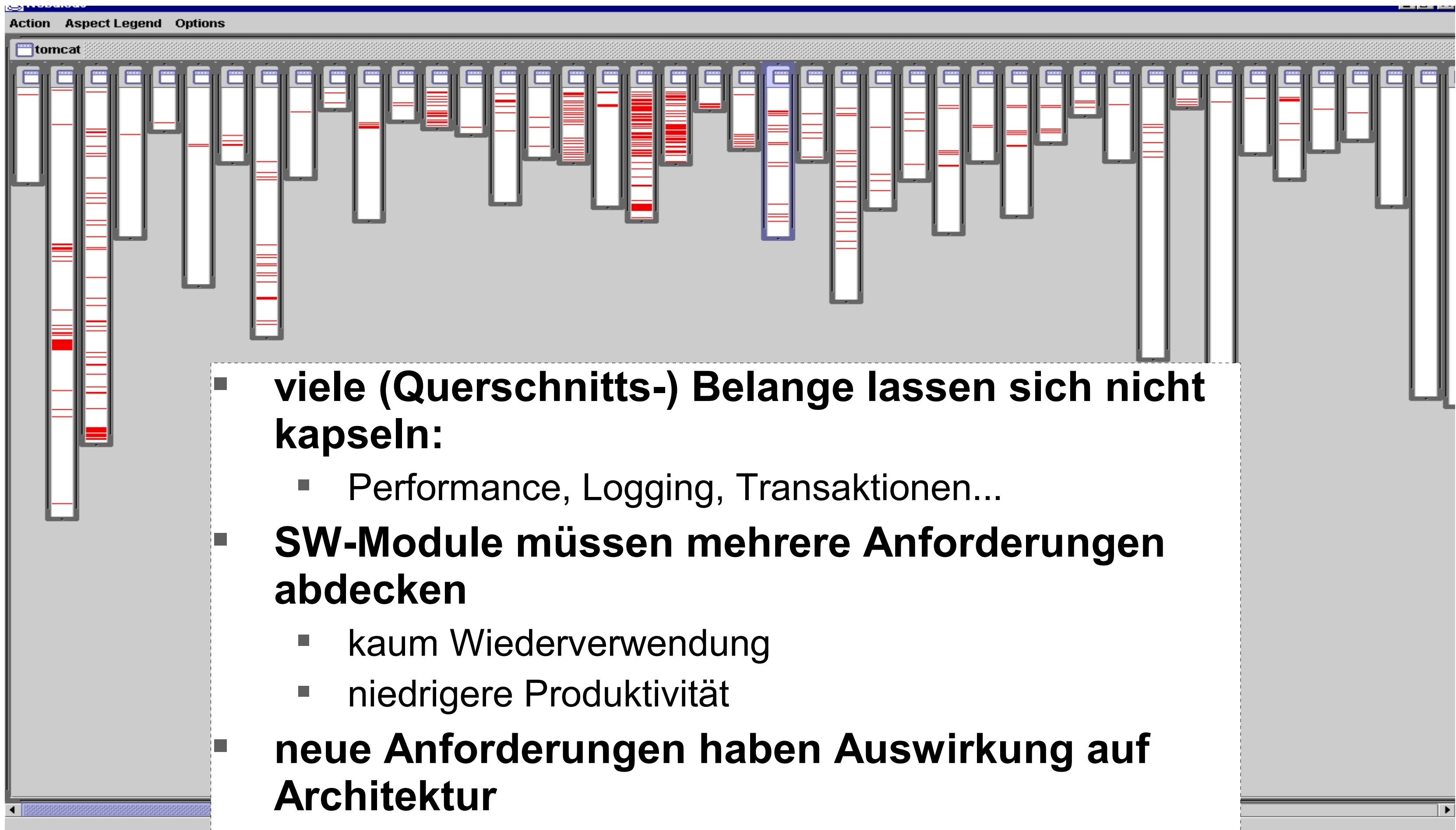
XML parsing



- XML-Parsing in org.apache.tomcat
  - rot = relevanter Code-Bereich
  - begrenzt auf 1 Modul

© agentes AG

aber...



Logging in org.apache.tomcat

- **viele (Querschnitts-) Belange lassen sich nicht kapseln:**
  - Performance, Logging, Transaktionen...
- **SW-Module müssen mehrere Anforderungen abdecken**
  - kaum Wiederverwendung
  - niedrigere Produktivität
- **neue Anforderungen haben Auswirkung auf Architektur**
  - Bsp.: Client-Server

© agentes AG

Beispiel Logging: Was tun bei Log-Änderungen / -Ergänzungen?

- **manuell**
  - aufwändig trotz IDE-Unterstützung (Refactoring)
  - fehleranfällig
- **Framework?**
  - Framework kann nicht entscheiden, wann geloggt werden soll
- **autom. Einfügen/Anpassung von Log-Anweisungen (Log-Generator)**
  - bei Änderungen muss Generator angepasst werden
  - wie konfigurieren? wie versionieren?
- **Log-Container + (XML-)Deskriptoren**
  - EJB-Ansatz
  - aufwendig
- **vielleicht AOP?**
  - mal sehn...

© agentes AG

Ein kleiner Ausflug mit AOP

© agentes AG

Beispiel: Bankkonto

agentes

fachliche Concerns

- Einzahlung
- Auszahlung
- Überweisung
- Bonitätsprüfung
- ...

nichtfachliche Concerns

- Logging
- Performance
- Authorisierung
- Sicherheit
- ...

© agentes AG

Ein Fallbeispiel...

agentes

```
public class Konto {

    private double kontostand = 0.0;

    public double abfragen() {
        return kontostand;
    }

    public void einzahlen(double betrag) {
        kontostand = kontostand + betrag;
    }

    public void abheben(double betrag) {
        kontostand = kontostand - betrag;
    }

    public void ueberweisen(double betrag, Konto anderesKonto) {
        this.abheben(betrag);
        anderesKonto.einzahlen(betrag);
    }

}
```

Konto
Kontostand
abfragen einzahlen abheben ueberweisen

© agentes AG

...mit Logging

agentes

```
public class Konto {

    private static Logger log = Logger.getLogger(Konto.class)
    private double kontostand = 0.0;

    ...

    public void einzahlen(double betrag) {
        kontostand = kontostand + betrag;
        log.info("neuer Kontostand: " + kontostand);
    }

    public void abheben(double betrag) {
        kontostand = kontostand - betrag;
        log.info("neuer Kontostand: " + kontostand);
    }

    ...

}
```

Achtung! Codeverschmutzung!

neue Anforderung:

alle Konto-Bewegungen müssen protokolliert werden!

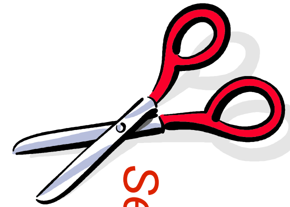
© agentes AG

...mit AspectJ

agentes

```
public aspect LogAspect {

    private static Logger log = Logger.getLogger(LogAspect.class);

    after(double neu) : set(double Konto.kontostand) && args(neu) {
        log.info("neuer Kontostand: " + neu);
    }

}
```

Konto
Kontostand
abfragen einzahlen abheben ueberweisen



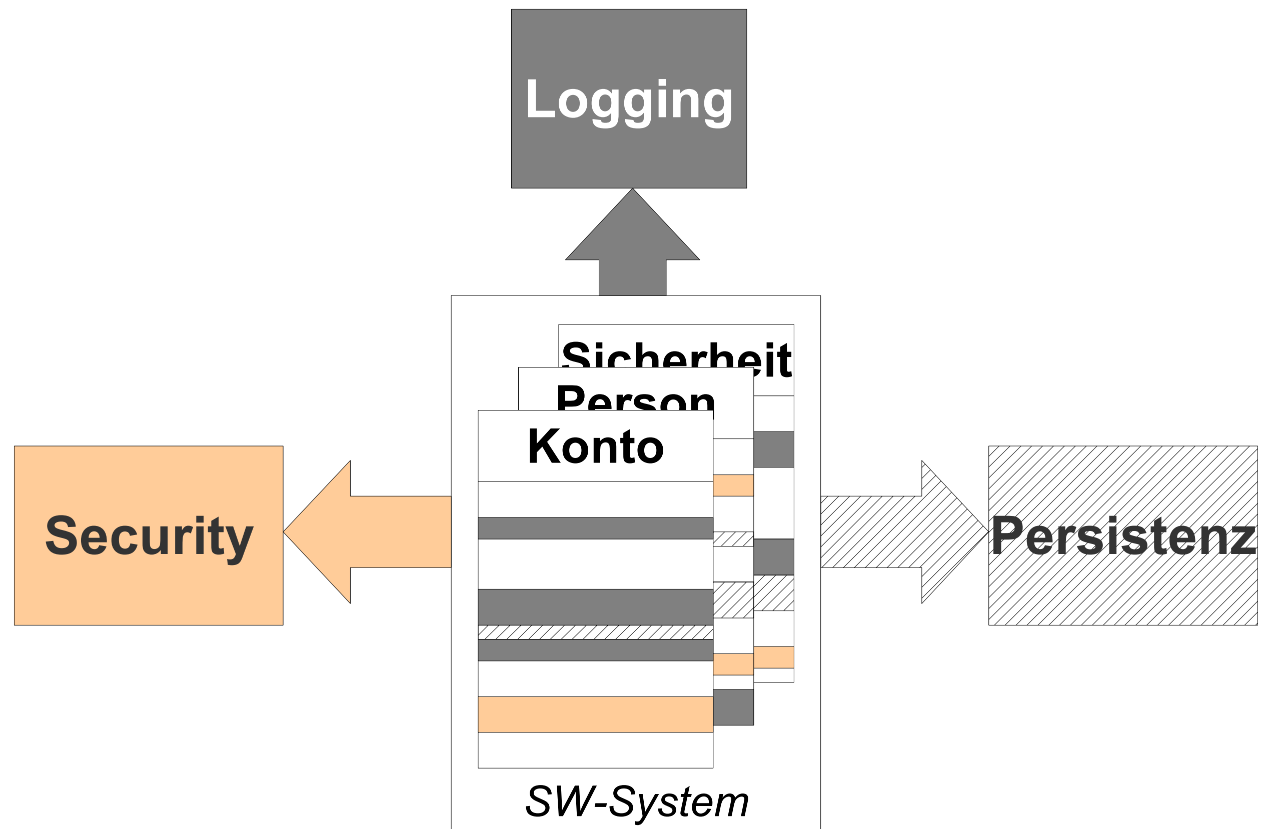
Separation of Concerns

LogAspect

© agentes AG

Do you think in AOP?

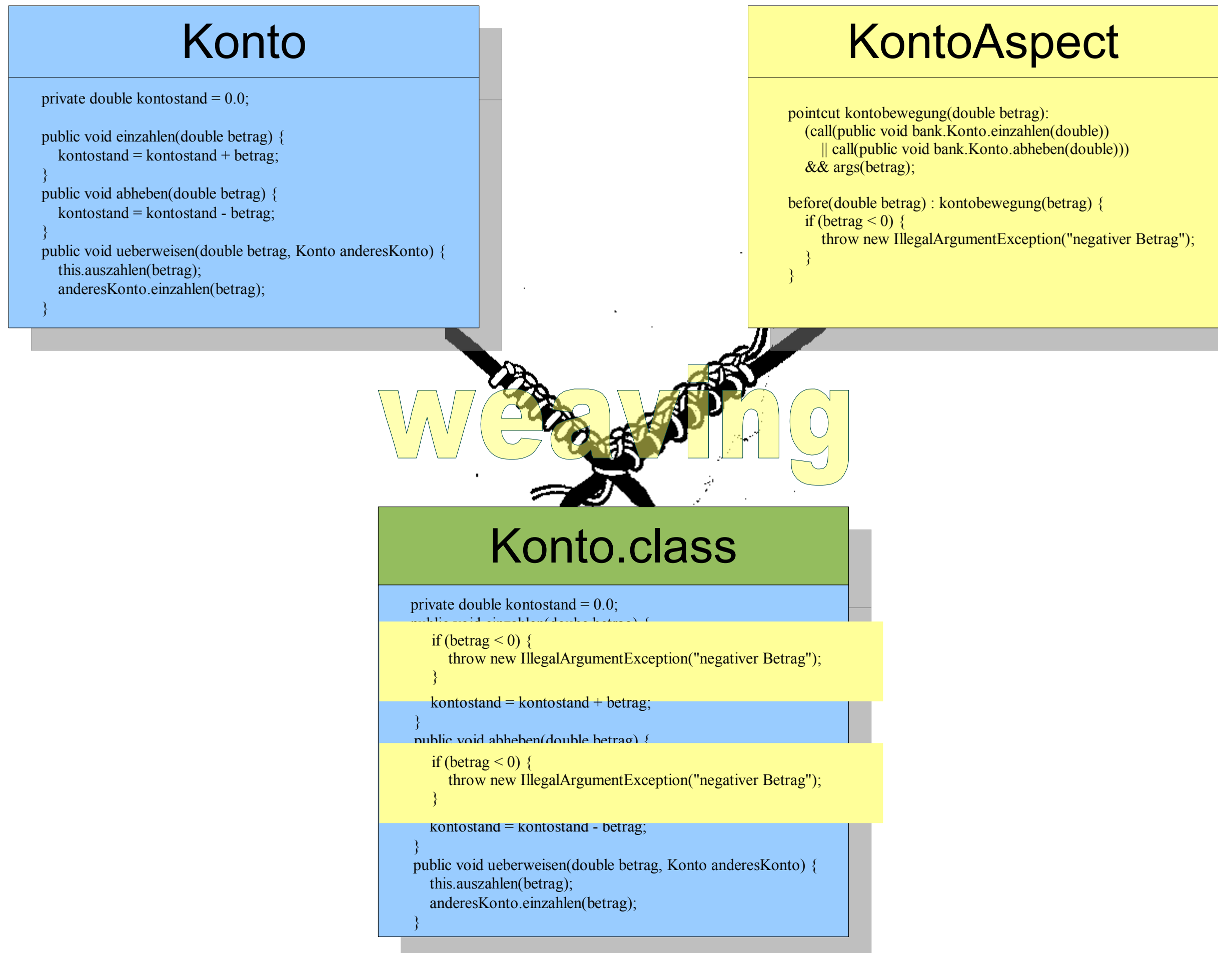
agentes



Das System als Menge von "Concerns"

© agentes AG

Der Webe-Vorgang (Weaving)



© agentes AG

Do you speak Aschbegt-Dschei?

© agentes AG

Begrifflichkeiten

agentes

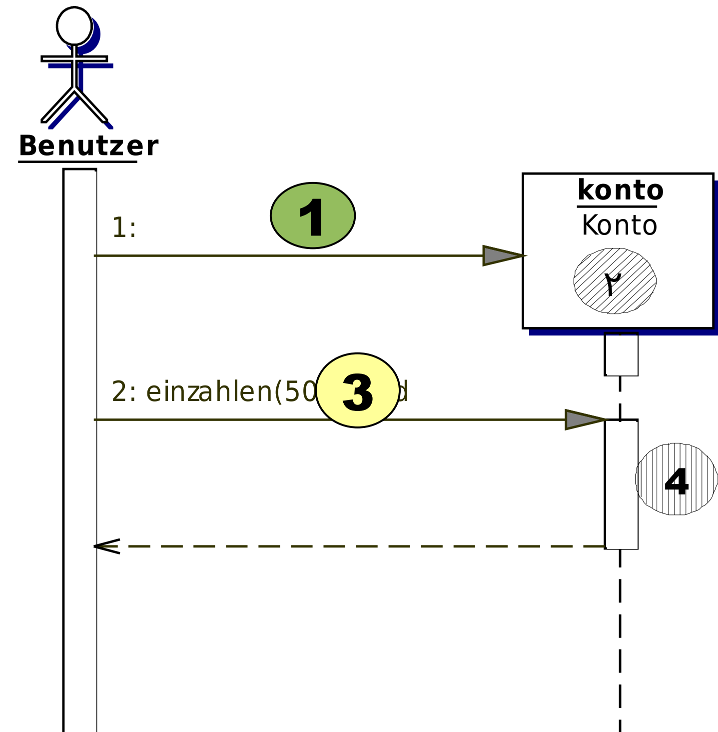
- **Joinpoint**
  - Punkte im Programm, an denen Code erweitert oder modifiziert werden soll
- **Pointcut**
  - eine Auswahl von Joinpoints
- **Advice**
  - der Code für den Pointcut
- **Introduction**
  - Erweiterung anderer Klassen, Interfaces, Aspekte um zusätzliche Funktionalität
- **Aspect**
  - Konstrukt, in dem das obige abgelegt wird

© agentes AG

Joinpoints können sein

- Aufrufen einer Methode
- Ausführen einer Methode
- Zugriff auf eine Variable
- Behandeln einer Exception
- Initialisierung einer Klasse
- Initialisierung eines Objekts

```
Konto konto = new Konto();
konto.einzahlen(500.0);
```



(1) Konstruktor aufrufen

(2) Objekt initialisieren

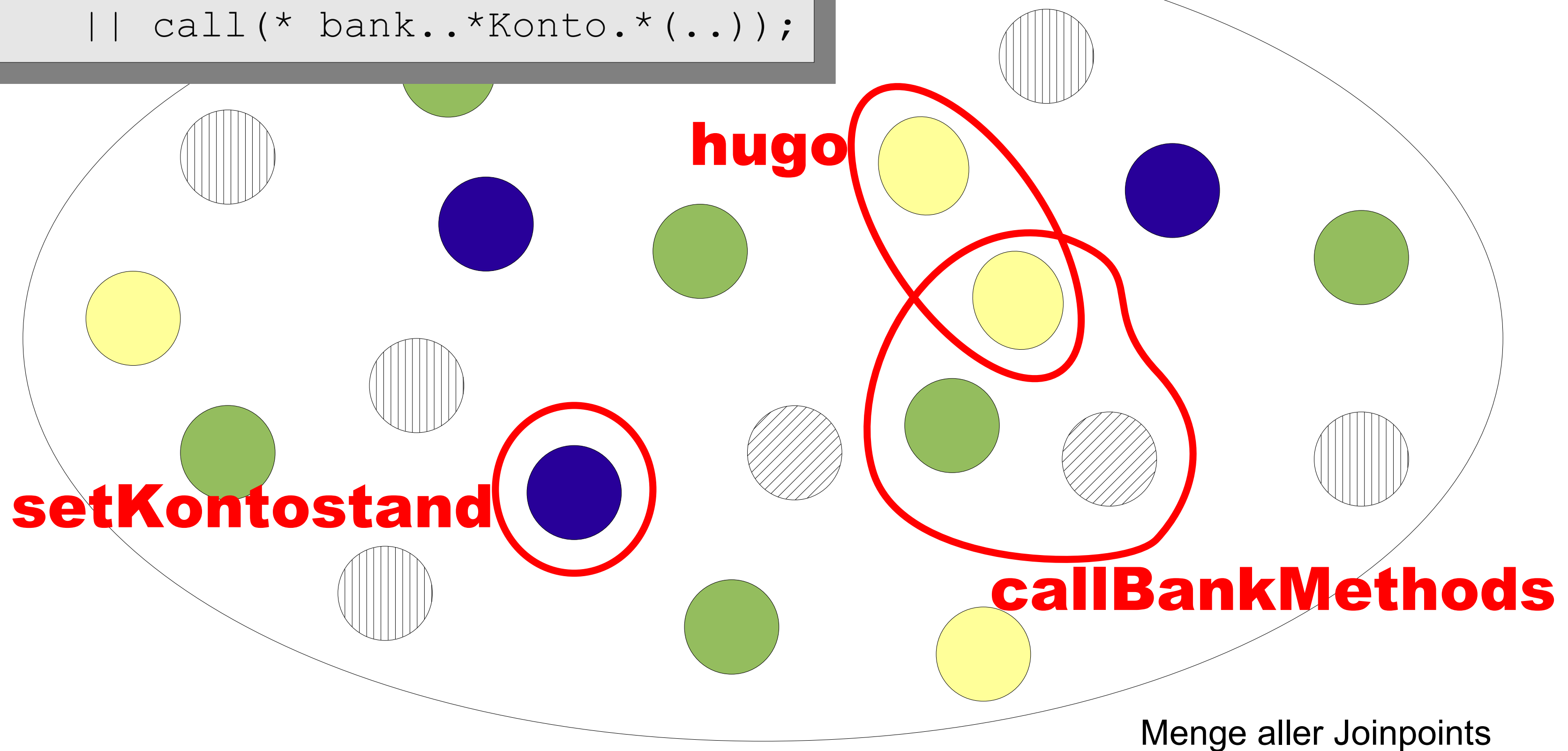
(3) Methode aufrufen

(4) Methode ausführen

© agentes AG

Pointcuts

```
pointcut setKontostand() :
    set(double bank.Konto.kontostand);

pointcut callBankMethods() :
    call(* bank.*Konto.*(*))
        || call(* bank..*Konto.*(..));
```



© agentes AG

Pointcut

agentes

- **Pointcut = Pattern oder Abfrageprädikat über Joinpoints**
  - Wildcards möglich
  - Logische Operatoren
- **Primitive Pointcut = Pointcut ohne Namen**
- **Named Pointcut**

Name

```
pointcut accessKontostand() :

    get(private double bank.Konto.kontostand)
    || set(private double bank.Konto.kontostand);
```


© agentes AG

Advice

agentes

- **Code, der eingewebt wird**
  - before()
  - after()
  - around()
- **Ähnlichkeit mit Methoden**
- **Zugriff auf Kontext:** `thisJoinPoint`

```
after() : callBankMethods() {
    log.debug("TEST: " + thisJoinPoint);
}
```

```
TEST: call(void bank.Konto.einzahlen(double))
TEST: call(double bank.Konto.abfragen())
...
```


© agentes AG

before-Advice

agentes

- **Aktion wird *vor* dem ursprünglichen Joinpoint ausgeführt**

```
before(): allPublic() {
        System.out.println(thisJoinPoint);
}
```

für alle public-Methode gib den Joinpoint aus

© agentes AG

after-Advice

agentes

- **Aktion wird *nach* dem eigentlichen Joinpoint ausgeführt**

```
after(): kontoMethods() {
    if (((Konto)thisJoinPoint.getTarget()).kontostand < 0) {
        throw new RuntimeException("Konto ueberzogen");
    }
}
```

überprüfe nach jeder Konto-Methode den Kontostand

© agentes AG

around-Advice

agentes

- **Aktion wird *anstelle* des Codes ausgeführt**

```
void around(double betrag): zahlen(betrag) {
        if (betrag < 0) {
                throw new IllegalArgumentException("negativer Betrag");
        }
        proceed(betrag);
}
```

überprüfe für die zahlen-Methoden den Parameter und rufe danach die eigentliche Methode auf

ruft den ursprünglichen Code auf

© agentes AG

Aspekte

agentes

- **Aspekt = Container für die neuen Konzepte**
  - können Advices und Pointcuts enthalten
  - Methoden, Variablen und Inner Classes
- **Aspekte sind Klassen sehr ähnlich**
  - verhalten sich zu Klassen wie Klassen zu structs in C++
- **können Interfaces implementieren**
  - z.B. um Default-Implementierung bereitzustellen
- **können von Klassen abgeleitet werden**

© agentes AG

Beispiel

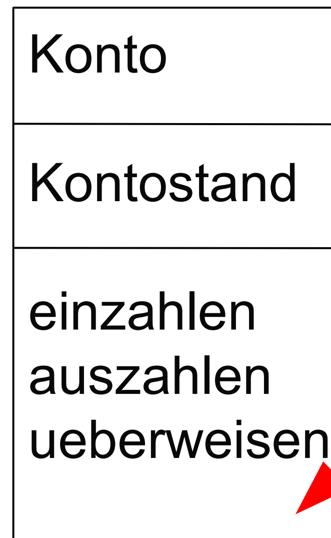
agentes

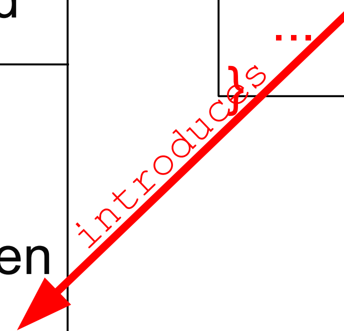
```
aspect WorldAspect {

    /**
     * Beispiel fuer einen benannten Pointcut ("mainOperation").
     */
    public pointcut mainOperation():
        execution(public static void hello.World.main(String[]));

    /**
     * say good bye
     */
    after(): mainOperation() {
        System.out.println("Good bye.");
    }

    /**
     * Beispiel fuer einen anonymen Pointcut ("execution(..)")
     */
    after() : execution(public static void hello.World.main(String[])) {
        System.out.println("Good bye.");
    }

}
```


© agentes AG

Introduction

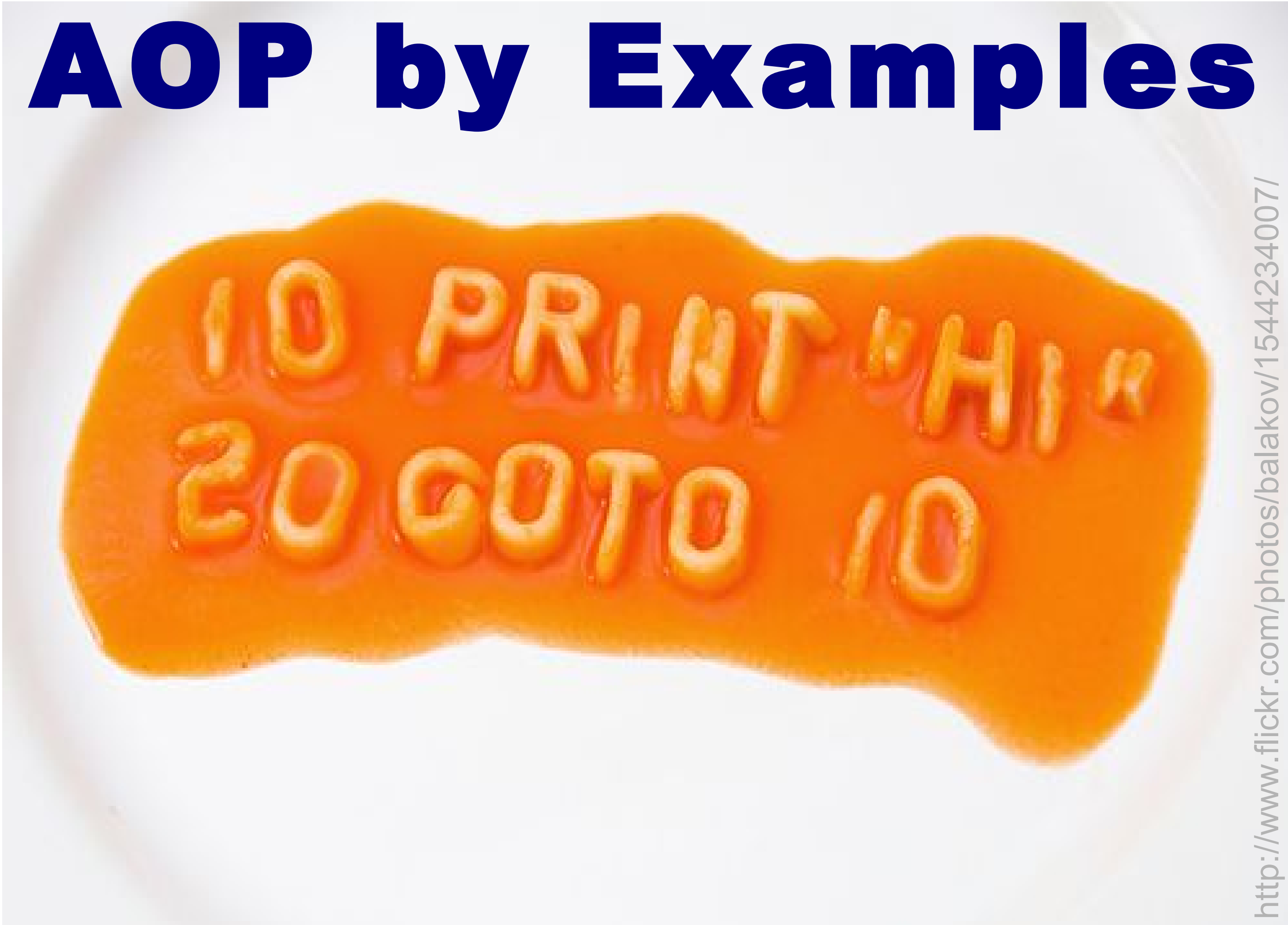
- **auch als “intra-class declaration” bekannt**
- **mit Introduction können andere Aspekte, Interfaces oder Klassen erweitert werden**
- **hinzugefügt werden können:**
  - Variablen
  - Methoden
  - Interfaces
  - Vaterklassen
- **bewährtes Konzept (“open classes”), z.B. in Python**





© agentes AG

AOP by Examples



© agentes AG

Kontosperre

agentes

- **kein Zugriff auf Kontostand**

```
pointcut accessKontostand() :
    get(private double bank.Konto.kontostand)
    || set(private double bank.Konto.kontostand);

before(Konto konto) : accessKontostand() && this(konto) {
    if (konto.isGesperrt()) {
        log.warn("Zugriff verweigert (Konto gesperrt)");
        throw new IllegalStateException("Konto ist gesperrt");
    }
}
```


© agentes AG

Autorisierung

agentes

- **für gekennzeichnete Methoden ist ein Login erforderlich**

```
public class Konto {
    ...
    @LoginRequired
    public void abheben(double betrag) {
        kontostand = kontostand - betrag;
    }
}
```

```
pointcut loginRequiredExecutions() :
    execution(@LoginRequired * bank.Konto.*(..));

Object around() : loginRequiredExecutions() {
    if (!AccessControl.loggedIn()) {
        try {
            AccessControl.login();
        } catch (LoginException e) {
            throw new AccessControlException("login failed");
        }
    }
    return proceed();
}
```


© agentes AG

Singleton mit AOP

agentes

```
/** Marker Interface */
public interface Singleton {}

/** Es kann nur einen geben... */
public class Highlander implements Singleton {
    ...
}
```

Idee:
es wird immer das gleiche Objekt bei new(..) zurückgegeben

```
public aspect SingletonAspect {

    private Hashtable singletons = new Hashtable();

    pointcut selectSingletons() : call((Singleton +).new(..));

    Object around() : selectSingletons() {
        Class singleton =
            thisJoinPoint.getSignature().getDeclaringType();
        synchronized(singletons) {
            if (singletons.get(singleton) == null) {
                singletons.put(singleton, proceed( ));
            }
        }
        return singletons.get(singleton);
    }

}
```


© agentes AG

Beispiel: Verbindungsaufbau

agentes

- **Socket aufbauen (teuer):**
  - `Socket socket = new Socket(hostname, 80);`
- **Socket lesen / schreiben**
- **Socket freigeben**
  - `socket.close();`
- **Problem Freigabe:**
  - Hmm, vielleicht brauche ich den Socket noch
  - andererseits: nicht benötigte Resourcen sollten freigegeben werden
- **Abhilfe**
  - Socket-Pooling
  - lohnt sich aber nur, wenn ich viele Verbindungen habe!!!
- **nächstes Problem:**
  - aber da muss ich ja alle Socket-Kreierungen abändern :-(

© agentes AG

Lösung: SocketPoolingAspect

agentes

```
/**
 * Pointcut fuer das Oeffnen bzw. Anlegen eines Sockets
 */
pointcut socketCreation(String host, int port) :
    call(public Socket.new(String, int))
    && args(host, port)
    && !within(SocketPool);

/**
 * Pointcut fuer das Schliessen eines Sockets
 */
pointcut socketClose(Socket socket) :
    call(public void Socket.close())
    && target(socket)
    && !within(SocketPool);
```

s. WebRobot4/src/web/SocketPoolingAspect.java

Lösung: SocketPoolingAspect (2)

agentes

```
/**
 * Das Oeffnen eines Sockets erfolgt nun ueber den socketPool.
 * Falls dies fehlschlaegt, wird der Original-Pointcut aufgerufen.
 */
Socket around(String host, int port)
        throws UnknownHostException, IOException
        : socketCreation(host, port) {
    Socket socket = socketPool.getSocket(host, port);
    if (socket == null) {
        return proceed(host, port);
    } else {
        return socket;
    }
}
```

s. WebRobot4/src/web/SocketPoolingAspect.java

© agentes AG

weitere Einsatzgebiete

agentes

- **Design-Patterns**
- **Fortschrittsbalken**
- **Performance-Messungen / Profiling**
- **Testen**
  - Simulationspunkte über AOP
  - Mocken mit AspectJ
- **Object Recording der Schnittstellen nach außen**
- **Caching / Pooling**
  - Paradebeispiel JBoss Cache
- **Spring**
  - setzt intern AOP / AspectJ ein
- **Patch**
- **u.v.m.**

© agentes AG

weitere Beispiele

agentes

- **Oliver Böhm**
  Aspektorientierte Programmierung mit AspectJ 5
  http://www.dpunkt.de/buch/3-89864-330-1.html
- **die Seite zum Buch**
  http://www.aosd.de
- **AOP-Ecke**
  http://www.agentes.de/aop/
- **Ramnivas Laddad**
  AspectJ in Action
  Manning Publications Co., 2003, ISBN 1-930110-93-6
- **PatternTesting**
  http://patterntesting.sf.net

© agentes AG

AOP in der Praxis

© agentes AG

AspectJ-Projekte: jGEBOS

agentes

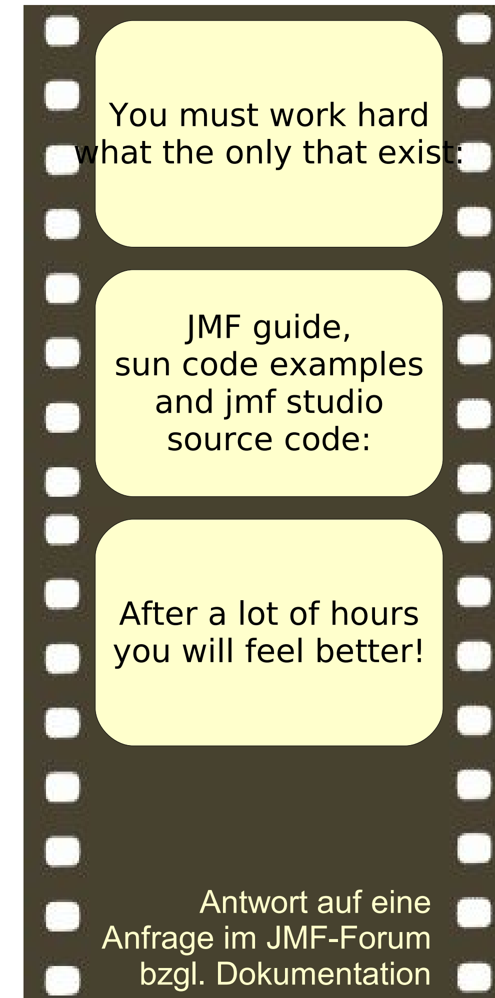
- **2000: Ablösung OS/2-Bankarbeitsplatz (C++)**
- **Multi-Tier-Architektur: Swing-Client / Application-Server / Backend (Mainframe)**
- **ab 2003: Beginn mit AspectJ 1.1**
  - erweitertes Logging zur Fehlersuche
  - Profiling-Aspekt
  - zweigleisige Strategie:
    - offizielle Version: pure Java
    - Notfall-Version: Nachbrenner-Build der ausgelieferten Jar-Files
- **Erfahrungen:**
  - IDE mit Schwächen (keine inkrementelle Compilierung, Debugger)
  - mühsamer KnowHow-Aufbau:
    - StackOverflowErrors are your best friend!
  - Ant-Unterstützung hilfreich und ausreichend

© agentes AG

AspectJ-Projekte: Media-Streamer in Java

agentes

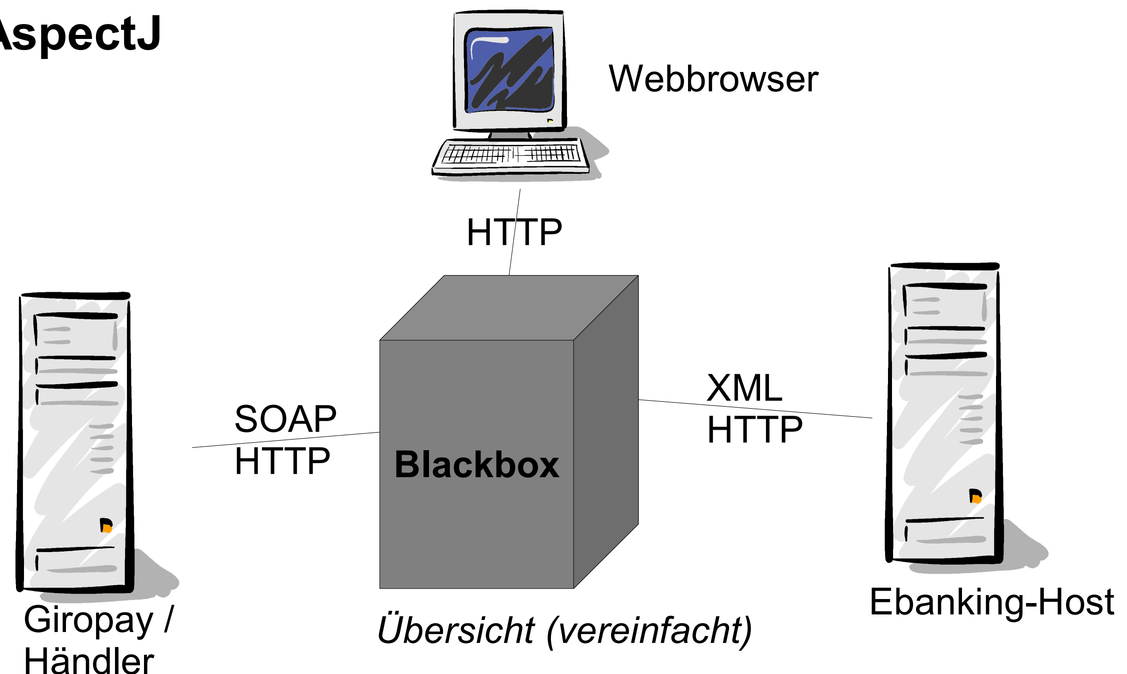
- **Auftrag 2004: Streamer für alle Arten von Daten**
- **Problemfall: JMF (Java Media Framework)**
  - Doku? veraltet, fehlerhaft
  - Source = Doku (JMStudio + viele Beispiele)
  - Zustands-/Event-getriebene Architekur
  - Anwendung? funktionierte selten wie erwartet
- **Vorgehen:**
  - Debugger? sinnlos, da eventbasierte Architetkur
  - Netzwerkmonitor (tcdump, ethereal, ...): mühsam
  - **Logging über Aspekte: äußerst hilfreich**
- **Erfahrungen:**
  - AspectJ ideal für Expeditionen in unbekanntes Gebiet
  - keine Änderungen im Code-Dschungel nötig
  - Änderungen im Gepäck (Aspect)
- **nähere Infos: http://www.agentes.de/artikel/jmf.html**



© agentes AG

AspectJ-Projekte: Blackbox

- **eBanking-Komponente**
  - J2EE Anwendung
  - Schnittstellen zu anderen Systemen
    - XML
    - SOAP
- **Basis-Funktionalität**
  - pure Java
- **Zusatz-Funktionalität mit AspectJ**
  - Logging, Tracing
  - Monitoring
  - Test-Unterstützung



Übersicht (vereinfacht)

© agentes AG

Beispiel: Tracing

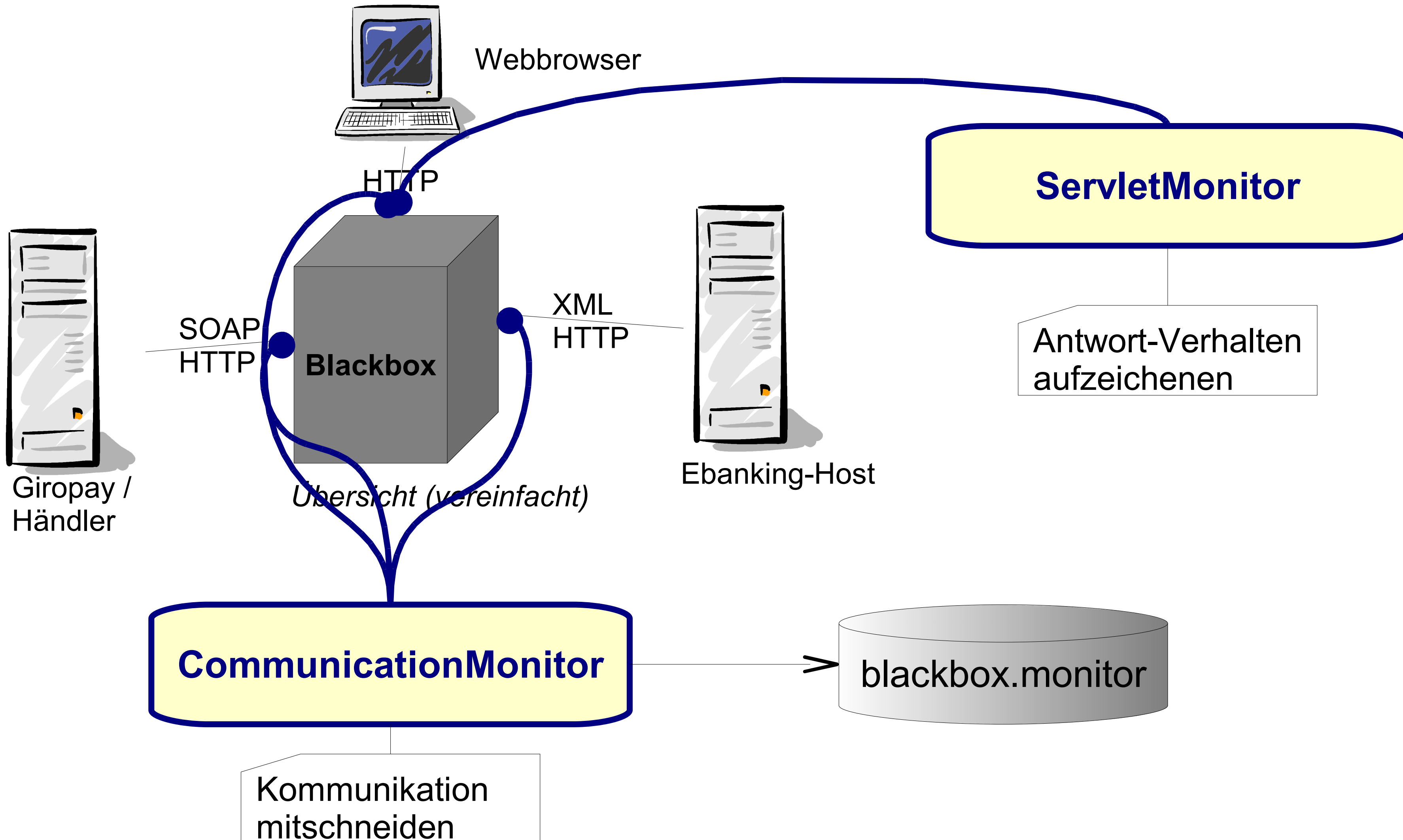
agentes

- **jede Methode wird protokolliert**
- **Aufwand: ca. 2 Stunden**
- **ROI:**
  - Endlosschleife während Testbetrieb gefunden
  - Kommunikationsprobleme konnten eingekreist werden

```
22:23:33,481 > blackbox.SmokeTest.testOnlineEbankingHost()
22:23:33,481 | > blackbox.admin.Configurator.getEbankingHost()
22:23:33,481 | < blackbox.admin.Configurator.getEbankingHost() = 10.11.12.13
22:23:33,484 | > blackbox.net.NetworkTester.isReachable(10.11.12.13)
22:23:33,484 | | > blackbox.net.NetworkTester.isReachable(10.11.12.13, 80)
22:23:33,486 | | | > blackbox.admin.Configurator.isDemoMode()
22:23:33,486 | | | | > blackbox.admin.Configurator.hasProperty(mode, demo)
22:23:33,487 | | | | < blackbox.admin.Configurator.hasProperty(mode, demo) = false
22:23:33,487 | | | < blackbox.admin.Configurator.isDemoMode() = false
22:23:36,499 | | < blackbox.net.NetworkTester.isReachable(10.11.12.13, 80) = false
22:23:36,500 | < blackbox.net.NetworkTester.isReachable(10.11.12.13) = false
22:23:36,500 <*blackbox.SmokeTest.testOnlineEbankingHost()
...
```


© agentes AG

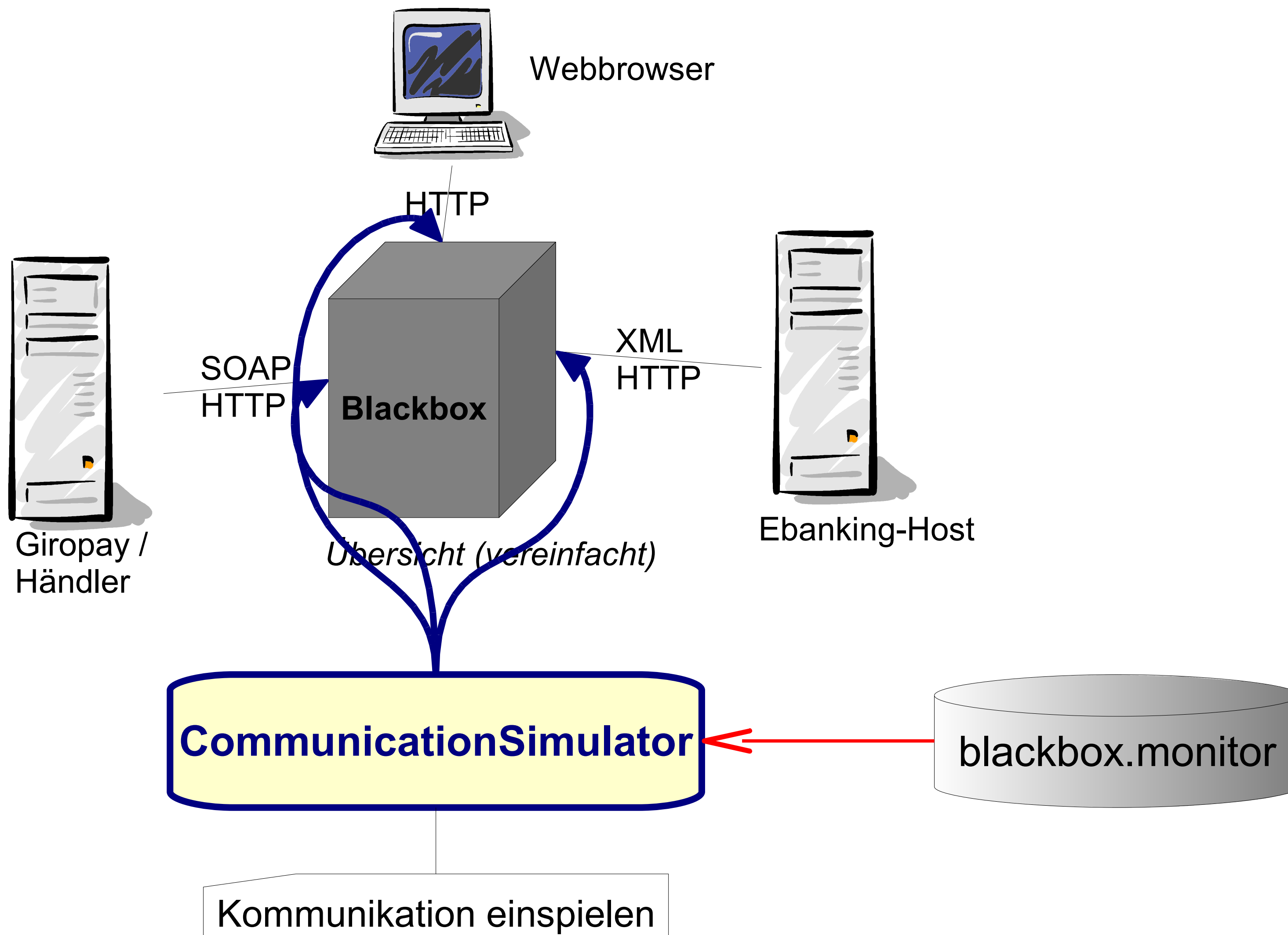
Beispiel: Monitoring



© agentes AG

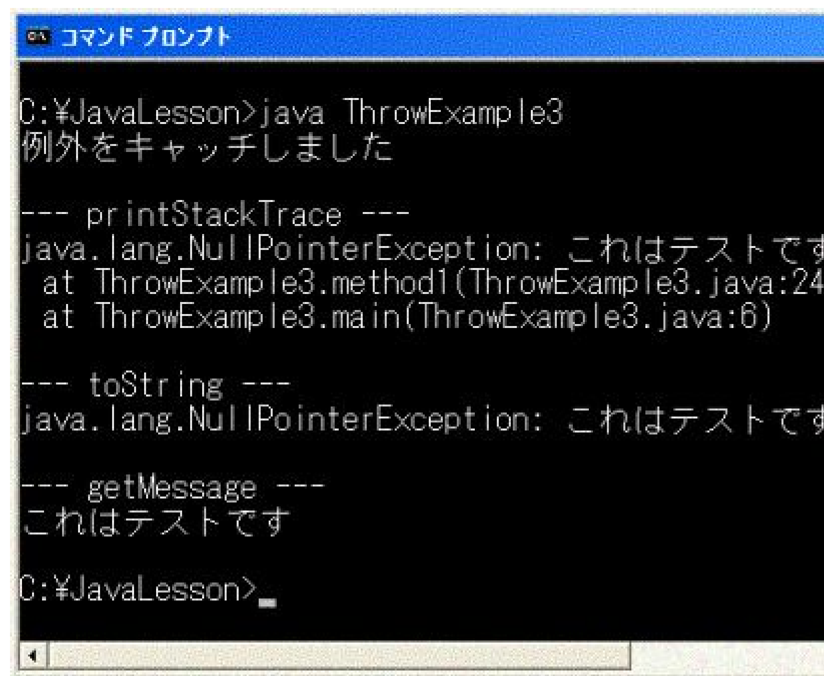
Beispiel: Testen

agentes



© agentes AG

QS mit AOP



© agentes AG

statische Prüfung

agentes

- **declare error / declare warning**

```
declare warning:
    call(public * java.sql.*(..))
        "please use persistence framework...";
```

- **Exception Handling**

```
pointcut handleException(Throwable ex):
    handler(java.lang.Throwable+) && args(ex);

/** protokolliere alle abgefangen Exceptions */
before(Throwable ex) : handleException(ex) {
    System.err.println("*** " + ex);
}
```


© agentes AG

Dynamische Prüfung / Demo

© agentes AG

Was sind NullPointerExceptions (NPE)?

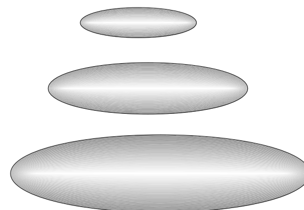
- **normal?**
  - http://www.gourmondo.de/hitlist.jsp (-> zeitweise nicht erreichbar)
  - http://www.sisby.de/sisby/base/de/Suche/GewImmoBoerse/ergSuche/index.jsp?focu
  - http://www.learn-line.nrw.de/wettbewerbe/details.jsp?id=109
  - http://www.casinoeuro.com/de/flash/fullflash.jsp?game=
  - s. Google: *"java.lang.NullPointerException" filetype:jsp*
- **ärgerlich!**
- **Zeichen unzureichender Tests**
- **gefährlich für die Anwendung**
- **schlecht für das Image**

© agentes AG

Wodurch entstehen NPEs?

agentes

- **durch Sorglosigkeit**
- **durch geänderte Rahmenbedingungen**
  - andere Umgebung (Produktion)
  - geänderte Anforderungen
- **durch unzureichende Tests**



Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.

(aus http://www.michael-puff.de/Ablage/Signaturen.php)

© agentes AG

Wie kann AOP helfen?

agentes

Wehret den Anfängen:

- (null) als Argument ist verboten
- (null) als Rückgabewert wird verboten
- Ausnahmen werden dokumentiert

Weitere Maßnahmen:

1) Tests
2) Tests
3) Tests

© agentes AG

Lösung mittels AOP

agentes

- **Joinpoints definieren:**
  - alle Methoden u. Konstruktoren mit mind. 1 Parameter

```
pointcut methodsWithNoNullArgs() :
    execution(* *..*(*, ..))
    || execution(*..new(*, ..));
```

- **Code einfügen (über Advice):**

```
before() : methodsWithNoNullArgs() {
    Object[] args = thisJoinPoint.getArgs();
    for(int i = 0; i < args.length; i++) {
        if (args[i] == null) {
            throw new AssertionError("arg" + i + " is null");
        }
    }
}
```


© agentes AG

erstes Ergebnis

- **Erschrecken**
  - verdammt viele unsaubere Code-Stellen
  - zu oft zu sorglos
- **Verzweiflung**
  - nicht alle gefundenen „Unsauberkeiten“ ließen sich einfach beheben
  - Ausnahmeregelung gefordert!

© agentes AG

Lösung II

- **Ausnahmen definieren**

```
pointcut nullArgsAllowed() :
    execution(public int
        ebanking.jaas.XmlLogin.loginWithVrNetKey(String,
        String, String))
    || execution(public void ebanking.design.Setting.set*(*))
    ;
```

- **Advice anpassen**

```
before() : methodsWithNoNullArgs()
        && !nullArgsAllowed() {
    ...
}
```

© agentes AG

Rückgabe überwachen

agentes

- **Joinpoint mit Ausnahme**

```
pointcut nonVoidMethods() :
    execution(Object+ *..*(..));
```

- **Ausnahmen**

```
pointcut mayReturnNull() :
    execution(* ebanking.PersonOverview.getPerson(String))
    || execution(* *..*.find*())
    ;
```

- **Advice**

```
after() returning(Object returned) :
        nonVoidMethods() && !mayReturnNull() {
    if (returned == null) {
        throw new AssertionError(thisJoinPoint.getSignature()
                + " returns (null)!");
    }
}
```


© agentes AG

Ergebnis

agentes

- **Aufdeckung vieler unsauberer Stellen**
  - setName(null) -> resetName()
- **Dokumentation der Ausnahmen**
  - Sorglosigkeit ersetzt durch Zwang zum Überlegen (dokumentiere ich oder mache ich es richtig?)
- **weniger NullPointerExceptions**
- **aber:**
  - *Tests nach wie vor notwendig!!!*

© agentes AG

PatternTesting

© agentes AG

Was ist PatternTesting?

agentes

PatternTesting is a testing framework that allows to automatically verify that Architecture/Design/Best practices recommendations are implemented correctly in the code. It uses AOP and AspectJ to perform this feat.

(aus: http://patterntesting.sf.net)

© agentes AG

die Idee dahinter

agentes

- **Aufspüren von „BugPattern“**
  - Übergabe von *null*-Argumenten
  - *null* als Rückgabewert
  - „bad smells“ (z.B. *e.printStackTrace()* statt Logging)
- **Einhalten von Programmierrichtlinien**
  - kein System.out.println()
  - Lasagne-Architektur:
    - obere Schicht darf nur darunterliegende Schicht aufrufen
    - nicht umgekehrt

© agentes AG

Status (2 years ago)

agentes

- **etwas verwahrlost**
- **seit 2 Jahren Stillstand**
  - letzte Version: PatternTesting-0.3 vom Sept. 2004
- **aber: gute Ideen**
  - z.B. Einbindung in Ant u. Maven

© agentes AG

PatternTesting SNAPSHOT

agentes

- **Anfrage / Kontaktaufnahme April 2007**
  - mit Vincent Massol u. Matt Smith
  - urspr. Grund: AOP-Day '07
  - zurzeit mit anderen Projekten beschäftigt (Ruleby, Xwiki, Cargo/Maven)
- **derzeitige Planung**
  - ✔ Umstellung Maven 1 -> Maven 2
  - ✔ Test-Fälle zum Laufen bringen
  - autom. Build (Continuum)
    - ✗ Build-Server gesucht
    - ✗ noch kein autom. Deployment
- **PatternTesting 0.5.1**
  - BugPattern + Anregungen aus JAX 2008
  - Einsatz von Annotations (-> Java5, AspectJ5)
- **künftige Planung (0.6)**
  - Java 6, AspectJ 6
  - aussagekräftigere (IO)Exceptions
  - weitere Aspekte aus bestehenden Projekten einpflegen

© agentes AG

Der Blick nach vorn

© agentes AG

Neuerungen in AspectJ 6

agentes

- **Unterstützung Java 6**
- verbesserte inkrementelle Kompilierung
- Bug-Fixing (Generics)
- Annotations jetzt auch für Parameter-Matching

© agentes AG

AOP meets MDA

agentes

- noch keine graphische Notation für Aspekte
- techn. Aspekte für techn. Randbedingungen
- flexibler
- agiler
- Gefahr: Aspekte ufern aus

- Idee: Modell enthält nur Business-Logik, Rest wird generiert
- restliche Logik im Generator manifestiert
- Gefahr: Generator-Entwicklung, Abhängigkeit vom Generator

© agentes AG

erwarteter Nutzen

- **Modularität auch für übergreifende Dinge (“Concerns”)**
  - Java-Code konzentriert sich auf Business-Logik
  - kürzerer Code
  - einfachere Wartung und Weiterentwicklung
- **erhöhte Wiederverwendbarkeit über**
  - Library-Aspekte
  - Plug n' Play-Aspekte (“Patterns”) für verschiedene Anforderungen

© agentes AG

Einführungs-Strategien

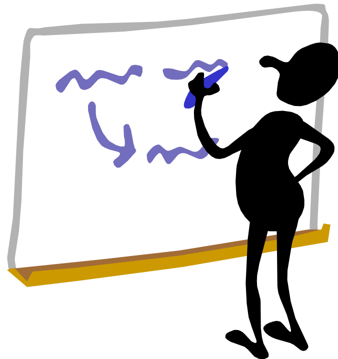
- **Top-Down (mit Unterstützung des Managements)**
- **Bottom up (don't tell)**
- **über die QS-Schiene**

© agentes AG

Adaptionsaufwand

agentes

- **Ungefähr so groß wie der Umstieg von Prozedural auf OOP**
- **Aber:**
  - er wird nicht kleiner
  - er wird kommen
  - warum nicht jetzt?



- **können wir aus der Vergangenheit lernen?**
- **erste OO-Projekte waren oft Fehlschläge**
  - Lernkurve unterschätzt
  - fehlende Begleitung von erfahrenen Experten
  - überzogene Erwartungen
- **erst folgende OO-Projekte waren erfolgreich**
  - klein mit unkritischem Projekt anfangen
  - Erfahrungen sammeln
  - Multiplikatoren ausbilden

© agentes AG

Erfahrungen

- ☺ **Logging ideal zum Einstieg**
  - bis zu 70% des Codes = Logging
  - Qualität ist entscheidend für Fehlersuche
- ☺ **höhere Qualtität durch**
  - eigene Fehler-/Warnungen
  - Architektur-Überprüfungen
- ☺ **Testen**
  - Mock-Ersatz bzw. Ergänzung
- ☺ **Architekturen werden einfacher**
  - weniger Abstractionslayer
  - mehr Business-Logik
- ☺ **passt gut zu XP / Agile Methoden**
  - Konzentration auf *eine* Sache
  - Randbedingungen können später noch berücksichtigt werden

- ☹ **Fallstricke**
  - übertriebener Einsatz ist kontra-produktiv
- ☹ **Refactoring**
  - wenig Unterstützung

© agentes AG

Zusammenfassung

agentes

- Am Anfang war der Assembler
- danach: prozedural, OO
- Problem: Querschnittsbelange (Crosscutting Concerns).
- Lösung: AOP
- AOP / AspectJ baut auf OOP / Java auf
- AOP / AspectJ gewinnt an Bedeutung
- Entwicklung wird effektiver
- kleinere Programme möglich

- maßloser Einsatz kann Gegenteil bedeuten
- Fallstricke lauern
- Selbstdisziplin nötig
- Lernkurve vergleichbar mit C -> C++

AspectJ = Java++

© agentes AG

Bedeutung von Aspekten in 2-5 Jahren

agentes

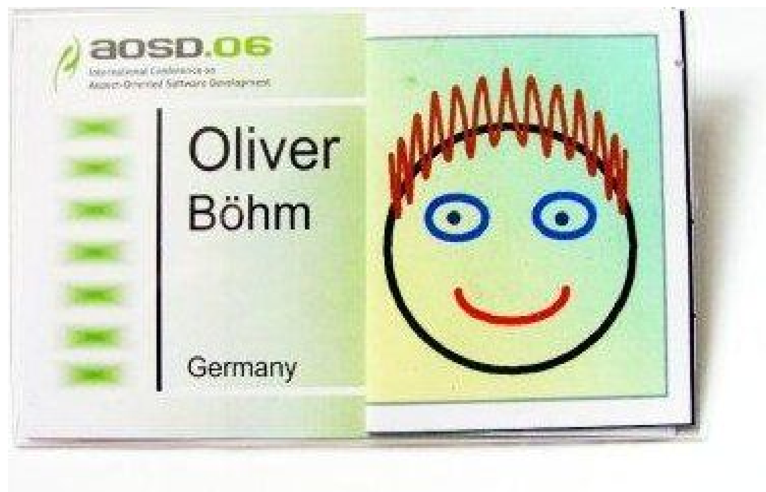
wichtige Veranstaltungen 2008/2009

agentes

- **18. September 2009: AOP-Abend**
  - Alte Scheuer / Stuttgart
  - http://www.jugs.org
- **März 2009: aosd.09**
  - http://www.aosd.net/2009/
- **geplant: AOP-Day/Camp 09**
  - Alte Scheuer / Stuttgart
  - http://www.jugs.org/
- **Anfang Juli 2009: Java Forum Stuttgart**
  - über 1000 Besucher
  - http://www.jfs2009.de

© agentes AG

Vielen Dank



agentes AG

Oliver Böhm

oli.blogger.de
oliver.boehm@agentes.de

Telefon 0711 / 25857 - 207
Telefax 0711 / 25857 - 299

Räpplenstaße 17
70191 Stuttgart

© agentes AG