Aspect Orientated Programming
mit Spring.NET

26. August 2008

Gerhard Schlemm

gerhard.schlemm@comma-soft.com
http://www.infonea.de, XING-Profil

Agenda

- Motivation
- Requirements und Concerns
- Aspect Orientated Programming
  - Definition und Begriffe
  - Demo
- AOP-Produkt für .NET
- Fazit

Motivation (I)

- Objektorientierte Programmierung (OOP)
  - Ist heute vorherrschendes Paradigma in der Softwareentwicklung
    - Grundlegende Arbeiten 60'er Jahre (Simula)
    - Durchbruch 90'er Jahre
  - OOP erlaubt effiziente Entwicklung komplexer Systeme
    - durch etablierte Konzepte (Kapselung, Abstraktion, Polymorphie, ...)
    - Entwurfsmuster, Vorgehensmodelle und Werkzeuge

Motivation (II)

- Objektorientierte Programmierung (OOP)
  - Wesentlich ist **Modularisierung**
    - Diese fördert Flexibilität, Wartbarkeit, Wiederverwendbarkeit, Testbarkeit, effiziente Entwicklung ...
    - Einheiten der OOP-Modularisierung sind Klassen/Objekte

Motivation (III)

- Und dennoch ...

```
public class BankAccount
{
    public void WithdrawAmout(string user, decimal amout)
    {
        if (!this.GetUserIsAuthenticated(user))
            performAuthentication();
        this.Lock();
        this.Logging.Log(string.Format("START: ..."));
        invalidateCache();
        this.balance -= amount;
        this.Unlock();
        this.Logging.Log(string.Format("END: ..."));
    }
    ... ...
}
```

(Vorsicht, überspitzt und konstruiert ;-))

Motivation (IV)

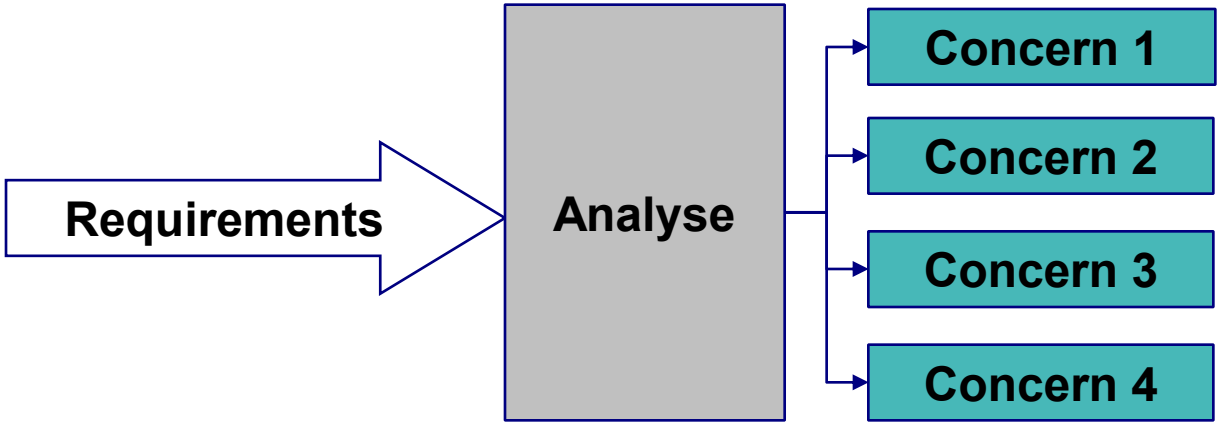
- Problem "Code Tangling"
  - Wo ist die Kernfunktionalität?
  - Der Code für Authentication, Caching, Logging und Locking hat mit der Kernfunktionalität nicht viel zu tun, ist aber in der Methode mit dieser *verwoben*
  - Der Entwickler jeden Moduls muß sich mit Authentication, Caching, etc. auseinandersetzen
    - Produktivität gehemmt
  - Wiederverwendung schwieriger

Motivation (V)

- Problem "Code Scattering"
  - Betrachte weitere Klassen und Methoden:
    Der Code, der Authentication, Caching, etc. erbringt, ist verteilt
    - zumindest der Glue-Code
  - Damit nicht einfach zu erfassen
  - Problem Weiterentwicklung:
    konzeptuelle Änderungen betreffen viele Module

Concerns (I)

- Anforderungen an Software
  - **Requirements** untergehen einer Analyse und führen zu **Concerns**
  - [Achtung: hier unscharfe Verwendung der Begriffe]



Concerns (II)

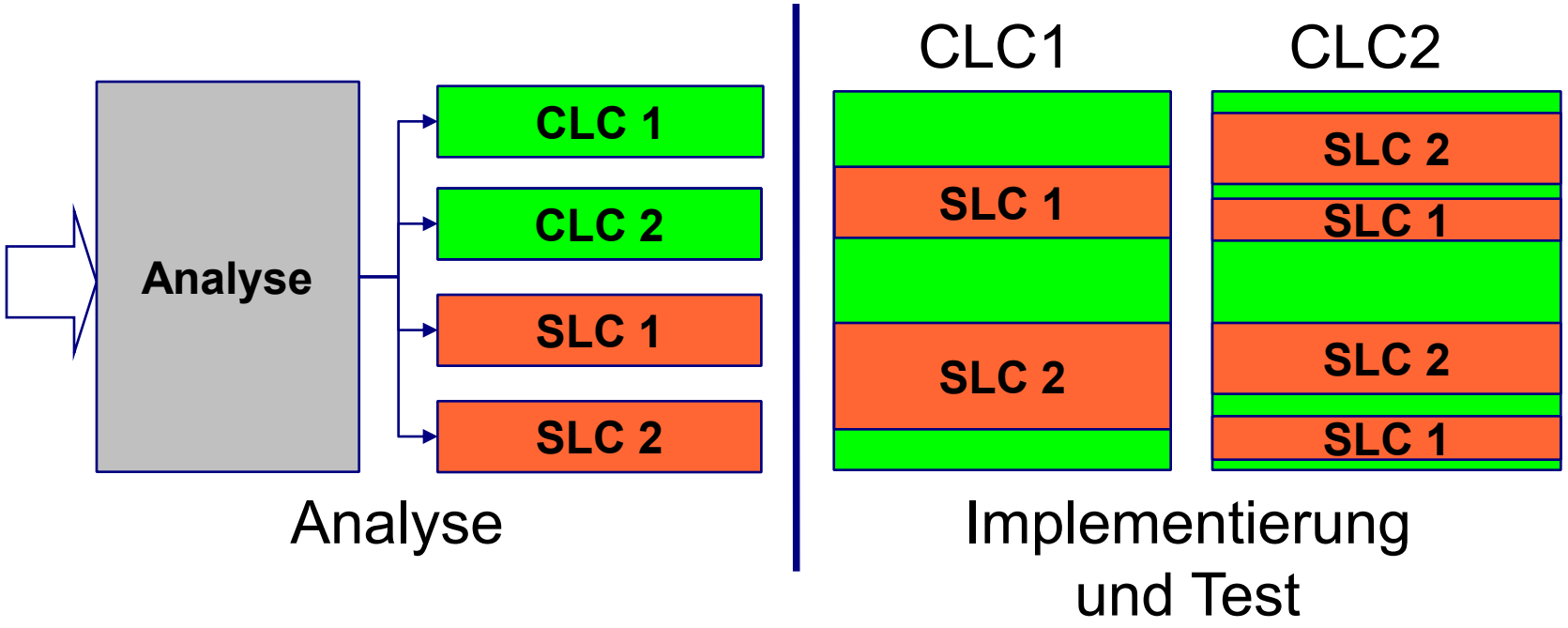
- **Core-Level Concerns**
  - betreffen den logischen Kern der Software
  - Alternative Bezeichnungen
    - "functional [requirements]"
    - "module-level concerns"
  - Im Beispiel:
    - `this.balance -= amount;`

Concerns (III)

- **System-Level Concerns**
  - betreffen das gesamte System und/oder technische Randbedingungen
    - Häufig orthogonal zueinander
    - Nicht ausschließlich technisch, auch fachlich
      - z.B. "Logging" zur Revisionssicherheit
  - Im Beispiel:
    - Authentication, Caching, Logging, Locking
  - Alternative Bezeichnungen
    - "non-functional [requirements]"

Concerns (IV)

- System-Level Concerns werden häufig **Cross Cutting Concerns** bezeichnet (CCCs)



Concerns (V)

- Implementierung konzentriert sich i.d.R. auf Core-Level Concerns
  - "Tyranny of Dominant Decomposition"
  - CCCs spielen eine untergeordnete Rolle
    - Sie fließen mit den bekannten unerwünschten Effekten in die Implementierung ein.
  - Grundproblem: Concerns sind multidimensional, Implementierungsmethodik eindimensional

Concerns (VI)

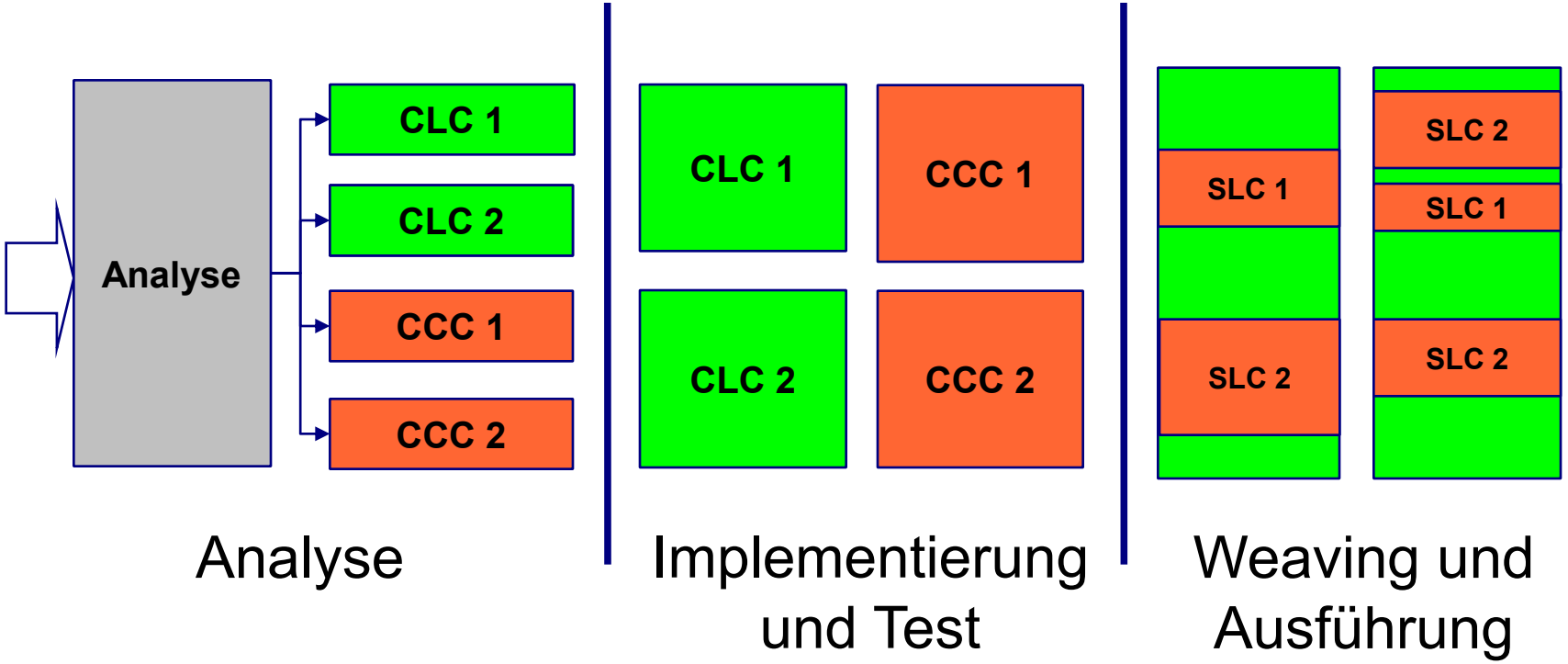
Beispiele für Cross Cutting Concerns (CCCs)

- Indexing
- Persistence
- Encryption
- Role-based security
- Authentication
- Logging
- Tracing
- Performance Counters
- Caching
- Resource-Pooling
- Transactions
- Parameter Validation
- Retry
- Domänenspezifische, fachliche Concerns

Aspect Orientated Programming (I)

- Ziel: CCCs besser Rechnung zu tragen
- Schritte: Cross Cutting Concerns (CCCs)
  - Identifizieren
  - Modularisieren (Modularisierungeinheit: **Aspekt**)
  - Module separat implementieren und testen
  - danach mit der Kernfunktionalität verbinden (Weaving)
- AOP möchte OOP ergänzen, nicht ablösen!
- Historie: Xerox Parc, 1996, AspectJ als erstes Produkt 1998, Grundideen aber älter

Aspect Orientated Programming (II)



Aspect Orientated Programming (III)

- Konkrete Schritte
  - Methodik zum Identifizieren der CCCs anwenden
    - z.B. Use Case-orientierte Methodik
    - [hier nicht mehr darüber]
  - Entwicklung der Module
    - Module (OOP:Klassen, AOP:Aspekte) werden wie gewohnt entwickelt
    - [hier nicht mehr darüber]
  - Weaving spezifizieren
    - **Was** soll **wie** verwoben werden?

Weaving (I)

- Weaving
  - Verbinden der Implementierungen:
    Im Kontrollfluss der Anwendung soll an bestimmten Punkten (Pointcut) Code einer Aspektimplementierung (Advice) ausgeführt.

  - **Advice**: Was soll ausgeführt werden?
  - **Pointcut**: Wann/wo soll es ausgeführt werden?

Weaving (II)

- **Joinpoint**
  - Definition: Stelle, an der potentiell ein Advice ausgeführt werden kann
  - Mögliche Joinpoints abhängig vom AOP-Produkt
    - Methodengrenzen , wobei Zugriff auf Properties als Methode aufgefasst wird
      - Eventuell nur virtuelle Methoden
    - Einige Produkte erlauben zusätzlich
      - Konstruktoraufrufe, Zugriff auf Felder
  - Ein Pointcut (s.o.) ist eine Menge von Joinpoints

Advices (I)

- Advice-Typen
  - **Before**
    - Vor Methodenaufruf (Joinpoint) Advice ausführen
    - Danach geht es stets mit der Methode weiter
      - Ausnahme: Advice wirft Exception
  - **Around**
    - Vor und nach dem Methodenaufruf Code ausführen
    - Ausführung der Methode kann unterbunden werden

Advices (II)

- Advice-Typen
  - **Throws**
    - Code wird bei Auftreten einer Exception im Methodenaufruf ausgeführt
  - **After Returning**
    - Code wird nach Durchführung einer Methode ohne
  - **Introduction**
    - Objekt soll per AOP zusätzliche Schnittstelle implementieren

Aspect Orientated Programming

DEMO

AOP mit Spring.NET

Weaving Strategien (I)

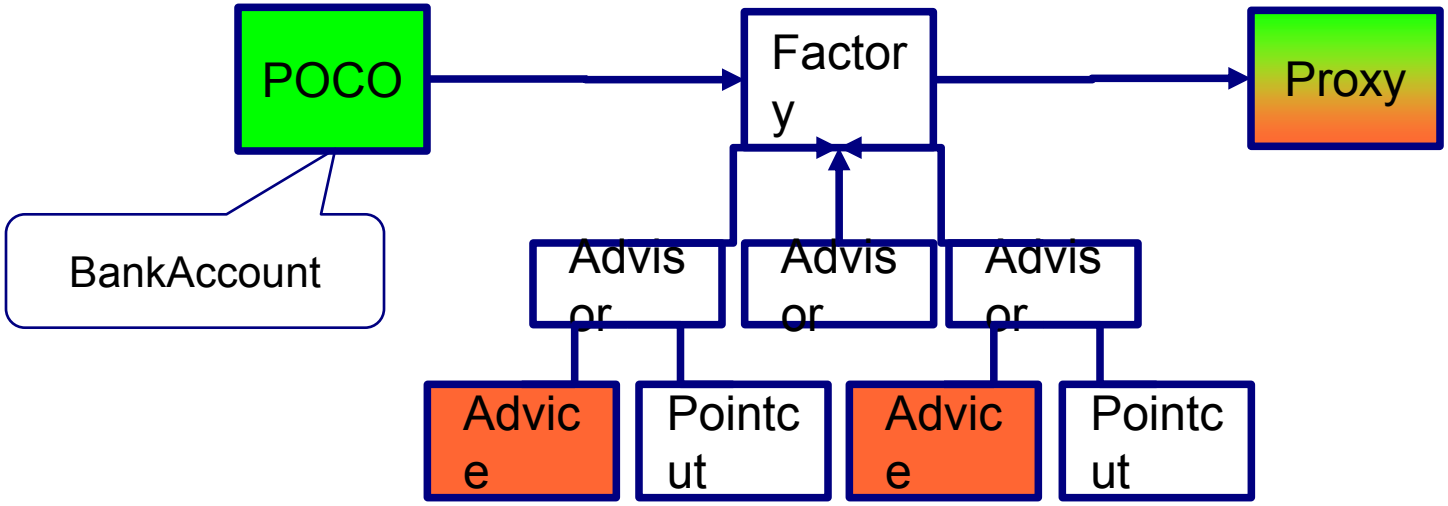
- **Wann** wird verwoben?
  - Compile-Time
  - Runtime [->Spring.NET]
- **Was** wird verwoben?
  - Quelltext
  - (C)IL [->Spring.NET]
  - Bytecode
  - JIT

Weaving Strategien (II)

- Wie müssen die mit AOP zu erweiternde Komponenten beschaffen sein?
  - Steht der Code der Core-Level Concerns zur Verfügung?
  - Muß der Code auf Erweiterbarkeit ausgelegt sein?
    - virtuelle Methoden, keine Versiegelung ("sealed")

Weaving Strategien (III)

- Proxy-Ansatz
  - Funktioniert AOP nur dann, wenn das Objekt nicht mit "new", sondern einer speziellen Factory erzeugt wurde?



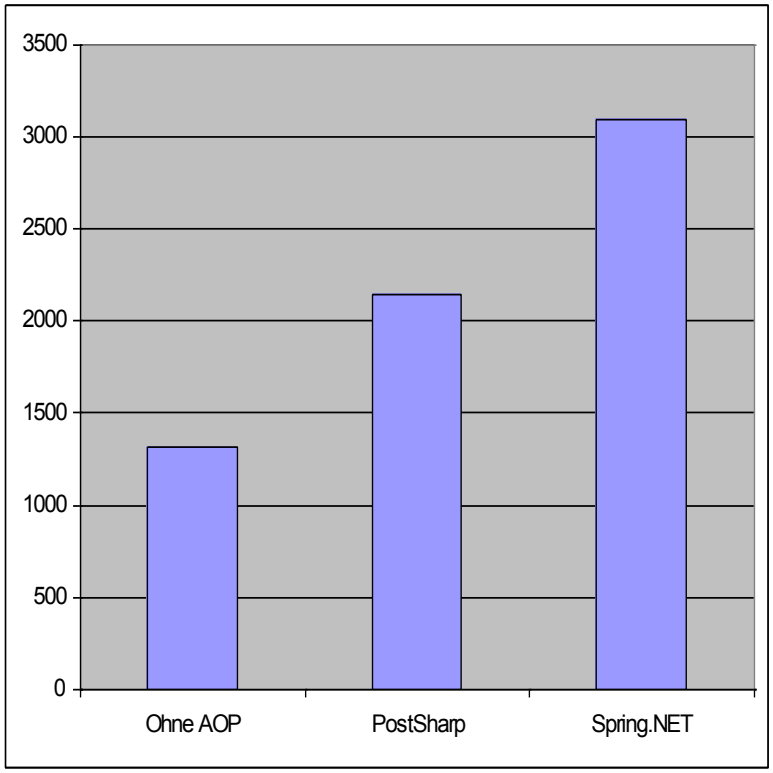
Weaving Strategien (IV)

- AOP-Produkte unterscheiden sich stark in verwendeten Weaving Strategien

- Strategie hat massive Auswirkungen auf
  - Funktionsumfang
  - Laufzeitperformance
  - Lernkurve in der Entwicklung

Weaving Strategien (V)

- Performance-Vergleich
  - [Naiv implementiert!]
  - 10000 Objekte erzeugen
  - jeweils 3 Operationen ausführen, Ergebnisse speichern (AOP-After Returning Advice)
  - Ergebnisse über IEnumerable (AOP-Introduction abrufen)
  - Messe Zeit in [ms]



AOP .NET Produkte (I)

- AOP in Spring.NET
  - Weaving zur Laufzeit, auf IL-Ebene
  - Mit AOP zu erweiternde Komponente(n)
    - Sourcen werden nicht benötigt
    - Methoden müssen nicht **`virtual`** sein
    - Klasse dürfen **`sealed`** sein
    - Aber: Möchte man gegen die zu dekorierende Klassen selbst arbeiten anstelle gegen deren Schnittstellen, gelten andere Regeln

AOP .NET Produkte (II)

- Spring.NET
  - AOP funktioniert nur auf Proxies (Factory), nicht bei **`new`** (dynamische Proxy-Generierung)
    - AutoProxying für Objekte, die im Spring.NET IoC leben ist möglich
    - Dies kann eine starke Einschränkung sein!
  - Aspektbibliothek ist Teil von Spring.NET
  - Spring.NET punktet bei Vergleich mit gutem Community-Support und "reifen" Versionen

AOP .NET Produkte (III)

- **PostSharp (1)**
- **Aspect#**
- **AspectDng**
- Rapier-Loom.NET
- Gripper-Loom.NET
- NAspect
- Encase
- Sesar
- SetPoint
- Compose
- Spring.NET (3)
- Policy Injection Block (2)
- Castle – Dynamic Proxy
- NKalore
- NAop
- AOP.NET
- Aspect.NET
- DotSpect
- Wicca
- ...

Legende: (N) Rating in CodeProject-Artikel (s.u.), Fettdruck für meistgenannt

AOP .NET Produkte (IV)

- AOP verfügbar für
  - WPF (Spring.NET 1.2 alpha)
  - Silverlight 2.0 (PostSharp)
  - Mono (PostSharp)
  - ASP.NET MVC ("Filter" - eingebaut)
  - .NET Compact Framework (PostSharp)

Fazit

- AOP "has a point"
  - Ein durchaus berechtigtes Paradigma!
- Allerdings
  - AOP ist noch nicht besonders etabliert
  - Reife der AOP-Produkte im .NET-Bereich gering
- Wünschenswert
  - Engere Integration mit Programmiersprachen und Entwicklungswerkezugen wie Visual Studio

Fragen?

Links

- JavaWorld-Artikel zu AOP
  http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html
- Wikipedia-Artikel
  http://de.wikipedia.org/wiki/Aspektorientierte_Programmierung
- Spring.NET
  http://www.springframework.net
- Vergleich Weaving-Strategien
  http://www.ayende.com/Blog/archive/2007/07/02/7-Approaches-for-AOP-in-.Net.aspx
- Vergleich von .NET-AOP-Frameworks
  http://www.codeproject.com/KB/cs/AOP_Frameworks_Rating.aspx
  http://csharp-source.net/open-source/aspect-oriented-frameworks

Ergänzungen

Interessante (Rand-)Anwendungen

- Funktionalität erweitern, auch wenn keine Quelletext vorliegen
- AOP gibt Anlass, die Kommunikation über Events neu zu überdenken
- Softwaretests
  "Ein weiteres Einsatzgebiet ist das Software-Testen, wo insbesondere das Einführen neuer Member in Klassen ohne die Veränderung ihrer Quelltexte (Inter-type Declarations) neue interessante Möglichkeiten für die Entwicklung von White-Box-Tests darstellt, z. B. um ein Tracing privater Member durchzuführen."

Alternativen zu AOP

- MixIn-Technik
  - IoC – der Business-Logic Komponente wird eine Klasse z.B. für das Logging von außen übergeben
  - Wert tut das?
  - Der Glue-Code bleibt bestehen!
- Patterns wie Vistor
  - haben dasselbe Problem
- Domänenspezifische Lösungen

Interessante (Rand-)Anwendungen

- Funktionalität erweitern, auch wenn keine Quelletext vorliegen
- AOP gibt Anlass, die Kommunikation über Events neu zu überdenken
- Softwaretests
  "Ein weiteres Einsatzgebiet ist das Software-Testen, wo insbesondere das Einführen neuer Member in Klassen ohne die Veränderung ihrer Quelltexte (Inter-type Declarations) neue interessante Möglichkeiten für die Entwicklung von White-Box-Tests darstellt, z. B. um ein Tracing privater Member durchzuführen."