

Ende des vergangenen Jahres hat Telligent sein CMS der Open-Source-Community übergeben. Es ist nunmehr über CodePlex verfügbar. Zusätzliche Bibliotheken können daneben von der Graffiti-Website heruntergeladen werden.

Info: [graffiticms.codeplex.com](http://graffiticms.codeplex.com)

[graffiticms.com/lib/lib.zip](http://graffiticms.com/lib/lib.zip)

## .NET UG Bremen

### Bitzalk Server

Eine Einführung  
Alexander Wesner

Der Biztalk Server von Microsoft ist für den einen ein Segen, für den anderen nach wie vor die große unbekannte Blackbox. Obwohl die Markteinführung bereits im Jahr 2000 stattfand, fristet der Biztalk Server in der .NET-Entwickler-Community eher ein Schattendasein. Das Treffen holt den Biztalk Server ans Licht. In dem Vortrag werden einige Einsatz-Gebiete und -Szenarien erläutert. Anhand eines praktischen Beispiels wird gezeigt, wie man Lösungen mit Hilfe des Biztalk Server umsetzt und implementiert.

10.03.2010  
Bremen

Info:  
[www.dotnet-hb.de/  
default.aspx?page=1](http://www.dotnet-hb.de/default.aspx?page=1)

-Anzeige-



ASP.NET AJAX:  
Web-Anwendungen mit  
Asynchronous JScript & XML

24. – 25.03.2010  
Frankfurt  
EUR 899,-

Info:  
[ppedv.de/schulung/kurse/  
ASP.NET2.0-AJAX-  
AsynchronousJScriptXml-  
JSON-Toolkit-Webservice.aspx](http://ppedv.de/schulung/kurse/ASP.NET2.0-AJAX-AsynchronousJScriptXml-JSON-Toolkit-Webservice.aspx)

## REFACTORING

Rico Fritzsche

# Besserer Code mit Refactoring

Dieser Artikel beschäftigt sich mit einem immer wichtiger werdenden **Thema der Softwareentwicklung**. In den letzten Jahren hat sich ein Begriff in den Entwicklerfokus geschoben, der zunehmend an Bedeutung gewinnt: **Refactoring**.

Als ich begann, mich in dieses Thema zu vertiefen, fragte ich mich, welche Bedeutung eigentlich hinter diesem Begriff steckt.

Bei Martin Fowler [1] fand ich folgende Definitionen:

- *Refactoring (noun):* A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
- *Refactor (verb):* To restructure software by applying a series of refactorings without changing its observable behavior.

Im Grunde geht es darum, bestehenden Code zu bereinigen und zu verbessern sowie interne Strukturen des Systems zu ändern, ohne dabei das Verhalten der Software nach außen zu beeinträchtigen. Dies heißt aber auch, dass es nicht darum geht, neue Funktionen zu implementieren, sondern bestehenden Code oder das Design der Software quasi »unter der Haube« zu überarbeiten. Im Grunde ist »Refactoring« mit dem Begriff »Code Cleanup« gleichzusetzen, da beide Begriffe dasselbe meinen.

Refactoring ist dabei nicht mit Performance-Optimierung zu verwechseln, da Refactoring das ausschließliche Ziel verfolgt, Code besser zu strukturieren und damit lesbarer sowie überschaubarer zu gestalten – beispielsweise für andere Entwickler, die zukünftig das System warten müssen.

### Reflektion als ständiger Prozess

Der Ausgangspunkt sollte immer die intensive Auseinandersetzung und die Reflektion über die persönliche Arbeit sein. Ich denke, dass diese Reflektion keine Aufgabe ist, die nur irgendwann einmalig ausgeführt werden muss. Ich sehe dies als stetigen Prozess der Weiterentwicklung der eigenen Fähigkeiten. Mir fällt in diesem Zusammenhang Stefan Liesers und Ralf Westphals Definition des professionellen Softwareentwicklers ein.

Diese determiniert essenziell den Qualitätsmaßstab in der Softwareentwicklung in der Auseinandersetzung mit der Materie, der damit verbundenen stetigen Reflektion über das eigene Handeln sowie einem Wertesystem, das sich grundsätzlich durch Prinzipien und Best Practices beschreibt [4]. Folgerichtig ist es empfehlenswert, sich das Wertesystem der »Clean Code Developer«-Initiative genauer anzuschauen [5].

Am Einfachsten ist Refactoring, wenn es Bestandteil des Entwicklungsprozesses an sich ist. Im Prinzip ist dies nichts anderes, als sich laufend mit seiner Arbeit

auseinanderzusetzen und sich damit stetig weiter zu entwickeln. Hier ist beispielsweise von Vorteil, dass ein wenig erfahrener Entwickler durch einen Erfahrenen beim Review des Codes unterstützt oder in bestimmten Entwicklungsabschnitten auf Pair Programming [6] gesetzt wird. Oft merkt man, dass man schon mit recht einfachen Mitteln eine Menge erreichen kann. Das ist sicherlich der einfachere Weg.

Schwieriger ist Refactoring meist, wenn man eine bereits bestehende Anwendung auf den Tisch bekommt, bei deren Entwicklung man sich eben nicht an ein Wertesystem [2] gehalten hat, und nun Fehler in dieser Anwendung beheben soll. Dabei besteht schon die erste Schwierigkeit, den vorhandenen, unstrukturierten Code zu verstehen, Bugs zu finden und bei deren Behebung nicht gleich wieder neue einzubauen. Diese Fakten sind zumeist die Intention, mit einem Refactoring des Codes zu beginnen.

In solchen Szenarien ist es häufig so, dass »Superklassen und -methoden«, also die berühmt-berüchtigten »eierlegenden Wollmilchsäue«, verstanden und in kleinere, übersichtlichere Teile zerlegt werden müssen. Der erste Schritt sollte dabei stets darin bestehen, sinnvolle Unit-Tests zu erstellen – falls noch keine vorhanden sind – und diese während des Refactoring-Prozesses immer wieder auszuführen. Unit-Tests sind integraler Bestandteil des Refactorings. Allerdings werde ich hier nicht weiter auf die erforderlichen Unit-Tests eingehen, da dies den Rahmen des Artikels sprengen würde.

Typische Beispiele für Refactoring sind auch das Ändern von Namen (»Rename Method« [1]) oder das Ersetzen eines Codeabschnittes durch einen Methodenaufruf (»Extract Method« [1]). Martin Fowlers Werk [1] gibt einen hilfreichen Katalog mit Ausgangssituationen und Lösungsansätzen vor.

### Ein schlichtes Beispiel

Ein recht häufiger Fall, bei dem Refactoring notwendig wird, besteht darin, lange und unleserliche Methoden aufzusplitten. Lange Methoden sind meist fehleranfällig, schwer verständlich und schlecht wartbar. Eine einfache Variante des Refactoring ist das Extrahieren von Methoden aus dem bestehenden Code. Damit wird der Code besser lesbar, ohne dass Veränderungen an der Funktionalität stattfinden. Dies ist auch notwendig, wenn man ein und denselben Code an verschiedenen Stellen in der Anwendung findet. Diese Variante des Refactoring nennt man »Extract Method« [1].

## RICO FRITZSCHE

E-Mail:  
[rico.fritzsche@visual-world.de](mailto:rico.fritzsche@visual-world.de)  
Web:  
[www.visual-world.de](http://www.visual-world.de)



Rico Fritzsche ist seit Mitte der Neunziger Jahre in der IT Branche tätig und seit 2006 Softwareentwickler und -architekt sowie Technologieberater bei dem auf Microsoft-Technologien spezialisierten Unternehmen VISUAL WORLD. Seine technologischen Schwerpunkte liegen in der Entwicklung von verteilten Anwendungen, WCF, MS SQL Server und Datenzugriffstechnologien wie beispielsweise ADO.NET und Entity Framework.

Nehmen wir einmal folgende Methode und stellen uns vor, sie tut einfach zu viele Dinge. Ein Weg zu mehr Übersicht zu gelangen, besteht darin, diese in mehrere kleine Methoden zu zerlegen.

```
void Foo()
{
    //... do something!!!

    // Calculate price
    _result = order.Price + order.PackingPosts
    + order.ShippingCosts;

    _result -= _discount;

    //... do anything else...
}
```

Wir lösen nun die Berechnung der Kosten aus unserer Methode heraus und erstellen dafür eine neue Methode. Im Ergebnis sieht es wie folgt aus:

```
public void Foo()
{
    //... do something!!!

    this.CalculatePrice(order);

    //... do anything else...
}

void CalculatePrice(Order order)
{
    _result = order.Price + order.PackingPosts
    + order.ShippingCosts;
    _result -= _discount;
}
```

Nutzt man für diese Arbeit beispielsweise JetBrains »Resharper« [3], kann man den zu extrahierenden Abschnitt im Code markieren, per Rechtsklick »Refactor« und dann »Extract Method« auswählen. Es erscheint ein Dialog (siehe Abbildung 1), in dem man die gewünschten Einstellungen für die Erstellung der Methode vornehmen kann.

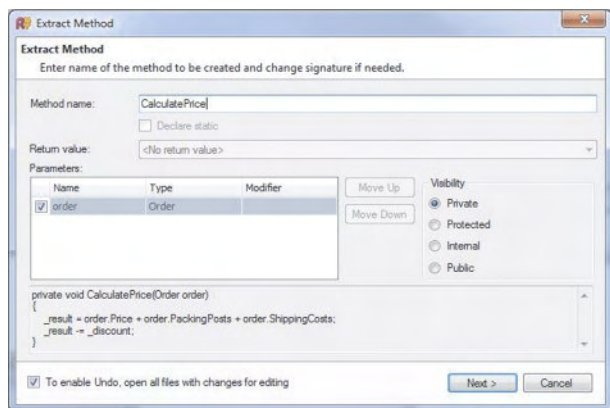


Abb. 1: Extrahieren einer Methode mit dem „Resharper“ von JetBrains

### Eine Fallstudie

Die Vielfalt der Möglichkeiten des »Refactorings« kann ein solcher Artikel natürlich nicht aufzeigen. Hier empfiehlt sich in jedem Falle weiterführende Lektüre. Meine Ausführungen sollen aber anregen, sich mit diesem Thema näher zu beschäftigen.

In der nun folgenden Fallstudie, an ein Beispiel aus Martin Fowlers Buch angelehnt, möchte ich auf einfache Art und Weise zeigen, wie simpel es ist Code zu verbessern, lesbarer und damit entscheidend wartbarer zu machen. In der Praxis sehen solche Code-Stellen natürlich meist viel komplizierter aus und wachsen mit zunehmender Komplexität der Anwendung schnell zu unübersichtlichen Monstern heran.

Martin Fowler bezeichnet diese Art des Refactoring im Übrigen als »Replace Conditional with Polymorphism«.

Dieses Beispiel veranschaulicht im Wesentlichen, wie man schlechtes, nicht objektorientiertes Design durch ein gut strukturiertes und objektorientiertes Design ersetzt. Das Verhalten einer Methode »PayAmount« wird in Abhängigkeit zum Objekttyp gebracht. Die Auswertung von Strings oder Enums, um ein entsprechendes Verhalten herbeizuführen, entfällt damit.

Die Klasse Employee im folgenden Code-Ausschnitt definiert über den Enum EmployeeType die Art des Mitarbeiters. Dieser kann beispielsweise vom Typ »Verkäufer« (Salesman) sein.

Die Methode PayAmount führt nun anhand der Art des Mitarbeiters eine gewisse Berechnung aus, was in der Realität wahrscheinlich durchaus komplexer sein würde als in diesem Beispiel.

```
public class Employee
{
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public EmployeeType EmployeeType { get; set; }
    public double MonthlySalary { get; set; }
    public double Bonus { get; set; }
    public double Commission { get; set; }

    public double PayAmount()
    {
        switch (this.EmployeeType)
        {
            case EmployeeType.Engineer:
                return MonthlySalary;
            case EmployeeType.Salesman:
                return MonthlySalary + (Commission * 1.5);
            case EmployeeType.Manager:
                return MonthlySalary + Bonus;
            default:
                throw new ArgumentOutOfRangeException(
                    "Unknown Type.");
        }
    }
}

public enum EmployeeType
{
    Engineer, Salesman, Manager
}
```

Grundsätzlich ist es immer vorteilhafter, Typen zu definieren anstatt Unterscheidungen über Eigenschaften und deren Abprüfung über if-else-Blöcke oder switch-case-Anweisungen vorzunehmen. Grund dafür sind natürlich bessere Lesbarkeit und Übersichtlichkeit und damit verbunden auch die Wartbarkeit des Systems sowie die Einhaltung von Paradigmen der Objektorientierung. Es ist nicht objektorientiert, sondern eine typisch prozedurale Herangehensweise, Entscheidungen über Bedingungen zu prüfen.

Mitarbeiter können, wie schon gesagt, im Unternehmen verschiedene Verantwortlichkeiten haben und divergente Tätigkeiten ausführen. Es gibt Eigenschaften, die allen Arten von Mitarbeitern gemein sind, wie zum Beispiel Vorname, Nachname und so weiter. In der Art der Berechnung ihrer Vergütung unterscheiden sie sich allerdings. Das ist Grund genug, Typen mit konkreten Implementierungen der Vergütungsberechnung zu schaffen. Dies kann man etwa durch Vererbung und Polymorphie erreichen. Polymorphie bedeutet in diesem Kontext, dass Methoden mit ein und derselben Signatur in verschiedene Klassen implementiert werden (siehe Abbildung 2).

Anzumerken sei hier jedoch noch, dass Vererbung kein Allheilmittel und nicht überall angebracht ist. Es gibt eine Menge Fälle in denen andere Ansätze der Vererbung vorzuziehen sind.

Die Klasse Employee muss zur Basisklasse gemacht werden. Die Eigenschaft EmployeeType benötigen wir

## .NET-Anwendungen für das iPad

Auch für das kürzlich von Apple vorgestellte iPad können nun mit den Entwicklertools „MonoTouch“ von Novell Anwendungen entwickelt werden. Das MonoTouch-SDK war bereits im vergangenen Jahr als Technik zum Entwickeln von .NET-Anwendungen für Apples iPhone und iPod Touch veröffentlicht worden. Laut Novell gehen Entwickler konform mit Apples Lizenzbedingungen, wenn sie dieses SDK verwenden.

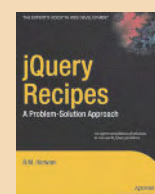
Die Version 1.9 von MonoTouch stellt bezüglich der iPad-Entwicklung noch eine Alpha-Version dar – die „fertige“ Ausgabe soll in diesen Wochen verfügbar sein. Zusätzlich zu dem MonoTouch-SDK werden noch das iPhone-SDK 3.2 und die Entwicklungsumgebung MonoDevelop benötigt.

Anders als auf Desktops kann die dem Entwicklerpaket zugrunde liegende .NET-Alternative Mono auf den Apple-Geräten keine JIT-Technik (Just in Time) zum Übersetzen des CIL-Bytecodes (Common Intermediate Language) in Maschinencode benutzen. Apple lässt solche JIT-Engines auf seinen Geräten nicht zu. MonoTouch enthält deshalb einen Compiler, der .NET-Executables und -Bibliotheken in den Maschinencode von iPhone & Co. übersetzt.

MonoTouch gibt es in drei Edition: in einer „Personal Edition“, in einer „Enterprise Edition“ und in der „Enterprise Edition 5“. Die Personal-Variante für Privat- und Hobbyentwickler kostet 399 USD, die Enterprise Edition in der Standardausführung mit 999. Mit Lizenzen für fünf Entwickler kostet es spürbar mehr, nämlich 3 999 USD.

Info: [monotouch.net/ipad](http://monotouch.net/ipad)

## jQuery



350 Seiten  
englisch  
erschienen 01/10  
ISBN:  
978-1-4302-2709-0  
USD 44,99

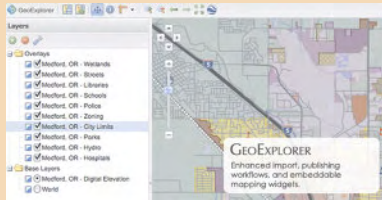
jQuery Recipes  
A Problem-Solution Approach  
Bintu Harwani  
Apress

## Unterhaltung



350 Seiten  
deutsch  
erschient 02/10  
ISBN:  
978-3-446-42077-9  
EUR 24,90

Heinz" Life  
Kleine Geschichte vom Kommen und Gehen des Computers  
Lutz Heuser  
Hanser

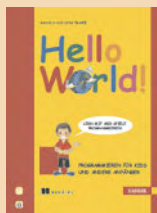


Eine integrierte Open-Source-Plattform zur Bereitstellung von Karten und Geodaten gibt es mit der „Open-Geo Suite 1.0“. Diese Suite umfasst die Komponenten „GeoServer“, „OpenLayers“, „GeoWebCache“, die PostgreSQL-Erweiterung „PostGIS“ und das JavaScript-Toolkit „GeoExt“.

Sie steht als Community Edition zum freien Download zur Verfügung. Sie ist aber auch mit Support, Training und zusätzlichen Optionen wie Unterstützung für Oracle-Datenbanken und Cluster-Installationen erhältlich – allerdings kostet dies dann zwischen 8000 und 98000 USD pro Jahr. Der Hersteller OpenGeo gehört der Non-Profit-Organisation TOPP (The Open Planning Project) an, deren Arbeitsschwerpunkte in den Bereichen Öffentliche Hand und Verkehr liegen.

Info: [opengeo.org/products/suite](http://opengeo.org/products/suite)

## Programmieren für Kids



430 Seiten  
deutsch  
erschienen 11/09  
ISBN:  
978-3-446-42144-8  
EUR 29,90

**Hello World!**  
Programmieren für Kids und andere Anfänger  
Warren D. Sande, Carter Sande  
Hanser

## Debuggen



230 Seiten  
englisch  
erschienen 11/09  
ISBN:  
978-1-934-35628-9  
EUR 34,00

**Debug It!**  
Find, Repair, and Prevent Bugs in Your Code  
Paul Butcher  
O'Reilly

## do.Net Dortmund

03.03.2010  
18:00 Uhr  
Lünen

### Info:

[www.xing.com/events/net-user-group-dortmund-treffen-03-18-00-entity-framework-linq-to-entity-oliver-lohkamp-465027](http://www.xing.com/events/net-user-group-dortmund-treffen-03-18-00-entity-framework-linq-to-entity-oliver-lohkamp-465027)

nun nicht mehr, da diese Klasse eine konkrete Implementierung verlangt, welche den Typ des Mitarbeiters bestimmt. Die Methode PayAmount wird als »abstract« gekennzeichnet und erzwingt damit in der konkreten Implementierung, dass die Methode überschrieben und die mitarbeiterspezifische Berechnung der Vergütung implementiert werden muss:

```
public abstract class Employee
{
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public double MonthlySalary { get; set; }
    public double Bonus { get; set; }
    public double Commission { get; set; }

    public abstract double PayAmount();
}
```

Für jeden Mitarbeitertyp erfolgt nun die Implementierung einer konkreten Klasse.

Die Klasse Engineer erbt von der Basisklasse Employee. Diese erzwingt, dass alle als abstract gekennzeichneten Member (in diesem Falle PayAmount) implementiert werden müssen. Ist – wie im Beispiel gezeigt – JetBrains Resharper installiert, weist uns dieser darauf hin, dass die entsprechenden Member benötigt werden. Per Rechtsklick (siehe Abbildung 3) auf das rote »Lämpchen« ist es möglich, die Aktion »Implement Members« auszuwählen, um die Methoden-Bodys automatisch erstellen zu lassen.

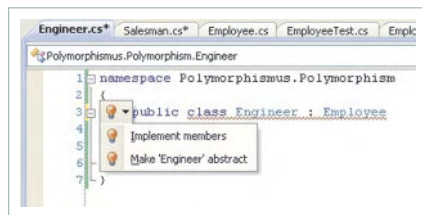


Abb. 3: Refactoring in Resharper

Diese werden dann, versehen mit einer NotImplementedException, in die Klasse geschrieben:

```
public class Engineer : Employee
{
    public override double PayAmount()
    {
        throw new NotImplementedException();
    }
}
```

Nun müssen wir die Methode nur noch mit unserer konkreten, für den Mitarbeitertyp erforderlichen Berechnungslogik füllen. In unserem Beispiel ist die Berechnung natürlich nur exemplarisch und stark vereinfacht:

```
public class Salesman : Employee
{
    public override double PayAmount()
    {
        return this.MonthlySalary + (this.Commission + 1.5);
    }
}
```

## Fazit

Was haben wir bis hierhin schon gewonnen? Ich denke, essenziell schon viel Übersichtlichkeit. Durch die strenge Typisierung der Objekte haben wir zudem die Fehleranfälligkeit wesentlich verringert.

Aber was macht Refactoring nun wirklich so bedeutend? Ich möchte zur Beantwortung dieser Frage wieder einmal etwas weiter ausholen. In den frühen Neunzigern (und bestimmt auch noch später) bestand der Anspruch in der Anwendungsentwicklung oftmals darin, etwas Lauffähiges beim Kunden abzuliefern. Damals

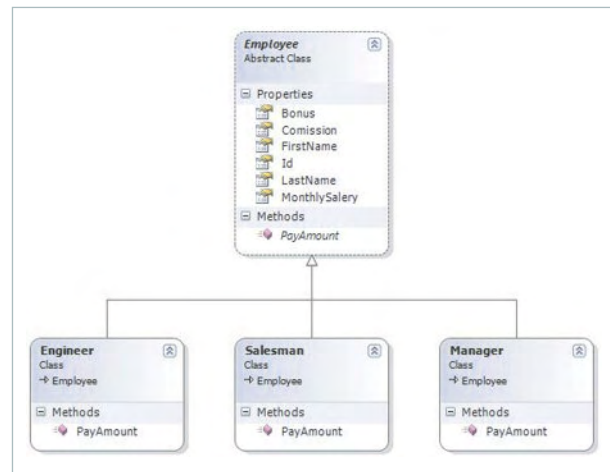


Abb. 2: Mitarbeiter berechnen ihr Gehalt sozusagen selber

wurde der Ausdruck »Never touch a running system« geprägt. Das bedeutete in etwa so viel wie: »Gut, dass soweit alles funktioniert. Nun aber bitte nichts mehr anfassen, damit auch nichts kaputt geht.«

Leider ist so etwas immer zu kurz gedacht, da Software kein unveränderliches, starres Produkt ist, sondern in der Regel an äußere Gegebenheiten angepasst werden muss. Genau diese, meist fachlich notwendigen Eingriffe, müssen von Anfang an berücksichtigt werden. Aus heutiger Sicht ist es wichtig, robuste und langlebige Systeme zu entwickeln, die jeglichen Änderungen von fachlicher Seite standhalten können und bei denen es jederzeit möglich ist, schnell und flexibel auf neue Kundenanforderungen zu reagieren.

Um die Codequalität zu verbessern, lohnt es in jedem Falle, sich diesem Thema vertieft zu widmen, sich mit dem eigenen Code und dem des Teams auseinanderzusetzen. Als weiterführende Lektüre ist erneut Martin Fowlers Buch unbedingt zu empfehlen. ■

## INFO

- [1] Martin Fowler, Refactoring. Improving the design of existing code

[www.pearsonhighered.com/educator/product/Refactoring-Improving-the-Design-of-Existing-Code/9780201485677.page](http://www.pearsonhighered.com/educator/product/Refactoring-Improving-the-Design-of-Existing-Code/9780201485677.page)

- [2] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

[www.pearsonhighered.com/educator/product/Clean-Code-A-Handbook-of-Agile-Software-Craftsmanship/9780132350884.page](http://www.pearsonhighered.com/educator/product/Clean-Code-A-Handbook-of-Agile-Software-Craftsmanship/9780132350884.page)

- [3] Resharper: [www.jetbrains.com/resharper/index.html](http://www.jetbrains.com/resharper/index.html)

- [4] Clean Code Developer: [www.clean-code-developer.de](http://www.clean-code-developer.de)

- [5] CCD-Wertesystem: [www.clean-code-developer.de/wiki/CcdWertesystem](http://www.clean-code-developer.de/wiki/CcdWertesystem)

- [6] Pair Programming: [de.wikipedia.org/wiki/Pairprogrammierung](http://de.wikipedia.org/wiki/Pairprogrammierung)