BASTA! ON TOUR

Rainer Stropek | software architects gmbh

Good Cop, Bad Cop

Steigerung der Codequalität durch StyleCop und Code Analysis

Abstract

Was ist "guter Code"? Über diese Frage lässt sich vortrefflich streiten. Es ist aber unbestritten, dass Codequalität ein Thema für jedes Team sein muss, das professionell Software erstellen möchte. Die beiden Microsoft-Werkzeuge StyleCop und FxCop bieten automatisierte Qualitätskontrolle für Code und sollten daher zum Repertoire von Entwicklerprofis gehören. Sie weisen auf Formatierungsfehler, Designfehler, Sicherheitslücken und vieles mehr hin und können in Visual Studio und Buildprozesse integriert werden. In dem Workshop zeigt Rainer Stropek, was StyleCop und FxCop leisten, gibt Tipps aus der Praxis, wie die Werkzeuge im Team nutzbringend eingesetzt werden können und gibt einen Überblick über die Anpassungsmöglichkeiten. Der Workshop ist ideal für Entwickler, die im Team arbeiten und vielleicht sogar wiederverwendbare Bibliotheken bauen.

Introduction

- software architects gmbh
- Rainer Stropek
- Developer, Speaker, Trainer
- MVP for Windows Azure
- rainer@timecockpit.com
- @rstropek

http://www.timecockpit.com

http://www.software-architects.com

Einleitung in StyleCop

STYLECOP

Was leistet StyleCop?

- Analysiert C# Source Code und überprüft Regeln bzgl. Stil und Konsistenz
  - Dokumentationsregeln
  - Layoutregeln
  - Regeln für bessere Wartbarkeit
  - Regeln für Namensschema
  - Reihenfolge der Memberdefinition
  - Lesbarkeitsregeln
  - Spacing Rules
- Open Source Projekt
  - Codeplex Site
- Version 4.4
  - Unterstützt C# 4 Syntax
- Kann sowohl interaktiv als auch im Buildprozess verwendet werden

Details zu Dokumentationsregeln siehe Workshop „Im Sandkasten – professionelle Codedokumentation mit Sandcastle & Co“

FORMATIERUNGSREGELN

Dokumentationsregeln

- SA16xx Regeln
  - Details siehe StyleCop Rule Documentation
- Jedes Element muss konsistent dokumentiert werden.
- Dokumentation muss valides XML sein.
- Leere/zu kurze/wiederholende Dokumentation nicht erlaubt ;-)
- Textprüfungen; Beispiele:
  - get/set ←→ Summary
  - Konstruktoren
- File Headers

```
//-----------------------------------------------------------
// <copyright file="Bucket.cs" company="Contoso Ltd.">
//     Copyright (c) Contoso Ltd. All rights reserved.
// </copyright>
//-----------------------------------------------------------
```

  - Mit optionaler Inhaltsprüfung

Layoutregeln

- SA15xx Regeln
  - Details siehe StyleCop Rule Documentation
- Sind die geschwungenen Klammern dort wo sie sein sollen?
- Jedes Statement in einer eigenen Zeile
- Unnötige/fehlende Leerzeilen

Regeln für Spacing

- SA10xx Regeln
  - Details siehe StyleCop Rule Documentation
- Richtige Leerzeichen bei
  - C# Keywords
  - Komma
  - Semicolon
  - Symbolen
  - Dokumentation
  - Operatoren
  - Etc.
- Tabs must not be used
  - Darüber kann man streiten…

Lesbarkeitsregeln

- SA11xx Regeln
  - Details siehe StyleCop Rule Documentation
- Aufruf eines Members mit `base.` Nicht erlaubt wenn keine lokale Implementierung existiert
  - SA1100
- Zugriff auf Instanzmembers immer mit `this.`
  - SA1101
- Regeln für die Formatierung von LINQ Abfragen
- Keine leeren Statements, Kommentare, etc.
- Diverse Formatierungsregeln (Position von Klammern, etc.)

Reihenfolgeregeln

- SA12xx Regeln
  - Details siehe [StyleCop Rule Documentation]
  - Hier nur ein Auszug
- `using`s
  - Innerhalb des Namespace
  - Sortiert
- Reihenfolge innerhalb der Klasse:
  - Fields, Constructors, Finalizers (Destructors), Delegates, Events, Enums, Interfaces, Properties, Indexers, Methods, Structs, Classes
- Innerhalb bestimmt Access Modifier die Reihenfolge
  - `public`, `internal`, `protected internal`, `protected`, `private`

Manchmal lästig, aber sehr wichtig!

NAMENSSCHEMA

Generelle Namensregeln

- Diskussionen darüber im Team? Warum nicht einfach die Regeln von Microsoft übernehmen?
- `PascalCasing` für alle Identifier mit Ausnahme von Parameternamen
  - Ausnahme: Zweistellige, gängige Abkürzungen (z.B. `IOStream`, aber `HtmlTag` statt `HTMLTag`)
- `camelCasing` für Parameternamen
  - Ausnahme: Zweistellige, gängige Abkürzungen (z.B. `ioStream`)
- Fields? Nicht relevant, da nie public oder protected ;-)
- Dont‘s
  - Underscores
  - Hungarian notation (d.h. kein Präfix)
  - Keywords als Identifier
  - Abkürzungen (z.B. `GetWin`; sollte `GetWindow` heißen)

Namen für Assemblies und DLLs

- Keine Multifileassemblies
- Oft empfehlenswert, den Namespacenamen zu folgen
- Namensschema
  - `<Company>.<Component>.dll`
  - `<Produkt>.<Technology>.dll`
  - `<Project>.<Layer>.dll`
- Beispiele
  - `System.Data.dll`
  - `Microsoft.ServiceBus.dll`
  - `TimeCockpit.Data.dll`
  - **`Transporters.TubeNetwork`**

Namen für Namespaces

- Eindeutige Namen verwenden (z.B. Präfix mit Firmen- oder Projektname)
- Namensschema
  - `<Company>.<Product|Technology>[.<Feature>][.<Subnamespace>]`
- Versionsabhängigkeiten soweit möglich vermeiden (speziell bei Produktnamen)
- Organisatorische Strukturen beeinflussen Namespaces nicht
- Typen nicht gleich wie Namespaces benennen
- Nicht zu viele Namespaces
- Typische Szenarien von seltenen Szenarien durch Namespaces trennen (z.B. `System.Mail` und `System.Mail.Advanced`)
- `PascalCasingWithDotsRecommended`

Was ist falsch (Code)?

```
namespace Transporters.TubeNetwork
{
  public class TubeNetwork
  {
    [...]
  }
}
```

Was ist falsch (Code)?

```
namespace Transporters.TubeNetwork
{
  public class Net
  {

  }
}
```

System.Net

Klassen-, Struktur- und Interfacenamen

- `PascalCasingRecommended`
- Keine üblichen oder bekannten Typnamen verwenden (z.B. keine eigene Klasse `File` anlegen)
- Klassen- und Strukturnamen
  - Meist Hauptwörter (z.B. `Window`, `File`, `Connection`, `XmlWriter` etc.)
- Interfacenamen
  - Wenn sie eine Kategorie repräsentieren, Hauptwörter (z.B. `IList`, etc.)
  - Wenn sie eine Fähigkeit ausdrücken, Adjektiv (z.B. `IEnumerable`, etc.)
- Kein Präfix (z.B. „C…“)
  - „I“ für Interfaces ist historisch gewachsen und deshalb sehr bekannt

Membernamen

- `PascalCasingRecommended`
- Methoden
  - Verb als Name verwenden (z.B. `Print`, `Write`, `Trim`, etc.)
- Properties
  - Hauptwörter oder Adjektive (z.B. `Length`, `Name`, etc.)
  - Mehrzahl für Collection Properties verwenden
  - Aktiv statt passiv (z.B. `CanSeek` statt `IsSeekable`, `Contains` statt `IsContained`, etc.)
- Events
  - Verb als Name verwenden (z.B. `Dropped`, `Painting`, `Clicked`, etc.)
  - Gegenwart und Vergangenheit bewusst einsetzen (z.B. `Closing` und `Closed`, etc.)
  - Bei Eventhandler typisches Pattern verwenden (`EventHandler`-Postfix, `sender` und `e` als Parameter, `EventArgs` als Postfix für Klassen
- Fields
  - Keine `public` oder `protected` Fields
  - Kein Präfix

StyleCop

HANDS-ON LAB 1
(10 MINUTEN)

ANWENDUNG VON STYLECOP

Setting-Files (1/2)

« SoftwareArchitects.SessionsAndTrainings ▸ StyleCop Workshop ▸ StyleCopDemo ▸

Search StyleCopDemo

Organize ▾ | Open ▾ | Burn | New folder

Favorites: ADC2010 - Azure Intro, azuretests, DemoApps, Desktop, Downloads, Recent Places

Libraries: Documents, Music, Pictures, Videos

Computer: Local Disk (C:), Local Disk (Q:)

Network

Name	Date modified	Type	Size
bin	08.12.2010 10:06	File folder	
obj	08.12.2010 10:06	File folder	
Properties	08.12.2010 10:06	File folder	
Bucket.cs	08.12.2010 10:23	Visual C# Source f...	3 KB
Local.testsettings	08.12.2010 10:06	Visual Studio Test ...	1 KB
Settings.StyleCop	08.12.2010 10:13	STYLECOP File	1 KB
StyleCop.Cache	08.12.2010 10:23	CACHE File	1 KB
StyleCopDemo.csproj	08.12.2010 10:06	Visual C# Project f...	3 KB
StyleCopDemo.csproj.vspscc	08.12.2010 10:06	Visual Studio Sour...	1 KB
StyleCopDemo.sln	08.12.2010 10:06	Microsoft Visual S...	3 KB
StyleCopDemo.suo			
StyleCopDemo.vsmdi			
StyleCopDemo.vssscc			
TraceAndTestImpact.testsettings			

Settings.StyleCop Date
STYLECOP File

Tipp: Zentrale Settings können lokal überschrieben werden

Microsoft StyleCop Project Settings

Rules | Settings Files | Results Caching | Company Information | Hungarian

Determines how to merge these settings with other settings files.

- ○ Do not merge with any other settings files
- ◉ Merge with settings file found in parent folders [Edit...]
- ○ Merge with the following settings file:

Location: [...] [Edit...]

OK | Cancel | Apply

Setting-Files (2/2)

```
<StyleCopSettings Version="4.3">
  <Analyzers>
    <Analyzer AnalyzerId="Microsoft.StyleCop.CSharp.ReadabilityRules">
      <Rules>
        <Rule Name="DoNotUseRegions">
          <RuleSettings>
            <BooleanProperty Name="Enabled">True</BooleanProperty>
          </RuleSettings>
        </Rule>
      </Rules>
      <AnalyzerSettings />
    </Analyzer>
    <Analyzer AnalyzerId="Microsoft.StyleCop.CSharp.SpacingRules">
      <Rules>
        <Rule Name="TabsMustNotBeUsed">
          <RuleSettings>
            <BooleanProperty Name="Enabled">False</BooleanProperty>
          </RuleSettings>
        </Rule>
      </Rules>
      <AnalyzerSettings />
    </Analyzer>
  </Analyzers>
</StyleCopSettings>
```

StyleCop Regeln deaktivieren

- StyleCop Regeln können mit dem Attribut `SuppressMessage` situativ deaktiviert werden
  - Details zum Attribut siehe MSDN
- Beispiel:

```
[SuppressMessage(
  "Microsoft.StyleCop.CSharp.DocumentationRules",
  "SA1600:ElementsMustBeDocumented",
  Justification = "No time to write documentation...")]
  public class Bucket<T>
  {
     …
  }
```

MSBuild Integration

Microsoft StyleCop 4.4.0.14 Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

Visual Studio Integration
MSBuild integration files

Files for setting up StyleCop integration with MSBuild.

This feature requires 918KB on your hard drive.

Reset

C:\Program Files (x86)\MSBuild\Microsoft\StyleCop\v4.4

Search v4.4

Organize ▾ Include in library ▾ Share with ▾ Burn New folder

Favorites
ADC2010 - Azure Intro
azuretests
DemoApps
Desktop
Downloads
Recent Places

Libraries
Documents

Name	Date modified	Type
Docs	08.12.2010 11:05	File folder
Microsoft.SourceAnalysis.Targets	20.05.2010 10:16	TARGETS File
Microsoft.StyleCop.CSharp.dll	20.07.2010 10:51	Application extens...
Microsoft.StyleCop.CSharp.Rules.dll	20.07.2010 10:51	Application extens...
Microsoft.StyleCop.dll	20.07.2010 10:51	Application extens...
Microsoft.StyleCo		TARGETS File
Settings.StyleCop		STYLECOP File
StyleCopSettingsE		Application

File description: Microsoft StyleCop C# Analyzers
Company: Microsoft Corporation
File version: 4.4.0.14
Date created: 20.07.2010 10:51
Size: 182 KB

StyleCop Build Integration

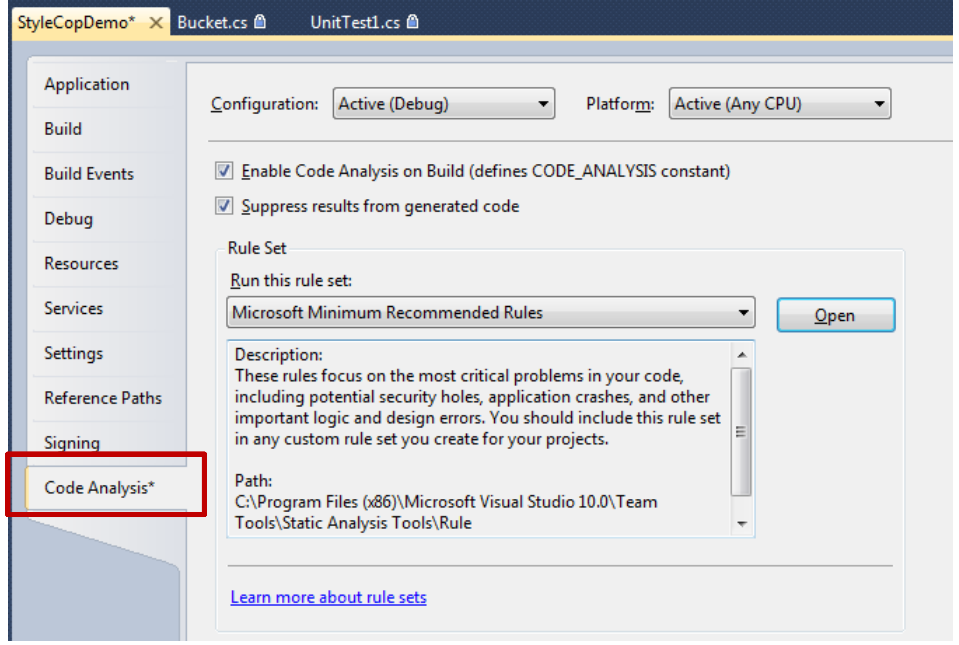
HANDS-ON LAB 2 (10 MINUTEN)

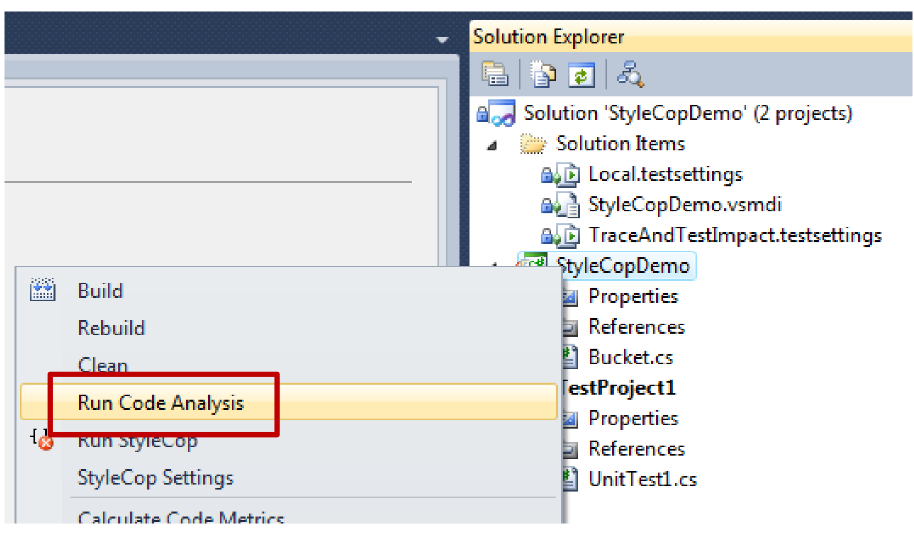
Früher FxCop, jetzt…

VISUAL STUDIO CODE ANALYSIS

Was leistet Code Analysis?

- Analysiert managed assemblies (nicht Sourcecode)
- Kontrolliert die Einhaltung der Framework Design Guidelines für .NET
  - Cwalina, Abrams: Framework Design Guidelines





Rule Sets

- Eingebaute Rule Sets
  - Siehe MSDN
- Können angepasst werden
  - Siehe MSDN
- Situative Unterdrückung von Code Analysis Warnings/Error wie bei StyleCop mit `SuppressMessage`
  - Beispiel:

```
[SuppressMessage("Microsoft.Design",
  "CA1039:ListsAreStrongTyped")]
Public class MyClass
{
    // code
}
```

The Power Of Code Analysis

HANDS-ON LAB 3 (20 MINUTEN)

BASTA! ON TOUR

The best of…

FRAMEWORK DESIGN GUIDELINES

Zentrale Fragestellungen

- Wie entwickelt man Komponenten, in denen andere Entwickler sich zu Hause fühlen?
- Wie sieht modernes Frameworkdesign aus?
- Welche Standards und Werkzeuge gibt es zur Unterstützung?

Wie weit muss/soll man gehen?

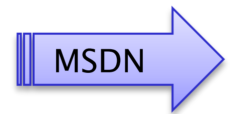
- Ist es meine Aufgabe, ein wiederverwendbares Framework zu erstellen?
- Welche Abhängigkeiten erzeuge ich durch die Make-or-buy Entscheidung?
- Gibt es diese Komponente bereits am Markt in einer für mich passenden Qualität?
- Bestehende Komponenten passen nicht 100%ig – ist es Zeit, die Anforderungen zu überdenken?
- U.v.m.

Tipps zur Komponentenauswahl

- Fertige Komponenten sind oft eine gute Wahl, wenn es sich nicht um eine differenzierenden Kernbereich handelt
  - Probleme, die viele Entwickler haben, sind in der Regel bereits gelöst
- Plattformen gegenüber Produkten vorziehen
- Marktposition des Anbieters berücksichtigen
- Aufwand zur Auswahl minimieren
  - KO Kriterien formulieren
  - Vergleichen statt endlosem Pflichtenheft

Wer die Wahl hat, hat die Qual!

KLASSEN ODER STRUKTUREN?



Klasse oder Struktur

- Strukturen in Betracht ziehen, wenn…
  - …der Typ klein ist UND
  - …Instanzen typischerweise kurzlebig sind UND
  - …meist eingebettet in andere Typen vorkommt.
- Strukturen nicht, wenn…
  - …der Typ logisch mehr als einen Wert repräsentiert ODER
  - …Größe einer Instanz >= 16 Bytes ODER
  - …Instanzen nicht immutable sind ODER
- Generell: Strukturen sind in C# sehr selten

Was ist falsch? (Code)

```
namespace Transporters.TubeNetwork
{
  public struct GeoPosition
  {
    public double Lat { get; set; }
    public double Long { get; set; }
  }
}
```

Value Types sollten wenn möglich immutable sein (wegen Kopie)

Beispiel für Struktur Teil 1 (Code)

```
namespace Transporters.TubeNetwork
{
   public struct GeoPosition
   {
   public GeoPosition(double lat, double lng)
   {
    this.Lat = lat;
    this.Long = lng;
   }

   public double Lat { get; private set; }
   public double Long { get; private set; }
}
```

Anmerkung:
Existiert bereits in Microsoft SQL Server System CLR Types
(SQL Server Feature Pack)

Beispiel für Struktur Teil 2 (Code)

```
public struct GeoPosition: IEquatable<GeoPosition>
{
     public GeoPosition(double lat, double lng)
          : this()
     {
          this.Lat = lat;
          this.Long = lng;
     }
     public double Lat { get; private set; }
     public double Long { get; private set; }

     public bool Equals(GeoPosition other)
     {
          return this.Lat == other.Lat
               && this.Long == other.Long;
     }

     public override bool Equals(object obj)
     {
          if (obj == null)
          {
               return base.Equals(obj);
          }

          if (!(obj is GeoPosition))
          {
               throw new InvalidCastException(
                    "Argument is not a GeoPosition obj.");
          }

          return this.Equals((GeoPosition)obj);
     }

     public override int GetHashCode()
     {
          unchecked
          {
             int hash = 17;
             hash = hash * 23 + this.Lat.GetHashCode();
             hash = hash * 23 + this.Long.GetHashCode();
             return hash;
          }
     }

     public static bool operator ==(GeoPosition x,
          GeoPosition y)
     {
          return x.Equals(y);
     }

     public static bool operator !=(GeoPosition x,
          GeoPosition y)
     {
          return !x.Equals(y);
     }
  }
}
```

Tipp: Strukturen sollten immer `IEquatable<T>` implementieren!

→Überschreiben von `Object.Equals` und `Object.GetHashCode`
→Operatoren `==` und `!=` überschreiben

Tipps für abstrakte und statische Klassen

- Abstrakte Klassen
  - `protected` oder `internal` Konstruktor
  - Zu jeder abstrakten Klasse mind. eine konkrete Implementierung
- Statische Klassen
  - Sollten die Ausnahme sein
  - Nicht als Misthaufen verwenden

Exkurs: Sealing

- C# Schlüsselwort `sealed`
- Kann angewandt werden auf
  - Klasse
  - Members
- Kein `sealed` bei Klassen außer es gibt gute Gründe
  - Grund könnten z.B. sicherheitsrelevante Eigenschaften in protected Members sein
- `sealed` macht oft Sinn bei überschriebenen Members

Designhilfsmittel

- Client-Code First
  - Test Driven Development
- Aufwand minimieren
  - Word & PowerPoint
  - Pseudocode
  - Scriptsprachen
- *„Simple things should be simple and complex things should be possible“* (Alan Kay, Turing-Preisträger)
- Feedback der späteren Anwender einholen

Auch Klassenbibliotheken haben eine Benutzerschnittstelle!

Programmierte Prototypen

This solution would replace `Install.stg` and `CodeBatchCollection`. There would be a Xaml file compiled into time cockpit's resources. There has to be a function to apply all update batches in the Xaml file similar to today's `InstallBatchManager.Install`. Additionally there will be functions to

1. find out all update batches that are missing on a certain database.
2. find out if the application can work with a certain database.

The following code snippets show how the API to install update batches would work:

Get update batch from XAML file stored in the assembly's resources.

```
UpdateBatch updateBatch = this.ReadUpdateBatchFromResources();
```

Note that DbClient.Create *will* ***not*** *automatically install update batches in the future any more.*

```
using (var dbClient = new DbClient.Create(...))
{
```

Find out which update batches are not installed in the database that dbClient *is pointing to.*

```
    IEnumerable<UpdateBatch> missingBatches =
        dbClient.GetMissingUpdateBatches(updateBatch);
    foreach (var missingBatch in missingBatches)
    {
        Console.WriteLine("{0} is missing", missingBatch.Guid);
    }
```

Find out if app can run without executing any batches (i.e. if mandatory batches are missing).

```
    switch (dbClient.GetUpdateBatchStatus(updateBatch))
    {
        case BatchStatus.Complete:
            Console.WriteLine("Update batch is completely installed.");
            break;
        case BatchStatus.Acceptable:
            Console.WriteLine("Some non-mandatory batches are missing.");
            break;
        case BatchStatus.Incomplete:
            Console.WriteLine("Mandatory batches are missing.");
            break;
    }
```

Install all missing update batches.

```
    dbClient.InstallMissingUpdateBatches(updateBatch);
}
```

Feature Files

On model level time cockpit will be extended by "features". A feature is a part of the logical data

2009-08-25 Planning Sprint 7 - TCQL Extensions.pptx - Microsoft PowerPoint

Must-have Requirement
Subqueries in the Select Clause

SUBQUERIES

```
From P In Project
Where P.Customer.CustomerName = "ABC"
Select New With
{
  .Project = P,
  .Hours = ( From T In P.Timesheets
             Where :Year(T.StartDate) = 2009
                   And P.Type = "XYZ"
             Select new With {
               .Hours = Sum(T.DurationInHours) } )
}

From P In Project
Where P.Customer.CustomerName = "ABC"
Select New With
{
  .Project = P,
  .Hours = Sum(P.Timesheets.DurationInHours)
}
```

Option – should we do that?

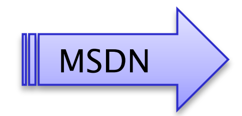
Folie 15 von 32 "FFG Zwischenpräsentation 2009-08-04" Deutsch (Österreich) 63 %

Beispiele

- Typische Szenarien von seltenen Szenarien durch Namespaces trennen (z.B. `System.Mail` und `System.Mail.Advanced`)
- Kurze Methodensignaturen für einfache Szenarien
  - Method overloading
  - Null als Defaultwert akzeptieren
  - C# 4: Named and Optional Arguments (siehe MSDN)
- Konstruktoren anbieten
  - Defaultkonstruktor wenn möglich/sinnvoll
  - Konstruktor mit wichtigsten Properties
- Einfache Szenarien sollten das Erstellen von wenigen Typen brauchen
- Keine langen Initialisierungen vor typischen Szenarien notwendig machen
- Sprechende Exceptions

Pleiten, Pech und Pannen oder…

TOP 10 TIPPS FÜR MEMBERDESIGN



Top 10 Tipps für Memberdesign

1. Kurze Methodensignaturen für einfache Szenarien
    - Method overloading
    - Null als Defaultwert akzeptieren
    - C# 4: Named and Optional Arguments (siehe MSDN)

C# 4: Optional Arguments (Code)

```
using System.ComponentModel;

namespace Transporters.TubeNetwork
{
    public class Station
    {
    public Station(string name = "")
    {
     this.Name = name;
    }

    public Station(string name, GeoPosition position)
     : this(name)
    {
     this.Position = position;
    }

    public string Name { get; set; }
    public GeoPosition Position { get; set; }
    }
}
```

C# 4: Optional Arguments (Code)

```
using System;

namespace OptionalParameters
{
   class Program
   {
    public static void DoSomething(int x = 17)
    {
        Console.WriteLine(x);
    }

    static void Main()
    {
        DoSomething();
    }
   }
}
```

- Versionierungsproblem
  Defaultwert in aufrufendem Code
- Nicht CLS Compliant
- Member overloading oft besser!

```
.method […] static void Main() cil managed
{
    .entrypoint
    .maxstack 8
    ldc.i4.s 0x11
    call void OptionalParameters.Program
        ::DoSomething(int32)
    ret
}
```

Top 10 Tipps für Memberdesign

1. Kurze Methodensignaturen für einfache Szenarien
   - Method overloading
   - Null als Defaultwert akzeptieren
   - C# 4: Named and Optional Arguments (siehe MSDN)
2. Methode statt Property wenn…
   - …Zeit zum Berechnen/Setzen des Wertes lang ist
   - …bei jedem Aufruf ein neuer Wert zurück gegeben wird (Negativbeispiel `DateTime.Now`)
   - …der Aufruf merkbare Seiteneffekte hat
   - …eine Kopie von internen Statusvariablen zurück gegeben wird

Top 10 Tipps für Memberdesign

3. Defaultwerte für Properties festlegen und klar kommunizieren
4. Ungültigen Status temporär akzeptieren
   - Properties können in beliebiger Reihenfolge gesetzt werden
5. Property Change Notification Events
   - Eventuell `INotifyPropertyChanged`
6. Konstruktoren anbieten
   - Defaultkonstruktor wenn möglich/sinnvoll
   - Konstruktor mit wichtigsten Properties

Top 10 Tipps für Memberdesign

7. Möglichst wenig Arbeit in Konstruktor
   - Auf keinen Fall virtuelle Methoden im Konstruktor aufrufen
8. Möglichst wenig bei Parametertypen voraussetzen
   - Z.B. `IEnumerable<T>` statt `List<T>`
9. Eingabeparameter immer prüfen
   - Eventuell `ArgumentException` (oder eine der Nachfahrenklassen)
10. Gute Namensgebung

Beispiel für Property Changed Notification (Code)

```
using System.ComponentModel;


namespace Transporters.TubeNetwork
{
  public class Station
  {
   public string Name { get; set; }
   public GeoPosition Position { get; set; }
  }
}
```

Notification fehlt.

Beispiel für Property Changed Notification (Code)

```
public class Station : INotifyPropertyChanged
{
    private string name;
    private GeoPosition position;

    public string Name
    {
    get { return this.name; }
    set
    {
        if (this.name != value) {
            this.name = value;
            this.OnPropertyChanged("Name");
        }
    }
    }

    public GeoPosition Position    {
        […]
    }

    public event PropertyChangedEventHandler PropertyChanged;

    private void OnPropertyChanged(string propertyName)
    {
    if (this.PropertyChanged != null) {
        this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
    }
}
```

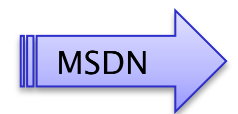
Mit Notification

Warum `INotifyPropertyChanged` und nicht Basisklasse (z.B. `NotifyingObject`)?

Standardmechanismus; ready for Data Binding ☺

Pest oder Cholera?

KLASSE ODER INTERFACE

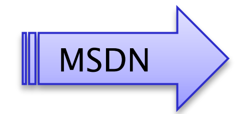


Basisklasse oder Interface?

- Generell Klassen Interfaces vorziehen
  - Aber ist das Interface nicht ein gutes Mittel zum Abbilden eines „Contracts“ zwischen Komponenten?
- Abstrakte Basisklasse statt Interface, um Contract und Implementation zu trennen
  - Interface ist nur Syntax, Klasse kann auch Verhalten abbilden
  - Beispiel: `DependencyObject` in WPF
- Tipp (Quelle: Jeffrey Richter)
  - Vererbung: „is a“
  - Implementierung eines Interface: „“can-do“

Genericssind Ihre Freunde!

COLLECTIONS



Was ist falsch (Code)?

```
using System.Collections;

namespace Transporters.TubeNetwork
{
  public class TrainNetwork
  {
  public ArrayList Stations
  { get; set; }
  }
}
```

Generics!
Sie sollten nichts anderes mehr verwenden!

Was ist falsch (Code)?

```
using System.Collections.Generic;


namespace Transporters.TubeNetwork
{
  public class TrainNetwork
  {
  public List<Station> Stations
   { get; set; }
  }
}
```

Set darf bei Collections nicht public sein!

Regeln für Collections (Teil 1)

- Keine „weakly typed“ Collections in öffentlichen APIs
  - Verwenden Sie stattdessen Generics
- List<T>, Hashtable, Dictionary<T> sollten in öffentlichen APIs nicht verwendet werden
  - Warum? Beispiel List<T>.BinarySort
- Collection Properties…
  - …dürfen nicht schreibbar sein
  - Read/Write Collection Properties: Collection<T>
  - Read-Only Collection Properties: ReadOnlyCollection<T> oder IEnumerable<T>
- Eigene thread-safe Collections für parallele Programmierung

Was ist falsch (Code)?

```
using System.Collections;
using System.Collections.Generic;

namespace Transporters.TubeNetwork
{
  public class TrainNetwork
  {
  private List<Station> stations
   = new List<Station>();
  public IEnumerable Stations {
   get { return this.stations; } }
  }
}
```

Keine weakly typed collections.

Wie soll `Stations` befüllt werden?

Beispiel Read/Write Collection (Code)

```
using System.Collections.ObjectModel;

namespace Transporters.TubeNetwork
{
    public class StationCollection : Collection<Station>
    {
    }
}
```

```
namespace Transporters.TubeNetwork
{
    public class TrainNetwork
    {
     public TrainNetwork()
     {
         this.Stations = new StationCollection();
     }

     public StationCollection Stations { get; private set; }
    }
}
```

Regeln für Collections (Teil 2)

- *„Require the weakest thing you need, return the stronges thing you have“*
(A. Moore, Development Lead, Base Class Libarary of the CLR 2001-2007)
- `KeyCollection<TKey, TItem>` nützlich für Collections mit primary Keys
- Collection oder Array?
  - Generell eher Collection statt Array (Ausnahme sind Dinge wie `byte[]`)
  - `Collection`- bzw. `Dictionary`-Postfix bei eigenen Collections

Fallbeispiel `IList` vs. `IEnumerable`

- Methodensignatur von `Parallel.ForEach`:

```
public static ParallelLoopResult ForEach<TSource>(
      IEnumerable<TSource> source,
      Action<TSource> body
   )
```

- *“Require the weakest thing you need…“*
  - `IEnumerable<TSource>` wird akzeptiert
  - Intern unterschiedlich implementiert für `IEnumerable<T>` und `IList<T>`
- „…*return the stronges thing you have*“
  - `Parallel.ForEach` ist schneller mit `IList<T>` als mit `IEnumerable<T>`
  - Framework sollte `IList<T>` zurückgeben, wenn intern eine Implementierung dieses Typs vorliegt

Was ist falsch (Code)?

```
public class StationStopCollection : Collection<StationStop>, INotifyCollectionChanged
{
    public event NotifyCollectionChangedEventHandler CollectionChanged;

    protected override void SetItem(int index, StationStop item)
    {
    base.SetItem(index, item);
    this.OnCollectionChanged(NotifyCollectionChangedAction.Replace);
    }

    protected override void ClearItems()
    {
    base.ClearItems();
    this.OnCollectionChanged(NotifyCollectionChangedAction.Remove);
    }
    […]

    private void OnCollectionChanged(NotifyCollectionChangedAction action)
    {
    if (this.CollectionChanged != null)
     {
      this.CollectionChanged(this, new NotifyCollectionChangedEventArgs(action));
     }
    }
}
```

Gibt es schon!
`ObservableCollection<T>`

Was ist falsch (Code)?

```
public IEnumerable<DateTime> GetCalendar(
  DateTime fromDate, DateTime toDate)
{
  var result = new List<DateTime>();
  for (; fromDate <= toDate;
     fromDate = fromDate.AddDays(1))
  {
  result.Add(fromDate);
  }

  return result;
}
```

Dafür gibt es `yield` Blocks!

Beispiel `yield` Block (Code)

```
public IEnumerable<DateTime>
   GetCalendar(DateTime fromDate, DateTime
   toDate)
{
   for (; fromDate <= toDate;
      fromDate = fromDate.AddDays(1))
   {
   yield return fromDate;
   }

   yield break;
}
```

Regeln für `yield` Blocks

- Rückgabewert muss `IEnumerable` sein
- Keine `ref` oder `out` parameter
- `yield` nicht erlaubt in `try-catch` (jedoch schon in `try-finally`)

Exkurs: Enums

- Enums statt statischer Konstanten
  - Einschränkung: Wertebereich muss bekannt sein
- Don‘ts bei Enums
  - „ReservedForFutureUse“ Einträge
  - Enums mit genau einem Wert
  - „LastValue“ Eintrag am Ende
- Flag-Enums
  - `[Flags]` Attribut nicht vergessen
  - 2erpotenzen als Werte verwenden (wegen OR-Verknüpfung)
  - Spezielle Werte für häufige Kombinationen einführen
  - Kein Wert 0 (sinnlos bei OR-Verknüpfung)

Beispiel Enum (Code)

```
using System;

namespace Transporters.TubeNetwork
{
   [Flags]
   public enum DayType
   {
   Workingdays = 1,
   WorkingdaysDuringHolidays = 2,
   Saturday = 4,
   Sunday = 8,
   Christmas = 16,
   Always = Workingdays | WorkingdaysDuringHolidays
    | Saturday | Sunday | Christmas,
   Weekend = Saturday | Sunday
   }
}
```

Was ist falsch (Code)?

```
if ((fromDate.GetDateType() & DayType.Weekend)
   == DayType.Weekend)
{
   [...]
}
```

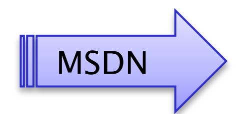
Kann nie `true` sein!

Beispiel Field Enum (Code)

```
if ((fromDate.GetDateType()&DayType.Weekend)!=0)
{
   […]
}
```

Exkurs: Extension Methods

- Sparsam damit umgehen!
  - Können das API-Design zerstören
- Verwenden, wenn Methode relevant für alle Instanzen eines Typs
- Können verwendet werden, um Abhängigkeiten zu entfernen
- Können verwendet werden, um Methoden zu Interfaces hinzuzufügen
  - Immer die Frage stellen: Wäre eine Basisklasse besser?



Beispiel Extension Methods (Code)

Unterschied `const/readonly` beachten!

```
public static class DateTimeType
{
    private const int ChristmasDay = 25;
    private const int ChristmasMonth = 12;
    private static readonly Tuple<DateTime, DateTime>[] Holidays = new[]
    {
     […]
    };
    public static DayType GetDateType(this DateTime day)
    {
     if (day.Month == DateTimeType.ChristmasMonth
                && day.Day == DateTimeType.ChristmasDay) {
      return DayType.Christmas;
     }
     else if (day.DayOfWeek == DayOfWeek.Saturday) {
      return DayType.Saturday;
     }
     else if (day.DayOfWeek == DayOfWeek.Sunday) {
      return DayType.Sunday;
     }
     else {
      […]
     }
    }
}
```

C# 3: Array Initializer, implizit Typing

Erweiterbare Frameworks

- Klassen nicht mit `sealed` anlegen
- Events vorsehen
  - `Func<...>`, `Action<…>` oder `Expression<…>` anstelle von Delegaten
- Virtuelle Members
  - `virtual` nur, wo Erweiterbarkeit explizit gewünscht ist

Was läuft hier falsch? (Code)

```
public IEnumerable<DateTime> GetWeekendsCalendar(DateTime fromDate, DateTime toDate)
{
    for (; fromDate <= toDate; fromDate = fromDate.AddDays(1)) {
     if ((fromDate.GetDateType() & DayType.Weekend) != 0) {
      yield return fromDate;
     }
    }
}

public IEnumerable<DateTime> GetWorkingDaysCalendar(DateTime fromDate, DateTime toDate)
{
    for (; fromDate <= toDate; fromDate = fromDate.AddDays(1)) {
     if ((fromDate.GetDateType() & DayType.Workingdays) != 0) {
      yield return fromDate;
     }
    }
}

public IEnumerable<DateTime> GetCalendar(DateTime fromDate, DateTime toDate)
{
    for (; fromDate <= toDate; fromDate = fromDate.AddDays(1)) {
     yield return fromDate;
    }
}
```

Duplizierter Code!

Beispiel Funktionsparameter (Code)

```
public IEnumerable<DateTime> GetWeekendsCalendar(DateTime fromDate, DateTime toDate)
{
    return this.GetCalendar(fromDate, toDate, d => (d.GetDateType() &
    DayType.Weekend)!=0);
}

public IEnumerable<DateTime> GetWorkingDaysCalendar(DateTime fromDate,
     DateTime toDate)
{
    return this.GetCalendar(fromDate, toDate, d
    =>(d.GetDateType()&DayType.Workingdays)!=0);
}

public IEnumerable<DateTime> GetCalendar(DateTime fromDate, DateTime toDate)
{
    return this.GetCalendar(fromDate, toDate, d => true);
}

private IEnumerable<DateTime> GetCalendar(DateTime fromDate, DateTime toDate,
     Func<DateTime, bool> filter)
{
    for (; fromDate <= toDate; fromDate = fromDate.AddDays(1)) {
    if (filter(fromDate)) {
     yield return fromDate;
    }
    }
}
```

Warum Dinge neu erfinden?
LINQ kann das sowieso!

Beispiel Funktionsparameter (Code)

```
public IEnumerable<DateTime> GetWeekendsCalendar(DateTime fromDate, DateTime toDate)
{
    return this.GetCalendar(fromDate, toDate)
     .Where(d => (d.GetDateType() & DayType.Weekend) != 0);
}

public IEnumerable<DateTime> GetWorkingDaysCalendar(DateTime fromDate, DateTime
     toDate)
{
    return this.GetCalendar(fromDate, toDate)
     .Where(d => (d.GetDateType() & DayType.Workingdays) != 0);
}

public IEnumerable<DateTime> GetCalendar(DateTime fromDate, DateTime toDate)
{
    for (; fromDate <= toDate; fromDate = fromDate.AddDays(1))
    {
     yield return fromDate;
    }
}
```

Eleganteste Lösung mit LINQ!

Sind die ersten beiden Methoden überhaupt notwendig?

Exceptions

- Exceptions statt error codes!
- `System.Environment.FailFast` in Situationen, bei denen es unsicher wäre, weiter auszuführen
- Exceptions nicht zur normalen Ablaufsteuerung verwenden
- Eigene Exceptionklassen erstellen, wenn auf den Exceptiontyp auf besondere Weise reagiert werden soll
- `finally` für Cleanup, nicht `catch`!
- Standard Exceptiontypen richtig verwenden
- Möglicherweise `Try…` Pattern verwenden (z.B. `DateTime.TryParse`)

Beispiel Exception (Code)

```
using System;
using System.Runtime.Serialization;

namespace Transporters.TubeNetwork
{
    [Serializable]
    public class NetworkInconsistencyException : Exception, ISerializable
    {
    public NetworkInconsistencyException()
     : base() { }

    public NetworkInconsistencyException(string message)
     : base(message)
     {
     }

    public NetworkInconsistencyException(string message, Exception inner)
     : base(message, inner)
     {
     }

    public NetworkInconsistencyException(SerializationInfo info, StreamingContext context)
     : base(info, context)
     {
     }
    }
}
```

Was läuft hier falsch? (Code)

```
public void AddTravel(IEnumerable<Station> stations, T line,
   DayType dayType, string routeFileName,
    TimeSpan leavingOfFirstTrain, TimeSpan leavingOfLastTrain,
    TimeSpan interval)
{
   var stream = new StreamReader(routeFileName);
   var route = XamlServices.Load(stream) as IEnumerable<Hop>;

   this.AddTravel(stations, line, dayType, route,
   leavingOfFirstTrain, leavingOfLastTrain, interval);

   stream.Close();
}
```

Disposable Pattern. Wichtig im Fall einer Exception!

Was läuft hier falsch? (Code)

```
public void AddTravel(IEnumerable<Station> stations, T line,
   DayType dayType, string routeFileName,
    TimeSpan leavingOfFirstTrain,
   TimeSpan leavingOfLastTrain, TimeSpan interval)
{
   using (var stream = new StreamReader(routeFileName))
   {
    var route = XamlServices.Load(stream) as IEnumerable<Hop>;

    this.AddTravel(stations, line, dayType, route,
        leavingOfFirstTrain, leavingOfLastTrain, interval);
    stream.Close();
   }
}
```