



BASTA!
NET, WINDOWS, VISUAL STUDIO

Besseres C#

Workshop

BASTA! Spring 2014

Besseres C#

Workshop



Rainer Stropek

software architects gmbh

Web

<http://www.timecockpit.com>

Mail

rainer@timecockpit.com

Twitter

@rstropek



time cockpit
Saves the day.

Die Sprache ist egal, das Framework
macht den Unterschied – richtig?

Was zeichnet guten C# Code aus?

Softfacts rund um guten C# Code

► Quality Culture

Weniger LOC/Tag heißt nicht weniger produktiv
Testability, Unit Tests

► Code Reviews

QA, Lerneffekt

► Coding und Design Guidelines

Best und worst practices
Wo möglich Toolunterstützung und automatische Prüfung

► Die richtigen Werkzeuge einsetzen

Z.B. Profiler, statische Codeanalyse

► Done Done Checkliste definieren

Persönlich und im Team

► Aus Fehlern lernen

Individuell, im Team (TFS)

Die wichtigsten fünf Gebote für Klassenbibliotheken

- ▶ **Je häufiger wiederverwendet desto höher muss die Qualität sein**

An der zentralen Klassenbibliothek arbeiten Ihre besten Leute

Design, Code und Security Reviews

Dokumentation

- ▶ **Folgen Sie den Microsoft Design Guidelines und Best Practices**

Nutzen Sie StyleCop und Code Analysis oder ähnliche Tools von Drittanbietern

Wissen im Team steigern

- ▶ **Schreiben Sie Unit- und Integrationstests**

Gleiche Qualitätskriterien wie beim Framework selbst

Qualität der Tests hängt zu großem Teil von der Qualität der *Asserts* ab

Monitoring der *Code Coverage*

Die wichtigsten fünf Gebote für Klassenbibliotheken

- ▶ Steigern Sie die Produktivität durch Automatisierung und Wiederverwendung

Visual Studio Templates, T4 Templates

Gutes Versions- und Paketmanagement für zentrale Klassenbibliotheken

- ▶ Verwenden Sie Scenario Driven Design

Siehe folgende Slides

Tipps für Frameworkdesign

- ▶ Beste Erfahrungen mit Scenario-Driven Design

Client-Code-First (hilft auch für TDD ☺)

Welche Programmiersprachen sind dabei für Sie interessant? Dynamische Sprachen nicht vergessen!

- ▶ „*Simple things should be simple and complex things should be possible*“
(Alan Kay, Turing-Preisträger)

- ▶ Einfache Szenarien sollten ohne Hilfe umsetzbar sein

Beispiele:

- Typische Szenarien von komplexen Szenarien mit Namespaces trennen

- Einfache Methodensignaturen bieten (Defaultwerte!)

- Einfache Szenarien sollten das Erstellen von wenigen Typen brauchen

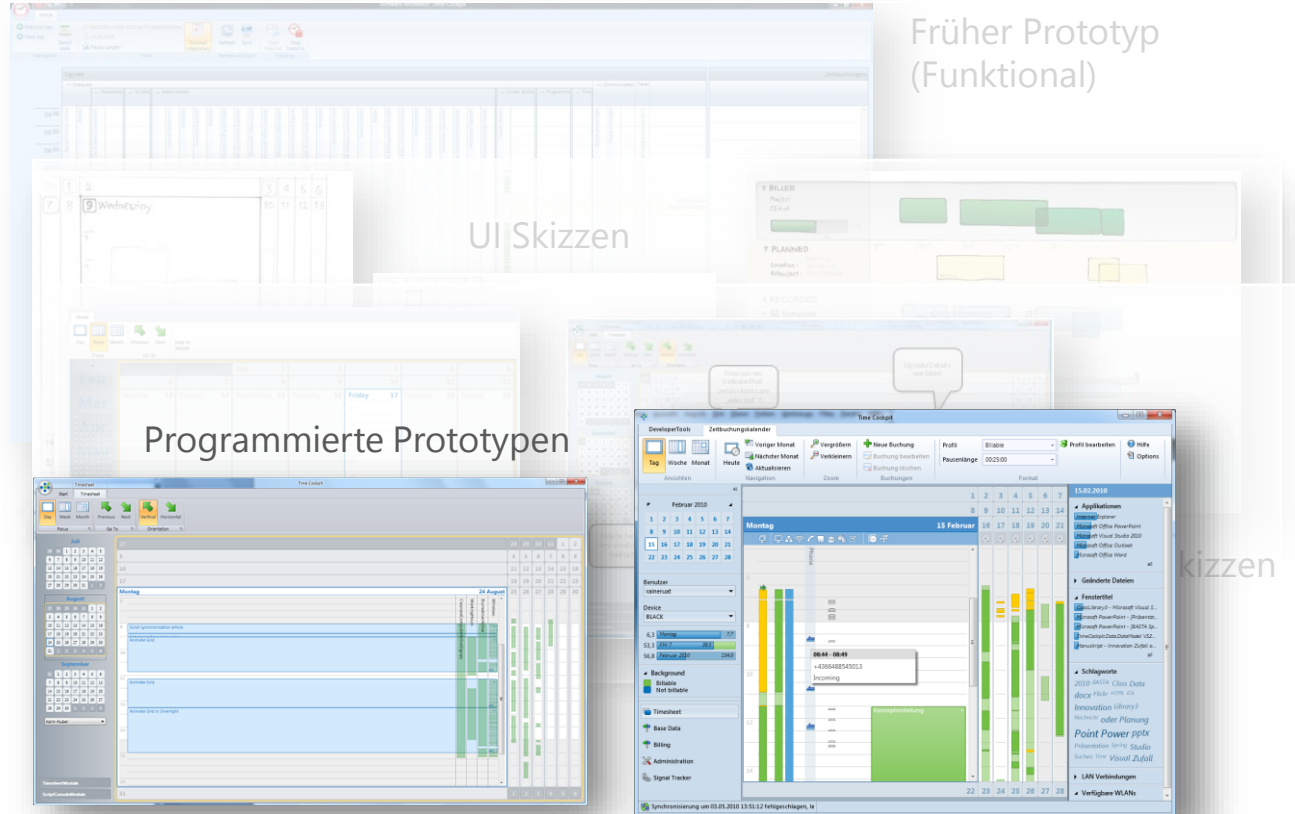
- Keine langen Initialisierungen vor typischen Szenarien notwendig machen

- Sprechende Exceptions

Bibliographie Framework Design Guidelines

- ▶ Cwalina, Abrams: Framework Design Guidelines
Sporadische Aktualisierungen im Blog von Cwalina
Abrams ist nicht mehr bei MS (früherer Blog)
- ▶ Auszug aus dem Buch kostenlos in der MSDN verfügbar
Design Guidelines for Developing Class Libraries
- ▶ Scenario Driven Design
Blogartikel Cwalina

SDD im UI – offensichtlich



SDD bei APIs - unterschätzt

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

OData also knows relations between entities. They correspond to relations in [CoFX's](#) data model but are referred to as *links* or *navigational properties*. When querying objects, relations are not included by default but can be expanded using the [\\$expand uri parameter](#). Here is an example:

REQUEST:

```
GET https://api.pros-rec-mgr.com/odata/APP_Customer?$expand=APP_Country
```

CORRESPONDING TCQL QUERY:

```
From C In APP_Customer.Include('APP_Country')  
Select C
```

RESPONSE:

```
{  
  "odata.metadata":  
    "https://api.pros-rec-mgr.com/odata/$metadata#APP_Customer",  
  "value": [  
    {  
      "APP_Country": {  
        "APP_CountryUuid": "950fb883-fe47-46e6-acf0-a934a65967c6",  
        "APP_Code": "DE" },  
      "APP_CustomerUuid": "4c75bb13-8d9a-4bf6-a0de-fdcdb69a80c",  
      "APP_Code": "Customer1",  
      "APP_Description": "This is customer 1", ...  
    }  
  ]  
}
```

```

using Samples.Sudoku;
var board = new Board();

var boardData = SomeSudokuGenerator.Generate(...);
    // Generator out of scope of this specification

try
{
    for (var rowIndex = 0; rowIndex < 9; rowIndex++) {
        for (var columnIndex = 0; columnIndex < 9; columnIndex++) {
            if (... /* value for cell in boardData has a value) {
                // Method for setting a cell
                board.SetCell(rowIndex, columnIndex,
                    ... /* value from boardData*/);

                // Preferred alternative: Indexer
                Board[rowIndex][columnIndex] =
                    ... /* value from boardData */;
            }
        }
    }
}
catch (BoardException ...)
{
    ... // Generally boardData is ok; handle internal error here
}

```

Demo

Sudoku Board - Szenarien

Board befüllen

Methode

Indexer

Diskussion

Passen die Namen?

Selbsterklärend?

```
var board = new Board();  
...  
  
var rowIndex = ... /* User input */  
var colIndex = ... /* User input */  
var value = ... /* User input */  
...  
  
if (value == 0 /* User wants to remove value from cell */)   
{  
    board.ResetCell(rowIndex, colIndex);  
}  
else  
{  
    if (!board.TrySetCell(rowIndex, colIndex, value))  
    {  
        ... /* let user know that input was wrong */  
    }  
}
```

Demo

Sudoku Board - Szenarien

Board befüllen (Benutzereingaben)

Exception nicht passend

TryXXX pattern

Diskussion

Passend für XAML+DataBinding?

```
byte[] content = ... /* read sudoku content from e.g. file */  
  
var board = (Board)content; // Create by casting from byte[]  
  
// Now we can do something with the board  
...  
  
content = (byte[])board; // Get byte[] via casting  
  
... /* write sudoku board to e.g. file */
```

Demo

Sudoku Board - Szenarien

Board speichern

Laden

Speichern

Diskussion

Reader/Writer in Scope?

```
var board = new Board();

// Iterate over board content
foreach(var row in board.Rows)
{
    foreach(var cell in row)
    {
        Console.Write(cell); // prints cell value
    }

    Console.WriteLine();
}

// Direct access to board content
Console.WriteLine(board[0][5]);

// Note: Make access to board threadsafe. There might
//       be multiple threads reading from the board at the
//       same time (e.g. for solver).
```

Demo

Sudoku Board - Szenarien

Board abfragen

Iteration

Direkter Zugriff auf Zellen

Async und Parallel Aspekte nicht vergessen

Threadsafe?

Async?

```
// Loading a board - Simple scenario
var board = await BoardReader.LoadFromFileAsync(
    "SampleBoard" [, directoryName]);
var board = await BoardReader.LoadFromBlobStorageAsync(
    accountName, accountKey, containerName, "Sample Board");
```

```
// Make the API flexible so that we can add additional
// storage types later
public class MyBoardStorage : IStreamManager
{
    public Task<Stream> OpenStreamAsync(
        string boardName, AccessMode accessMode) { ... }
}
```

```
var repository = new BoardStreamRepository(
    new MyBoardStorage(...);
await repository.LoadAsync("Sample Board");
await repository.SaveAsync("Sample Board");
```

Demo

Sudoku Board - Szenarien

Board laden und speichern

Motto: The simple things should
be simple, the complex things
should be possible

Sudoku Board

Demo

Framework design

Part 1

Namensschema

Manchmal lästig, aber sehr wichtig!



Generelle Namensregeln

- ▶ Diskussionen darüber im Team? Warum nicht einfach die Regeln von Microsoft übernehmen?

Toolunterstützung durch StyleCop

- ▶ 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*

Exkurs: *public*, *private*, *protected*, *internal* und *protected internal*

- ▶ 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
Code Analysis warnt
- ▶ Typische Szenarien von seltenen Szenarien durch Namespaces trennen (z.B. *System.Mail* und *System.Mail.Advanced*)
- ▶ PascalCasingWithDotsRecommended

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

Klassen oder Strukturen?

Wer die Wahl hat, hat die Qual!



Klasse oder Struktur

- ▶ Strukturen in Betracht ziehen, wenn...
 - ...der Typ klein ist UND
 - ...Instanzen typischerweise kurzlebig sind UND
 - ...meist eingebettet in andere Typen vorkommt UND
 - ...*immutable*
- ▶ 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

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

Was ist falsch?

Value Types sollten wenn
möglich *immutable* sein

Private Setter

Konstruktorparameter

Code Contracts

```

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);
    }
}

```

Beispiel für Struktur

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

Überschreiben von
Object.Equals und
Object.GetHashCode

```

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);
}
}

```

Beispiel für Struktur

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

Top 10 Tipps für Memberdesign

Pleiten, Pech und Pannen oder...



Top 10 Tipps für Memberdesign

► Kurze Methodensignaturen für einfache Szenarien

Method overloading

Null als Defaultwert akzeptieren

C# 4: Named and Optional Arguments (siehe [MSDN](#))

```
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; }
    }
}
```

Optional Arguments

```
using System;
```

```
namespace OptionalParameters
```

```
{
```

```
    class Program
```

```
    {
```

```
        public static void DoSomething(int x = 17)
```

```
        {
```

```
            Console.WriteLine(x);
```

```
        }
```

```
        static void Main()
```

```
        {
```

```
            DoSomething();
```

```
        }
```

```
    }
```

```
}
```



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

Optional Arguments

Versionierungsproblem

Defaultwert in aufrufendem Code

Nicht CLS Compliant

Member overloading oft
besser

Top 10 Tipps für Memberdesign

► Kurze Methodensignaturen für einfache Szenarien

Method overloading

Null als Defaultwert akzeptieren

C# 4: Named and Optional Arguments (siehe [MSDN](#))

► 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

- ▶ Defaultwerte für Properties festlegen und klar kommunizieren
- ▶ Ungültigen Status temporär akzeptieren
Properties können in beliebiger Reihenfolge gesetzt werden
- ▶ Property Change Notification Events
Z.B. *INotifyPropertyChanged*
- ▶ Konstruktoren anbieten
Defaultkonstruktor wenn möglich/sinnvoll
Konstruktor mit wichtigsten Properties

Top 10 Tipps für Memberdesign

- ▶ **Möglichst wenig Arbeit in Konstruktor**
Auf keinen Fall virtuelle Methoden im Konstruktor aufrufen
- ▶ **Möglichst wenig bei Parametertypen voraussetzen**
Z.B. *IEnumerable<T>* statt *List<T>*
- ▶ **Eingabeparameter immer prüfen**
Z.B. *ArgumentException* oder eine der Nachfahrenklassen
- ▶ **Gute Namensgebung**

```
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));
        }
    }
}
```

Changed Notification

Alternative: MVVM Frameworks Z.B. Prism Core

Klasse oder Interface

Pest oder Cholera?



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“

Erweiterbare Frameworks

- ▶ Klassen nicht mit *sealed* anlegen
- ▶ Events und Callbacks vorsehen
Func<...>, *Action<...>* oder *Expression<...>* anstelle von Delegaten
Testability verbessern durch Ermöglichen von Mocking
- ▶ Virtuelle Members
virtual nur, wo Erweiterbarkeit explizit gewünscht ist
- ▶ Modularisierung
MEF, NuGet – Details später

Collections

Generics sind Ihre Freunde!

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

```
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; }  
    }  
}
```

Beispiel

Regeln für Collections (Teil 2)

- ▶ „Require the weakest thing you need, return the strongest thing you have“
(A. Moore, Development Lead, Base Class Library 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
- ▶ *ObservableCollection<T>* für Data Binding

```
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;  
}
```

Was ist falsch?

Dafür gibt es *yield* blocks

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

Beispiel yield Block

Regeln für `yield` Blocks

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

Collections für parallele Programm

- *System.Collections.Concurrent* für Thread-Safe Collections

BlockingCollection<T>

Blocking und Bounding-Funktionen

ConcurrentDictionary<T>

ConcurrentQueue<T>

ConcurrentStack<T>

ConcurrentBag<T>

- Optimal zur Umsetzung von Pipelines

Datei wird gelesen, gepackt, verschlüsselt, geschrieben

- *TPL Data Flow Library* als Option

```
var buffer = new BlockingCollection<long>(10);
var cancellationTokenSource = new CancellationTokenSource();

var producer = Task.Factory.StartNew((cancellationTokenObj) => {
    var counter = 100000000;
    var cancellationToken = (CancellationToken)cancellationTokenObj;
    try {
        while (!cancellationToken.IsCancellationRequested && counter-- > 0) {
            // Here we get some data (e.g. reading it from a file)
            var value = DateTime.Now.Ticks;
            // Write it to the buffer with values that have to be processed
            buffer.Add(value);
        }
    }
    finally {
        buffer.CompleteAdding();
    }
}, cancellationTokenSource.Token);
```

Producer/Consumer

```
var consumer = Task.Factory.StartNew((cancelTokenObj) =>
{
    var cancelToken = (CancellationToken)cancelTokenObj;
    foreach (var value in buffer.GetConsumingEnumerable())
    {
        if ( cancelToken.IsCancellationRequested )
        {
            break;
        }

        // Here we do some expensive procesing
        Thread.SpinWait(1000);
    }
}, cancelTokenSource.Token);
```

Producer/Consumer

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

Zerpotenzen als Werte verwenden (wegen OR-Verknüpfung)

Spezielle Werte für häufige Kombinationen einführen

Kein Wert 0 (sinnlos bei OR-Verknüpfung)



```
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
```

```
    }
```

```
}
```

Beispiel Enum

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



Kann nie true sein!

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



Was ist falsch?

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?




```
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 {
        [...]
    }
}
}
```

Extension Methods

Beispiel

Exceptions

So nicht!



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*)

```
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)
```

```
        {
```

```
        }
```

```
    }
```

```
}
```

Beispiel Exception

Code Analysis warnt bei
falscher Implementierung

```

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();
}

```

```

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();
    }
}

```

Was ist falsch?

IDisposable

Code Analysis warnt bei
falscher Implementierung

BASTA! Spring 2014

Q&A

Thank your for coming!



Rainer Stropek

software architects gmbh

Mail
Web
Twitter

rainer@timecockpit.com
<http://www.timecockpit.com>
@rstropek



time cockpit
Saves the day.



time cockpit is the leading time tracking solution for knowledge workers. Graphical time tracking calendar, automatic tracking of your work using signal trackers, high level of extensibility and customizability, full support to work offline, and SaaS deployment model make it the optimal choice especially in the IT consulting business.

Try **time cockpit** for free and without any risk. You can get your trial account at <http://www.timecockpit.com>. After the trial period you can use **time cockpit** for only 0,20€ per user and day without a minimal subscription time and without a minimal number of users.



time cockpit ist die führende Projektzeiterfassung für Knowledge Worker. Grafischer Zeitbuchungskalender, automatische Tätigkeitsaufzeichnung über Signal Tracker, umfassende Erweiterbarkeit und Anpassbarkeit, volle Offlinefähigkeit und einfachste Verwendung durch SaaS machen es zur Optimalen Lösung auch speziell im IT-Umfeld.

Probieren Sie **time cockpit** kostenlos und ohne Risiko einfach aus. Einen Testzugang erhalten Sie unter <http://www.timecockpit.com>. Danach nutzen Sie **time cockpit** um nur 0,20€ pro Benutzer und Tag ohne Minstdauer und ohne Mindestbenutzeranzahl.