BASTA!

.NET, WINDOWS, VISUAL STUDIO

Besseres C#

Workshop

BASTA! Spring 2014

Modularisierung

Workshop

Rainer Stropek

software architects gmbh

Web http://www.timecockpit.com
Mail rainer@timecockpit.com
Twitter @rstropek

time cockpit
Saves the day.

Modularisierung

Grundlagen

Warum modulare Programmierung?

The benefits expected of modular programming are:

- **Managerial** - development time should be shortened because separate groups would work on each module with little need for communication
- **Product flexibility** - it should be possible to make drastic changes to one module without a need to change others
- **Comprehensibility** - it should be possible to study the system one module at a time. The whole system can therefore be better designed because it is better understood.

Quelle: Parnas, On the Criteria To Be Used in Decomposing Systems into Modules, Carnegie-Mellon University, 1972

Was ist eine Softwarefabrik?

A software factory is an organizational structure that specializes in producing computer software applications or software components [...] **through an assembly process**.

The software is created by **assembling predefined components**. Traditional coding, is left only for creating new components or services.

A **composite application** is the end result of manufacturing in a software factory.

Quelle: "Software Factory" in Wikipedia

Vorteile einer Softwarefabrik (1/2)

- **Kernkompetenzen der Mitarbeiter werden hervorgehoben**
  Fachlich orientierte Berater konfigurieren (Vokabular näher bei der jeweiligen Domäne des Kunden)
  Softwareentwickler erstellen Basiskomponenten
- **Steigerung der Effizienz**
  Das Rad wird weniger oft neu erfunden
- **Steigerung der Qualität**
  Anspruchsvolle QS-Maßnahmen für Basiskomponenten

Vorteile einer Softwarefabrik (2/2)

- **Reduktion der Projektrisiken**
  Fachberater mit besserem Kundenverständnis
  Höhere Qualität der Basiskomponenten
- **Steigerung des Firmenwerts**
  Systematisches Festhalten von Wissen über die Herstellung einer Familie von Softwarelösungen
  Design Patterns, Frameworks, Modelle, DSLs, Tools
- **Vereinfachung des Vertriebs- und Spezifikationsprozesses**
  Konzentration auf projektspezifische Lösungsteile
  Quick Wins durch Standardkomponenten

Was eine Softwarefabrik nicht will…

- Reduktion des Entwicklungsprozesses auf standardisierte, mechanische Prozesse
  Im Gegenteil, mechanische Prozesse sollen Tool überlassen werden
- Verringerung der Bedeutung von Menschen im Entwicklungsprozess
- „Handwerk“ der Softwareentwicklung wird nicht gering geschätzt sondern gezielt eingesetzt
- Entwicklung von Frameworks statt Lösungen für Kunden

Software Factories → Economy of Scope

Economy of Scale

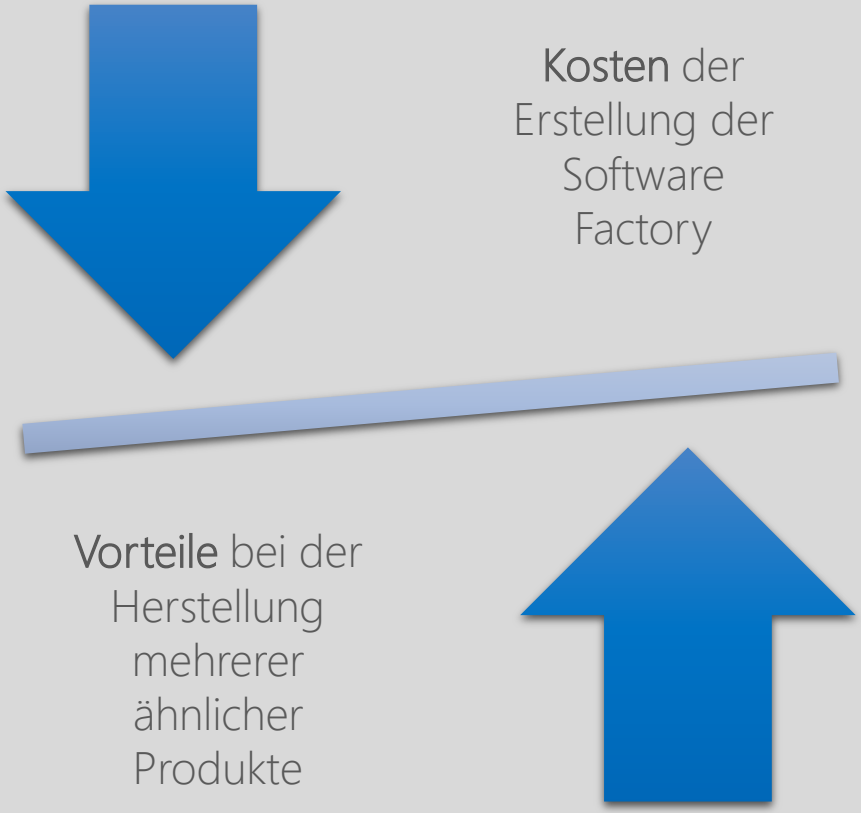
- Multiple implementations (=Copies) of the same design
- Mehr oder weniger mechanische Vervielfältigung von Prototypen
  Massengüter
  Software (z.B. Auslieferung auf Datenträger)

Economy of Scope

- Production of multiple designs and their initial implementations
- Ähnliche Designs (=Familien von Anwendungen) auf Grundlage gemeinsamer Techniken und Technologien
  Individuelle physische Güter (z.B. Brücken, Hochhäuser)
  Individualsoftware, Softwareplattformen (vgl. PaaS)

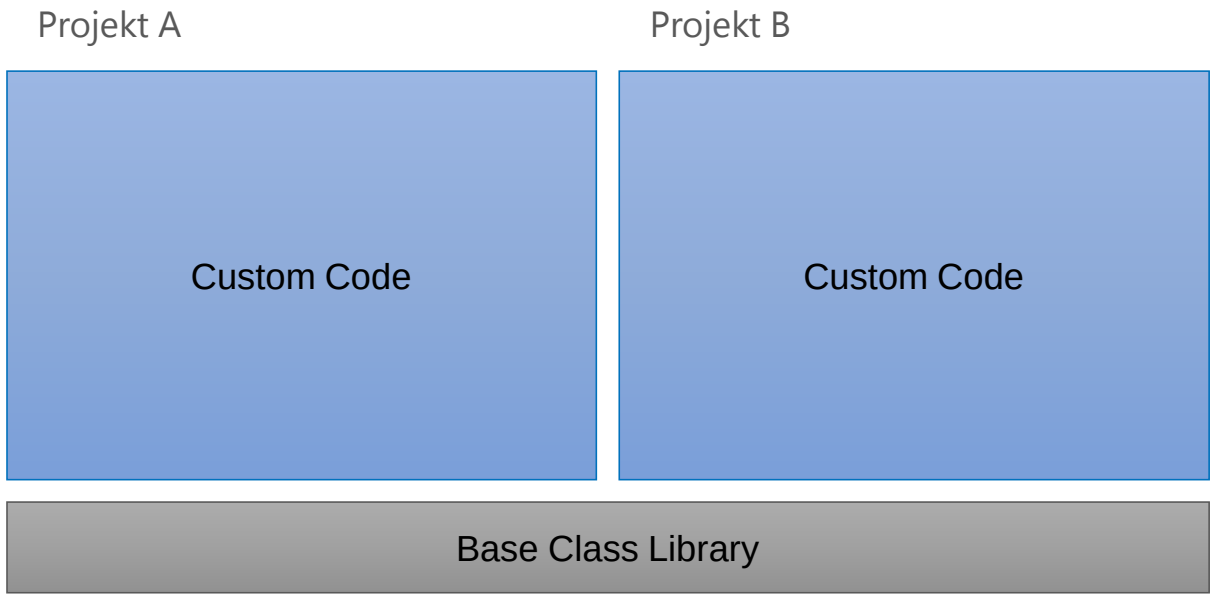
Quelle: Greenfield J., Short K.: Software Factories

Kosten/Nutzen Abwägung

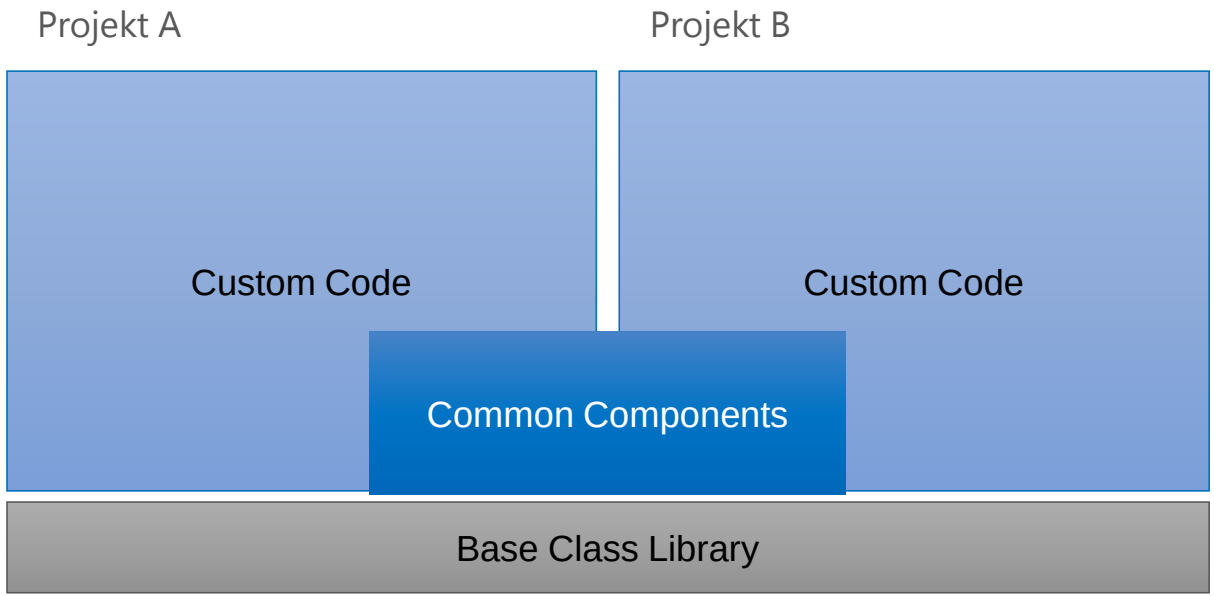


Spezialisierung als zentraler Erfolgsfaktor beim Einsatz der Software Factory Prinzipien

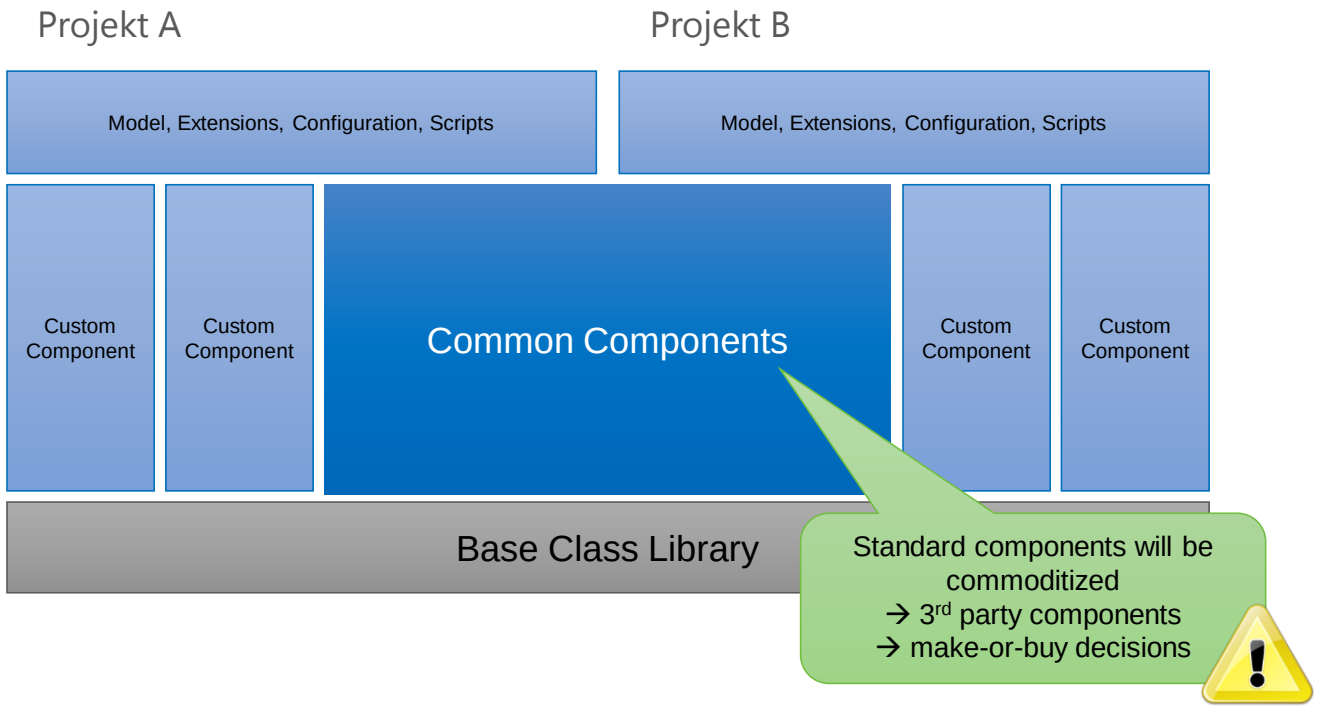
Entwicklung einer Software Factory



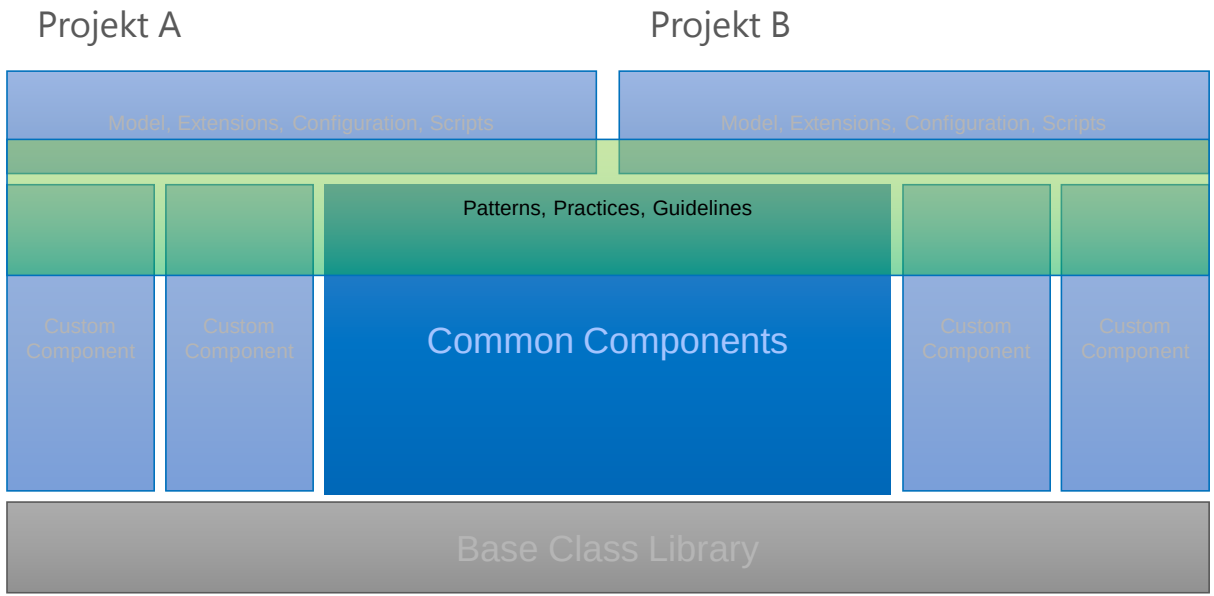
Entwicklung einer Software Factory



Entwicklung einer Software Factory



Entwicklung einer Software Factory



Nette Theorie, aber in der Praxis??

- **Abstraktionsgrad**
  Je abstrakter desto mächtiger ☺
  Je abstrakter desto spezifischer ☹
- **Abhängigkeiten**
  Vertrauen in Werkzeuge
  Vertrauen in Lieferanten
  Vertrauen in Mitarbeiter
- **Kleinster gemeinsamer Nenner**
  Ausnutzung aller Möglichkeiten der zugrunde liegenden Plattform
  Performanceprobleme (Beispiel: OR-Mapper vs. T-SQL)

Werkzeuge

- Klassenbibliotheken
  - Dokumentation
  - Statische Codeanalyse
  - Deployment
- Codegeneratoren
  - Vorlagen
  - Patterns in Form von Assistenten
- Domänenspezifische Sprachen
  - XML-basierend oder
  - individuell (Compiler-Compiler)
  - Compiler (Codegenerator) vs. interpretiert
- Scriptsprachen
- Anwendungsmodularisierung
- Prozessautomatisierung
  - Qualitätssicherung
  - Build

- MS Framework Design Guidelines
  - Sandcastle
  - StyleCop, Code Analysis, 3rd party tools
  - NuGet
- Codegeneratoren
  - T4, ANTLR StringTemplates
  - Visual Studio Templates
- Domänenspezifische Sprachen
  - XAML (UI und Workflows), EF
  - ANTLR (Compiler-Compiler)
- DLR, Project „Roslin“
- MEF, MAF
- Prozessautomatisierung
  - Visual Studio Unit Tests
  - TFS Buildautomatisierung

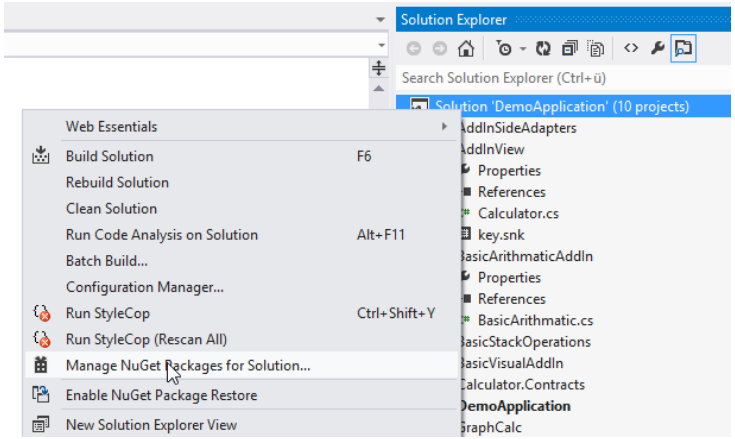
NuGet

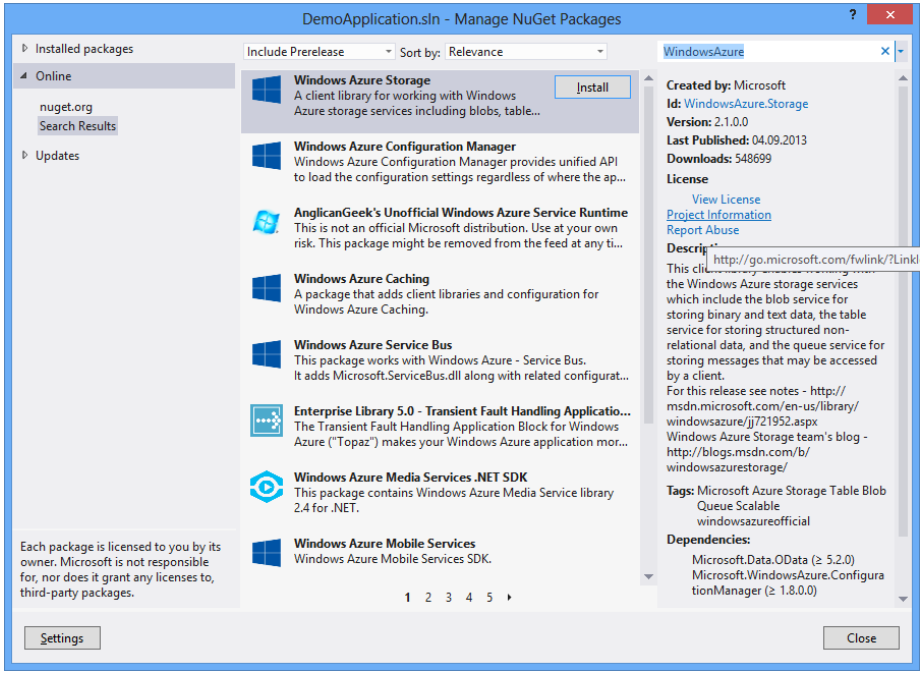
Package Manager für die Microsoft-Entwicklungsplattform

Was ist NuGet?

- Werkzeug zur einfachen Verteilung von Paketen (=Libraries und Tools)
- Alles Notwendige in einem Paket
  Binaries für verschiedene Plattformen
  Optional Symbole und Sourcen
  Anpassungen am Projekt (z.B. Referenzen, Änderungen am app/web.config)
- UI-Integration in Visual Studio
  Ab VS2010, eingeschränkt auch in Mono
  Express-Editionen werden unterstützt
- http://www.nuget.org
  http://nuget.codeplex.com/ (Sourcecode)

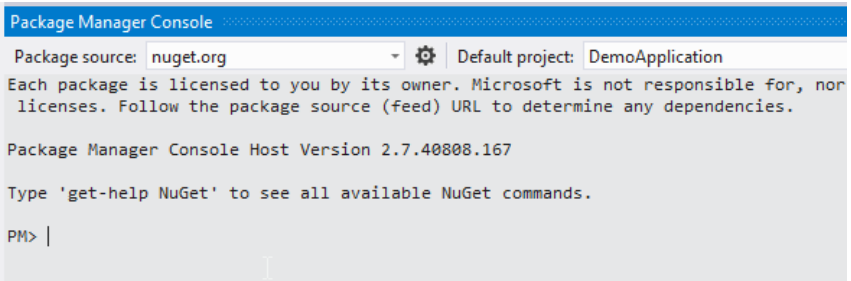
NuGet in Visual Studio

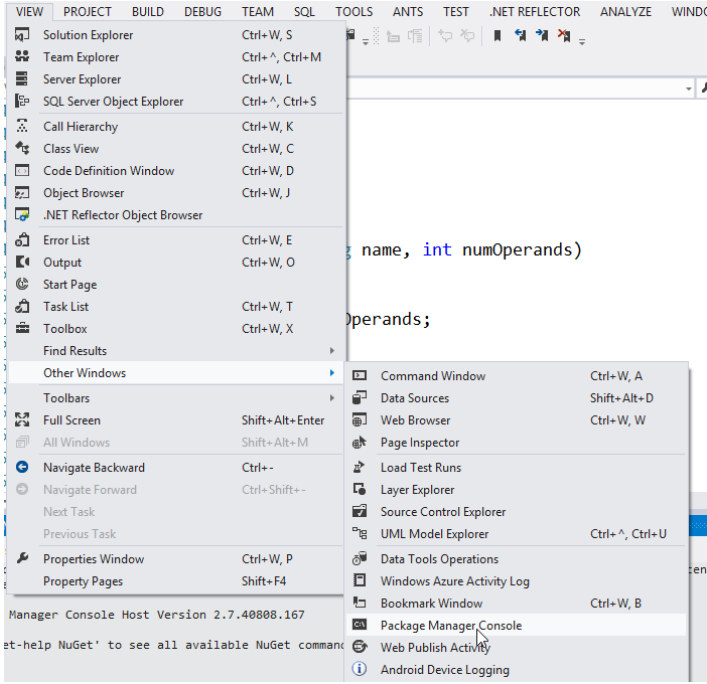




NuGet in Visual Studio

- Package Manager Console
- PowerShell console in Visual Studio
- Automate Visual Studio and NuGet
- NuGet PowerShell Reference





NuGet Pakete erstellen

- Kommandozeilentool *nuget.exe*
  Pakete erstellen (*Pack* Command)
  Pakete veröffentlichen (*Push*, *Delete* Command)
  Paket installieren (*Install*, *Restore*, *Update* Command)
  Generieren eines *nuspec*-Files (*Spec* Command)
  Wichtig für Buildautomatisierung
  Kommandozeilenreferenz
- NuGet Package Explorer
  Grafisches UI zur Erstellung/Bearbeitung von NuGet Paketen und *nuspec* Files
  http://npe.codeplex.com/

```
<?xml version="1.0" encoding="utf-16"?>
<package xmlns="http://schemas.microsoft.com/packaging/2012/06/nuspec.xsd">
  <metadata>
    <id>CockpitFramework.Data</id>
    <version>$version$</version>
    <title>Cockpit Framework Data Layer</title>
    <authors>software architects gmbh</authors>
    <owners>software architects gmbh</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>...</description>
    <releaseNotes></releaseNotes>
    <dependencies>
      <group targetFramework=".NETFramework4.0">
        <dependency id="CockpitFramework.Dependencies"
          version="[$version$]" />
      </group>
      <group targetFramework="sl5">
        <dependency id="CockpitFramework.Dependencies"
          version="[$version$]" />
      </group>
    </dependencies>
  </metadata>
```

Example

nuspec File

```
<dependencies>
  <group targetFramework=".NETFramework4.0">
    <dependency id="log4net" version="[1.2.11]" />
    <dependency id="Microsoft.SqlServer.Compact.Private"
      version="[4.0.8482.1]" />
    <dependency id="AvalonEdit" version="[4.2.0.8783]" />
    <dependency id="ClosedXML" version="[0.68.1]" />
    <dependency id="DocumentFormat.OpenXml" version="[1.0]" />
    <dependency id="IronPython" version="[2.7.3]" />
    <dependency id="LumenWorks.Framework.IO" version="[1.0.0]" />
    <dependency id="Newtonsoft.Json" version="[5.0.6]" />
    <dependency id="WindowsAzure.Storage" version="[2.0.5.1]" />
    <dependency id="Microsoft.Bcl.Async" version="[1.0.16]" />
  </group>

  <group targetFramework="sl5">
    …
  </group>
</dependencies>
```

Example

nuspec File

Version range syntax

```
1.0   = 1.0 ≤ x
(,1.0]  = x ≤ 1.0
(,1.0)  = x < 1.0
[1.0] = x == 1.0
(1.0) = invalid
(1.0,) = 1.0 < x
(1.0,2.0) = 1.0 < x < 2.0
[1.0,2.0] = 1.0 ≤ x ≤ 2.0
empty = latest version.
```

```
  <files>
    <!-- net4 -->
    <file src=".\$configuration$\TimeCockpit.Common.dll"
      target="lib\net4" />
    <file src=".\$configuration$\TimeCockpit.Data.dll"
      target="lib\net4"/>
    …

    <!-- sl5 -->
    <file src=".\SL\$configuration$\TimeCockpit.Common.dll"
      target="lib\sl5" />
    <file src=".\SL\$configuration$\TimeCockpit.Data.dll"
      target="lib\sl5" />
    …

    <!-- include source code  for symbols -->
    <file src=".\...\*.cs" target="src\TimeCockpit.Common" />
    <file src=".\...\*.cs" target="src\TimeCockpit.Data" />
</package>
```

Example

nuspec File

Folder Structure

For Details see [NuGet Docs]

```
\lib
    \net11
        \MyAssembly.dll
    \net20
        \MyAssembly.dll
    \net40
        \MyAssembly.dll
    \sl40
        \MyAssembly.dll
```

```
\content
    \net11
        \MyContent.txt
    \net20
        \MyContent20.txt
    \net40
    \sl40
        \MySilverlightContent.html

\tools
    init.ps1
    \net40
        install.ps1
        uninstall.ps1
    \sl40
        install.ps1
        uninstall.ps1
```

Versioning Notes

- Things to remember
  - NuGet never installs assemblies machine-wide (i.e. not in GAC)
  - You cannot have multiple versions of the same DLL in one AppDomain
- DLL Hell
  - Policy too loose: Problems with breaking changes
  - Policy too tight: Problems with library having dependencies on other libraries (e.g. ANTLR and ASP.NET MVC, everyone depending on Newtonsoft JSON)
- For Library Publishers: SemVer
  - *X.Y.Z* (Major.Minor.Patch)
  - Rethink your strong naming policies

```
<runtime>
   <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
         <assemblyIdentity name="mscorcfg"
           publicKeyToken="b03f5f7f11d50a3a" culture=""/>
         <bindingRedirect
           oldVersion="0.0.0.0-65535.65535.65535.65535"
           newVersion="1.0.3300.0"/>
      </dependentAssembly>
   </assemblyBinding>
</runtime>
```

Binding Redirects

Note: NuGet can generate this for you

Add-BindingRedirect Command

See online reference for details

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
    <package id="SomePackage"
      version="2.1.0"
      allowedVersions="[2,3)" />
</packages>
```

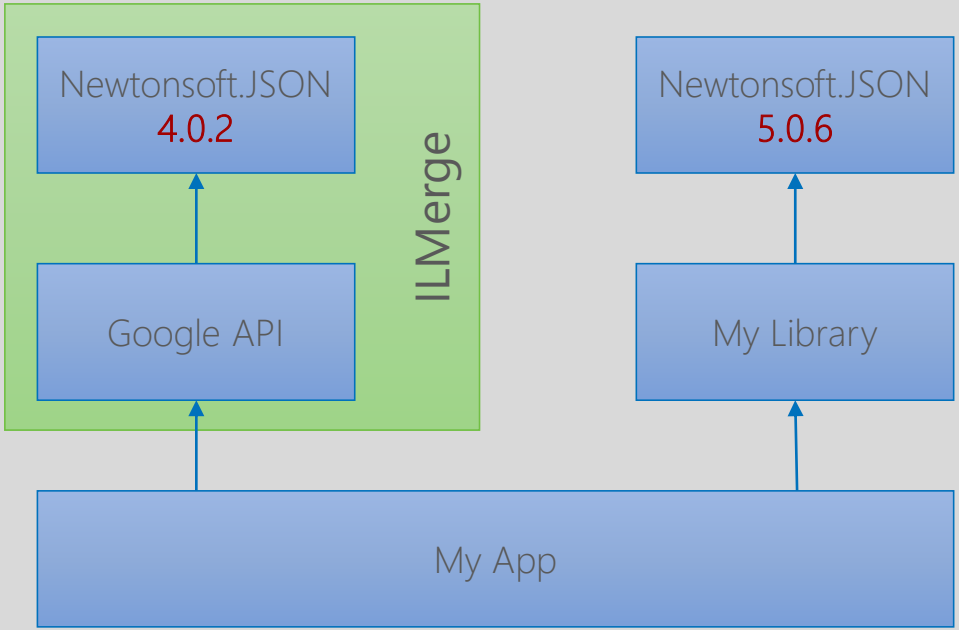
Versioning

Constraings in packges.config

Manual editing necessary

ILMerge

Soliving version conflicts



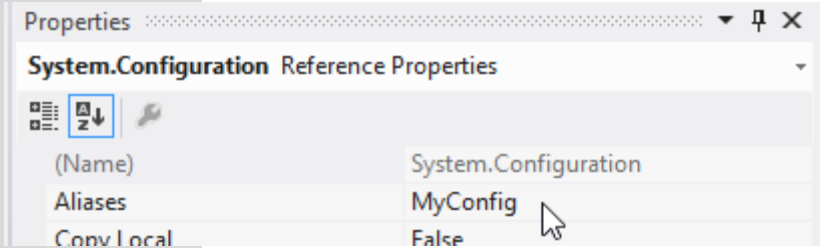
[Microsoft Download]

```
ILMerge
  "Assemblies\Google.Apis.Authentication.OAuth2.dll"
  "Assemblies\Google.Apis.dll"
  "Assemblies\Google.Apis.Latitude.v1.dll"
  "Assemblies\DotNetOpenAuth.dll" "Assemblies\log4net.dll"
  "Assemblies\Newtonsoft.Json.Net35.dll"
  /out:"c:\temp\Google.Apis.All.dll" /lib:"Lib„

extern alias MyConfig;
using Conf = MyConfig::System.Configuration;

namespace MyTinyMvvmToolkit
{
  public class NotificationObject
  {
    public void ReadConfiguration()
    {
      var setting =
        Conf.ConfigurationManager.AppSettings["MyDB"];
    }
  }
}
```

ILMerge

Soliving version conflicts

C# *extern alias*



```
<?xml version="1.0" encoding="utf-16"?>
<package xmlns="http://schemas.microsoft.com/packaging/2012/06/nuspec.xsd">
  <metadata>…</metadata>

  <files>
    <file src="content\app.config.transform"
      target="content\" />
    <file src="content\TimeCockpitInitialization.cs.pp"
      target="content\" />
  </files>
</package>
```

```
<configuration>
    <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
   </startup>
   <system.data>
      <DbProviderFactories>
         <remove invariant="System.Data.SqlServerCe.4.0"/>
         <add name="Microsoft SQL Server Compact Data Provider 4.0"
               invariant="System.Data.SqlServerCe.4.0"
               description=".NET Framework Data Provider for Microsoft SQL Server Compact"
               type="System.Data.SqlServerCe.SqlCeProviderFactory, System.Data.SqlServerCe,
               Version=4.0.0.1, Culture=neutral, PublicKeyToken=89845dcd8080cc91"/>
      </DbProviderFactories>
   </system.data>
</configuration>
```

Content Files

New in NuGet 2.6: XDT

```
…

namespace $rootnamespace$
{
  using System;

  /// <summary>
  /// Class taking care of cockpit framework initialization
  /// </summary>
  public class TimeCockpitInitialization
  {
    …
  }
}
```

Content Files

Sourcecode Transformations in .cs.pp File

Available properties see MSDN

User PowerShell scripts to modify project properties
NuGet Docs

Demo

Nuget in Practice

Nuget at software architects

Nuspec files

- Files
- Dependencies
- Build

Packages in NuGet Explorer

Packages in VS

Publishing NuGet Packages

- http://www.nuget.org
  Public NuGet Feed
- Create your private feed
  File system
  Private NuGet Server
  For details see NuGet Help
- Use a NuGet SaaS like *MyGet*
  http://www.myget.org/

MEF

Managed Extensibility Framework

Original Goals

- Before MEF
  Multiple extensibility mechanism for different Microsoft tools (e.g. Visual Studio, Trace Listeners, etc.)
  Developers outside of MS had the same problem
- MEF: Provide standard mechanisms for hooks for 3rd party extensions
- Goal: Open and Dynamic Applications
  Make it easier and cheaper to build extensible applications and extensions

MEF vs. MAF

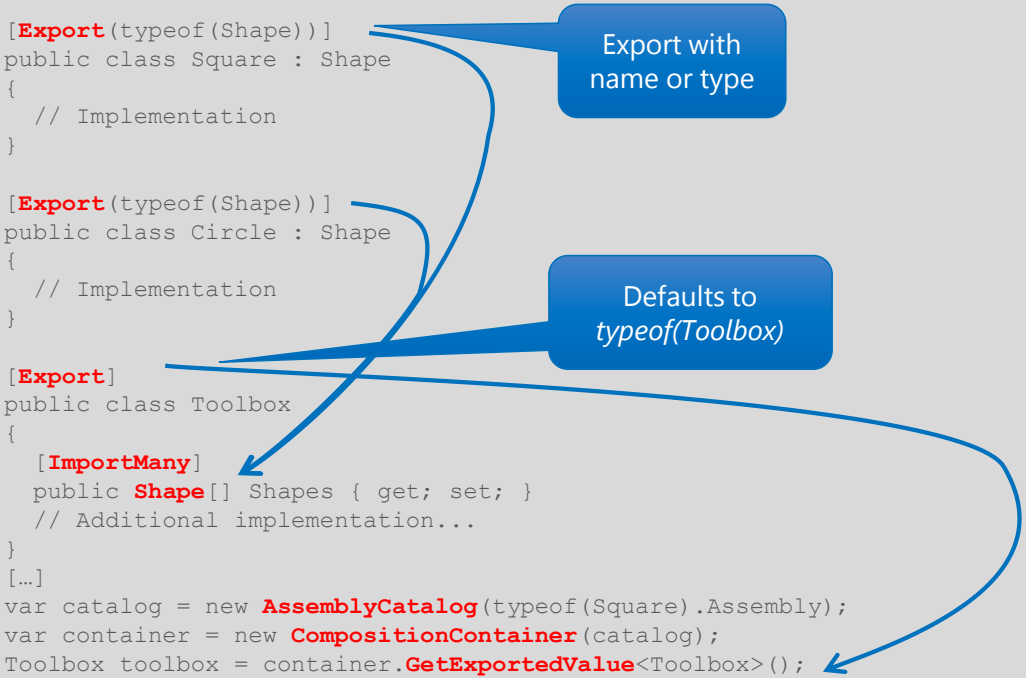
- Managed AddIn Framework
  System.AddIn
- MAF has higher-level goals
  Isolate extension
  Load and unload extensions
  API Compatibility
- Adding MAF leads to higher effort than adding MEF
  A single application can use both

MEF „Hello World“

Anatomy of a program with MEF

Attributed Programming Model

```
[Export(typeof(Shape))]
public class Square : Shape
{
  // Implementation
}

[Export(typeof(Shape))]
public class Circle : Shape
{
  // Implementation
}

[Export]
public class Toolbox
{
  [ImportMany]
  public Shape[] Shapes { get; set; }
  // Additional implementation...
}
[…]
var catalog = new AssemblyCatalog(typeof(Square).Assembly);
var container = new CompositionContainer(catalog);
Toolbox toolbox = container.GetExportedValue<Toolbox>();
```



MEF „Hello World“

- Parts
  Square, *Circle* and *Toolbox*
- Dependencies
  Imports (*Import*-Attribute)
  E.g. *Toolbox.Shapes*
- Capabilities
  Exports (*Export*-Attribute)
  E.g. *Square*, *Circle*

MEF „Hello World“

MEF Basics

- Basic Exports
- Basic Imports
- Catalogs
- Composition

Demo

Exports And Imports

- *Export* attribute
  - Class
  - Field
  - Property
  - Method
- *Import* attribute
  - Field
  - Property
  - Constructor parameter
- Export and import must have the same contract
  - Contract name and contract type
  - Contract name and type can be inferred from the decorated element

Inherited Exports

```
[Export]
public class NumOne
{
    [Import]
    public IMyData MyData
       { get; set; }
}

public class NumTwo : NumOne
{
}

[InheritedExport]
public class NumThree
{
    [Export]
    Public IMyData MyData { get; set; }
}

public class NumFour : NumThree
{
}
```

Import automatically inherited

Export NOT inherited
→ *NumTwo* has no exports

Member-level exports are never inherited

Inherits export with contract *NumThree* (including all metadata)

MEF Catalogs

- Catalogs provide components
- Derived from *System.ComponentModel.Composition.Primitives.ComposablePartCatalog*

AssemblyCatalog
Parse all the parts present in a specified assembly

DirectoryCatalog
Parses the contents of a directory

TypeCatalog
Accepts type array or a list of managed types

AggregateCatalog
Collection of *ComposablePartCatalog* objects

Demo

Catalogs

Loading of modules using *DirectoryCatalog*

```
public class MyClass
{
    [Import]
    public Lazy<IMyAddin> MyAddin
    { get; set; }
}
```

Lazy Imports

Imported object is not instantiated immediately

Imported (only) when accessed

```
[ImportingConstructor]
public MyClass(
  [Import(typeof(IMySubAddin))]
  IMyAddin MyAddin)
{
  _theAddin = MyAddin;
}
```

Could be removed here; automatically imported

Prerequisite Imports

Composition engine uses parameter-less constructor by default

Use a different constructor with *ImportingConstructor* attribute

```
public class MyClass
{
    [Import(AllowDefault = true)]
    public Plugin thePlugin { get; set; }
}
```

Optional Imports

By default composition fails if an import could not be fulfilled

Use *AllowDefault* property to specify optional imports

Creation Policy

- *RequiredCreationPolicy* property
- *CreationPolicy.Any*
  Shared if importer does not explicitly request NonShared
- *CreationPolicy.Shared*
  Single shared instance of the part will be created for all requestors
- *CreationPolicy.NonShared*
  New non-shared instance of the part will be created for every requestor

MEF Object Lifetime

- Container holds references to all disposable parts
  Only container can call *Dispose* on these objects
- Manage lifetime of disposable objects
  Dispose the container → it will dispose all managed objects
  Call *ReleaseExport* on a non-shared object to dispose just this object
  Use *ExportFactory<T>* to control lifetime
- *IPartImportsSatisfiedNotification*
  Implement if you need to get informed when composition has been completed

Demo

Part Lifecycle

Metadata and Metadata views

Advanced exports

Goal

- Export provides additional metadata so that importing part can decide which one to use
- Import can inspect metadata without creating exporting part
- Prerequisite: Lazy import

Metadata

```
namespace MetadataSample
{
   public interface ITranslatorMetadata
   {
      string SourceLanguage { get; }

      [DefaultValue("en-US")]
      string TargetLanguage { get; }
   }
}

namespace MetadataSample
{
   [Export(typeof(ITranslator))]
   [ExportMetadata("SourceLanguage", "de-DE")]
   [ExportMetadata("TargetLanguage", "en-US")]
   public class GermanEnglishTranslator : ITranslator
   {
      public string Translate(string source)
      {
         throw new NotImplementedException();
      }
   }
}
```

Export Metadata can be mapped to metadata view interface

```
namespace MetadataSample
{
   class Program
   {
      static void Main(string[] args)
      {
         var catalog = new AssemblyCatalog( typeof(ITranslator).Assembly);
         var container = new CompositionContainer(catalog);

         // We need a translator from hungarian to english
         Lazy<ITranslator, ITranslatorMetadata> translator =
            container
            .GetExports<ITranslator, ITranslatorMetadata>()
            .Where(t => t.Metadata.SourceLanguage == "hu-HU"
              && t.Metadata.TargetLanguage == "en-US")
            .FirstOrDefault();
      }
   }
}
```

Metadata

(Continued)

Custom Export Attributes

```
[Export(typeof(ITranslator))]
[ExportMetadata("SourceLanguage", "de-DE")]
[ExportMetadata("TargetLanguage", "en-US")]
public class GermanEnglishTranslator
  : ITranslator
{
  public string Translate(
    string source)
  {
    throw new NotImplementedException();
  }
}

[TranslatorExport("de-DE", "en-US")]
public class GermanEnglishTranslator
  : ITranslator
{
  public string Translate(
    string source)
  {
    throw new NotImplementedException();
  }
}
```

Custom export attributes makes code much cleaner.

```
[MetadataAttribute]
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]
public class TranslatorExportAttribute
  : ExportAttribute, ITranslatorMetadata
{
  public TranslatorExportAttribute(
    string sourceLanguage, string targetLanguage)
    : base(typeof(ITranslator))
    {
      this.SourceLanguage = sourceLanguage;
      this.TargetLanguage = targetLanguage;
    }

    public string SourceLanguage { get; private set; }
    public string TargetLanguage { get; private set; }
  }
}
```

Custom Export Attributes

(Continued)

Convention-Based Programming

New in .NET 4.5

Goals

- Reduce the need for attributes
  Note that attributes in the source code override conventions
- Convention-based coupling
  Infer MEF attributes for objects' types
  Example: Export all classes derived from *Controller*
- *System.ComponentModel.Composition.Registration.RegistrationBuilder*
  ForType – creates a rule for a single type
  ForTypesDerivedFrom – creates a rule for all types derived from a certain type
  ForTypesMatching – creates a custom rule based on a predicate
  Returns a *PartBuilder* object that is used to configure imports and exports

```
public interface IShapeMetadata { bool Is2D { get; } };

public class Shape { }
public class Circle : Shape { }
public class Rectangle : Shape { }

[…]

[ImportMany(typeof(Shape))]
private Shape[] shapes;

[…]

// Export all descendants of Shape and add some metadata
var rb = new RegistrationBuilder();
var pb = rb.ForTypesDerivedFrom<Shape>();
pb.Export<Shape>(eb => eb.AddMetadata("Is2D", true))
   .SetCreationPolicy(CreationPolicy.NonShared);

// Use registration builder with catalog
var me = new Program();
var container = new CompositionContainer(
   new AssemblyCatalog(me.GetType().Assembly, rb));
container.ComposeParts(me);
```

Example

Export all descendants of a given class

Add some metadata

Set a creation policy

BASTA! Spring 2014

Q&A

Thank your for coming!

Rainer Stropek

software architects gmbh

Mail rainer@timecockpit.com
Web http://www.timecockpit.com
Twitter @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

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 Mindestdauer und ohne Mindestbenutzeranzahl.