

# **Vorlesung Software aus Komponenten**

## **3. Komponentenmodelle**

apl. Prof. Dr. Hans-Gert Gräbe  
Wintersemester 2009/10

## 3.7. Das .NET-Konzept

### Was ist .NET?

"... komplette Neudefinition der Art, wie Microsoft in Zukunft Geschäfte machen will ... und wie Software entwickelt werden soll."

**Westphal, 2002**

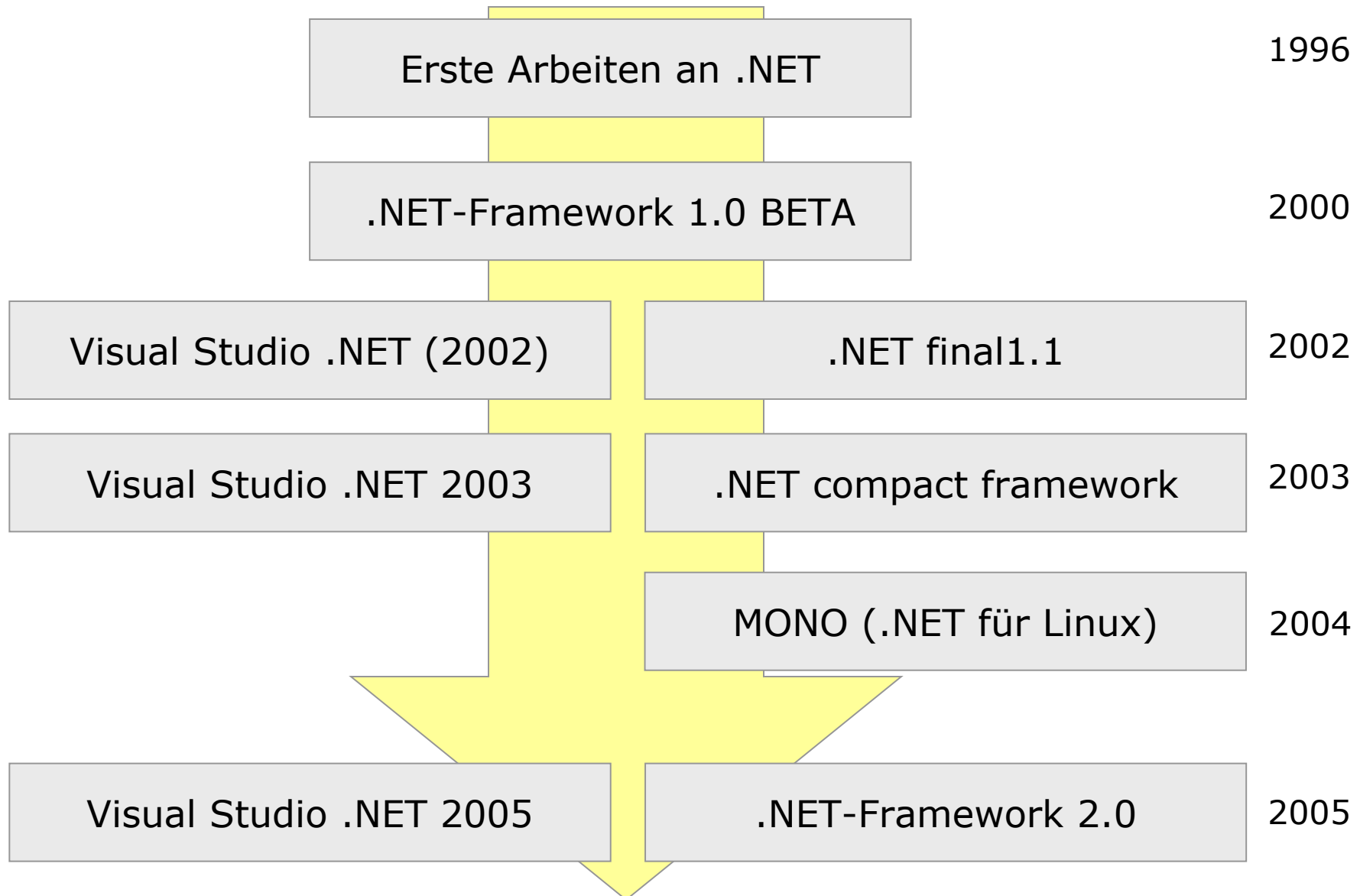
- Plattform soll bisherige Vorgehensweisen der Windows-Programmierung ersetzen, flexibel auf Betriebssystem- und Basisfunktionen zugreifen und Austausch zwischen Programmen unterstützen.
- Ausgerichtet auf den Einsatz auf verschiedenen Hardware-Plattformen bis hin zu Handys und PDAs. Java-Idee ohne Beschränkung auf Java als Programmiersprache
- Ziele
  - Sicherheit
  - Plattformunabhängigkeit
  - Interoperabilität
  - Homogenität

#### Vorgeschichte:

- Rechtsstreit zwischen Sun und Microsoft um Java
  - Microsoft erweitert Java nach eigenen Vorstellungen und Bedürfnissen und gefährdet damit die Java-Kompatibilität
  - Microsoft-Implementierungen J++ und J#
- Weitere Probleme:
  - Auch die für Windowsprogrammierung meist verwendeten Sprachen Visual Basic, C++ und J++ waren nicht kompatibel
  - String-Datentypen sogar nicht binär kompatibel
  - kein einheitliches Modell der Speicherverwaltung

## 3.7. Das .NET-Konzept

### Geschichtliche Einordnung



## 3.7. Das .NET-Konzept

### Geschichtliche Einordnung

- August 2000 – C# und die CLI werden von MS, HP und Intel zur Standardisierung bei der ECMA eingereicht
  - ECMA – European Computer Manufacturers Association
- Dezember 2001 – Fertigstellung des ersten Standards und Weitergabe an die ISO
- April 2003 – Verabschiedung des ISO-Standards
  - ISO/IEC 23270 (C#)
  - ISO/IEC 23271 (CLI)
- Oktober 2003 – Standardisierung der Bindung von C++ an die CLI beginnt
  - ECMA-372 (Dezember 2005)
- 2004: Marktanteil steht noch immer in keinem Verhältnis zur Aufmerksamkeit, die .NET in den Medien findet
- Ende 2005: Visual Studio 2005, .NET Framework 2.0
- Ende 2006: .NET Framework 3.0
- Ende 2007: Visual Studio 2008 und .NET Framework 3.5
- Anfang 2010: .NET Framework 4.0 Beta 2

## 3.7. Das .NET-Konzept

### Prinzipielles Konzept

- Entscheidung für ein **laufzeitbasiertes System**, das nicht direkt aus der Hochsprache in den Maschinencode des Zielsystems kompiliert.
- Aufgreifen der Erfahrungen von Java sowie den Skriptsprachen, dass sich mit zunehmender Komplexität von Software **der Schwerpunkt von Performanz der Programme zur Wartbarkeit verschiebt**.
- Programmcode kann **zur Laufzeit** mittels Reflection über ein Objektmodell generiert und direkt im Speicher in lauffähigen Code übersetzt werden.
- Zusammenarbeit von Code in der Plattform (managed code) und außerhalb der Plattform (unmanaged code) ist möglich.

#### Aktuelle Implementierungen

- Microsoft (Windows) <http://msdn.microsoft.com/net>
- MONO (Linux) <http://www.mono-project.com>
  - Wird vor allem von Novell als SuSe-Eigentümer vorangetrieben

# Microsoft .NET Strategie

## **.NET Framework**

Visual Studio .NET  
Codeeditor  
Fenstereditor  
Debugger  
Server-Explorer  
Entwurfshilfen  
C# / VB .NET / J#  
ASP.NET

## **.NET Enterprise Server**

Application Center  
Exchange Server  
BizTalk Server  
...  
- werden speziell an  
.NET angepasst  
- Dienste zentral  
für .NET Anwen-  
dungen

## **.NET My Services**

konkrete  
WebServices  
  
.NET Contacts  
.NET Wallet  
.NET Lists  
Microsoft Passport  
...

## **.NET Devices**

Hardware wie PDA,  
Handys, Tablet,  
PC, Auto-PC ...  
  
- Mobile Internet  
Toolkit (MIT)  
- .NET Compact  
Framework (CF)

# 3.7. Das .NET-Konzept

## Die Entwicklungsumgebung von Microsoft

### Visual Studio .NET Enterprise Architect

Integration mittels Visio  
Erstellen von Enterprise Templates

+ BizTalk - Server

### Visual Studio .NET Enterprise Developer

VS Analyzer  
Nutzung von Enterprise Templates  
Source Save

+ W2K Server, SQL-, Exchange-, Commerce-, Host Integration Server

### Visual Studio .NET Professional

Entwicklungsumgebung  
Crystal Reports  
Editoren  
Designer  
Wizards

### .NET Framework SDK

Dokumentation, Beispiele, Tools

### .NET Framework Redistributable

Common Language Runtime  
Kommandozeilencompiler für  
VB .NET, C# und JavaScript .NET

- kostenpflichtig
- kostenlos
- kostenloses Add-On

### Mobile Internet Toolkit (MIT)

Mobile WebForm  
Designer für VS .NET

Dokumentation

### Mobile Controls

neue ASP.NET Server-  
Steuerelemente



## 3.7. Das .NET-Konzept

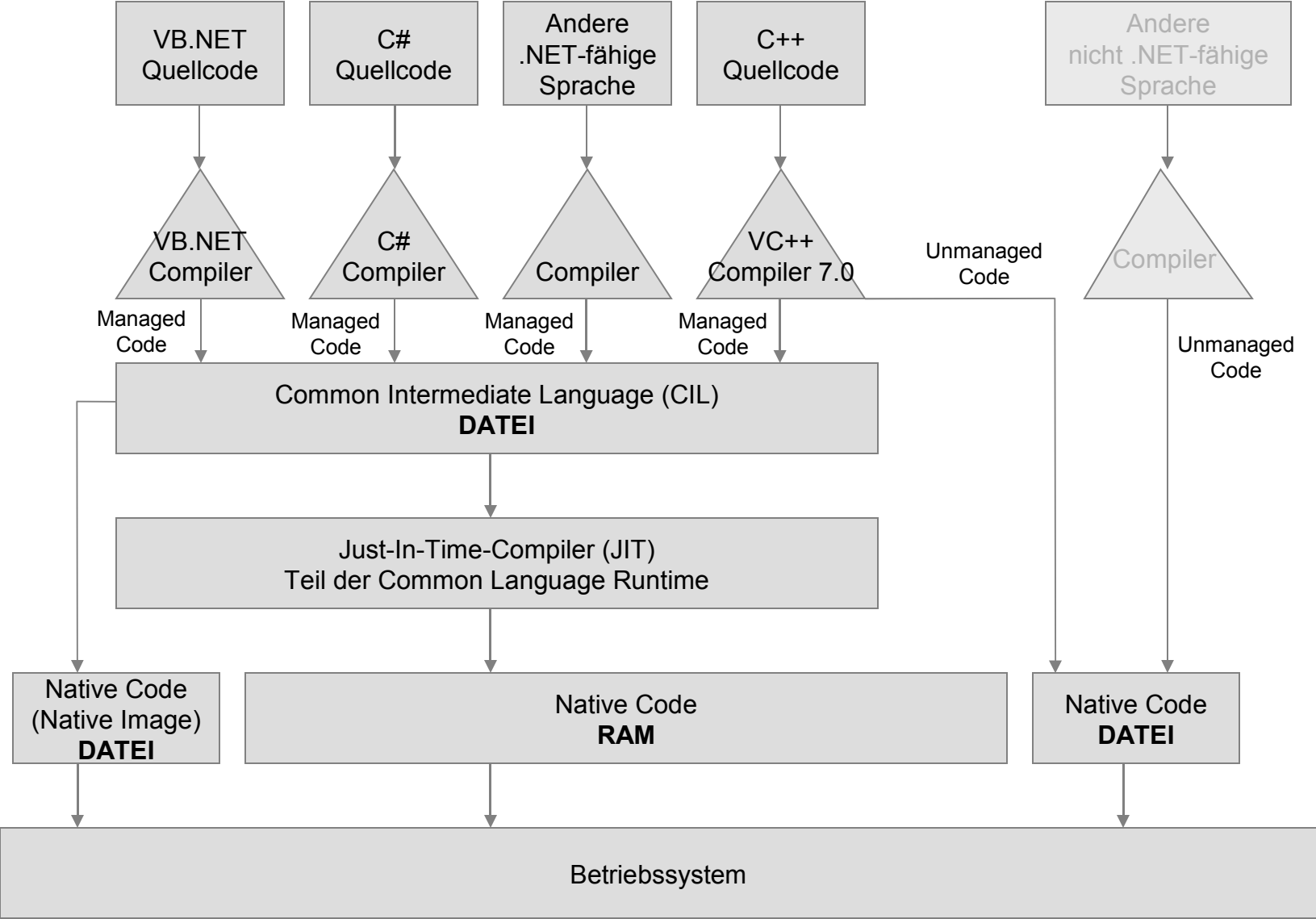
### Konzept

### Das .NET-Konzept

- CLI – **Common Language Infrastructure**
  - Basis zur Ausführung von Programmen, die in unterschiedlichen Programmiersprachen erstellt wurden
  - Zugriff auf eine **virtuelle Maschine** und eine gemeinsame Klassenbibliothek – die **Base Class Library**
- CIL – **Common Intermediate Language**
  - Hochsprachenunabhängige Zwischensprache
- CLR – **Common Language Runtime**
  - Laufzeitumgebung für CIL-Zwischencode
- CTS – **Common Type System**
  - Sicherung der Kompatibilität der Ressourcenzugriffe über einen sprachübergreifenden Standard von OO-Datentypen
  - .NET wurde von Anfang an für den Betrieb mit mehreren Programmiersprachen entwickelt
- Assemblies – **Packungsformat** für Komponenten

# 3.7. Das .NET-Konzept

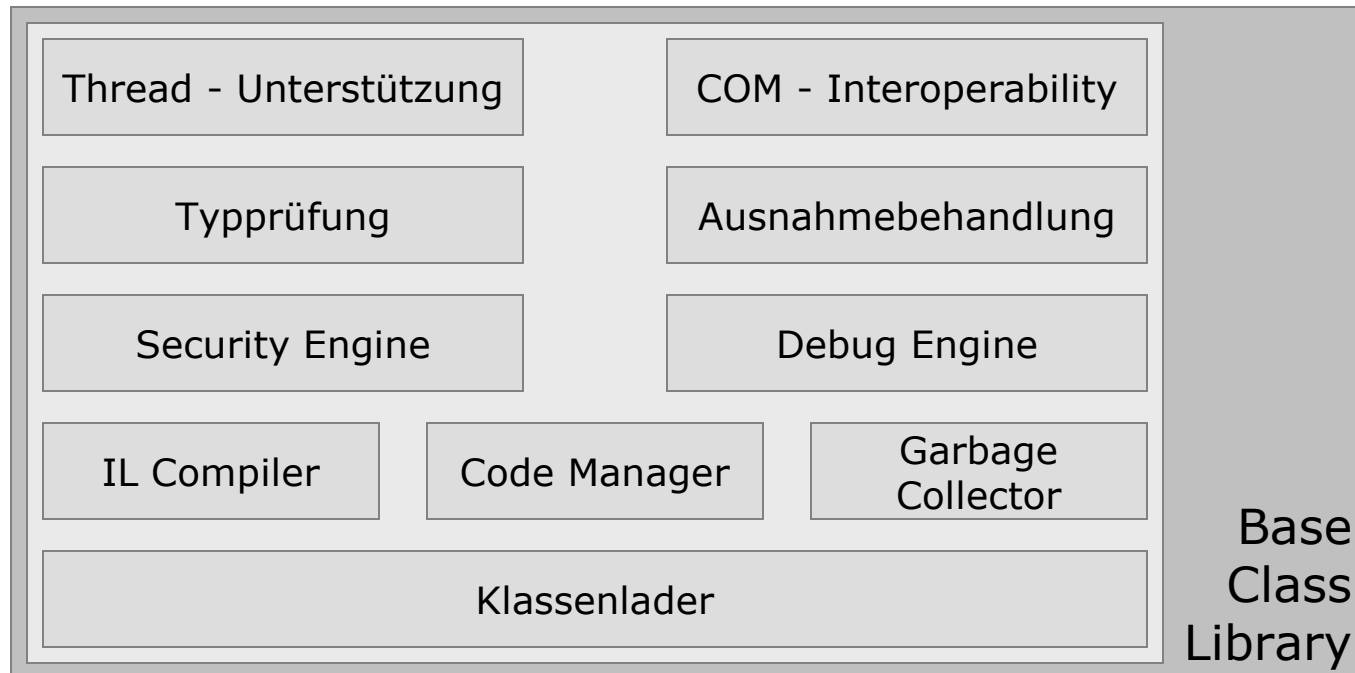
## Konzept



## 3.7. Das .NET-Konzept

### Common Language Runtime

- Übersetzung von Zwischensprachencode (CIL) in Maschinencode
- Speicherverwaltung
- Verwaltung von Prozessen und Threads
- Durchsetzung von Sicherheitsmechanismen
- Laden von Komponenten
- **Alle** .NET-Sprachen setzen auf die CLR als Runtime auf



## 3.7. Das .NET-Konzept

### Common Language Runtime

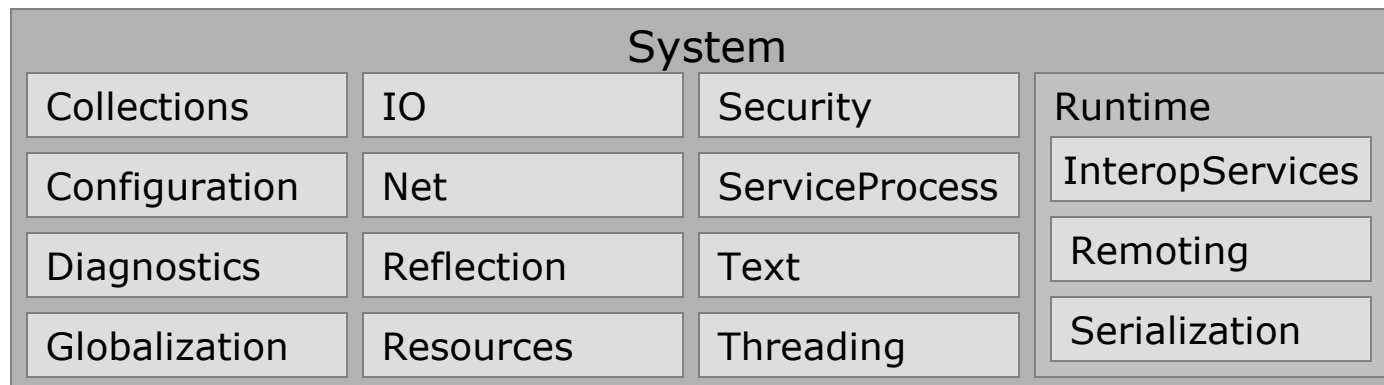
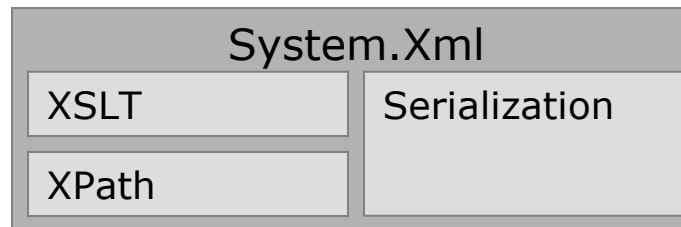
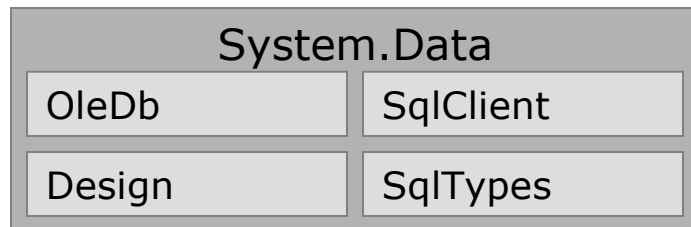
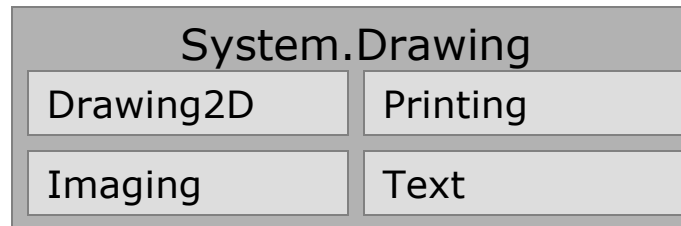
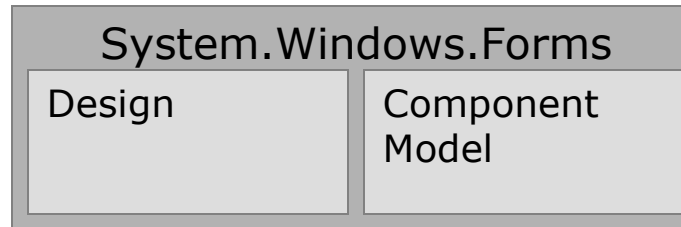
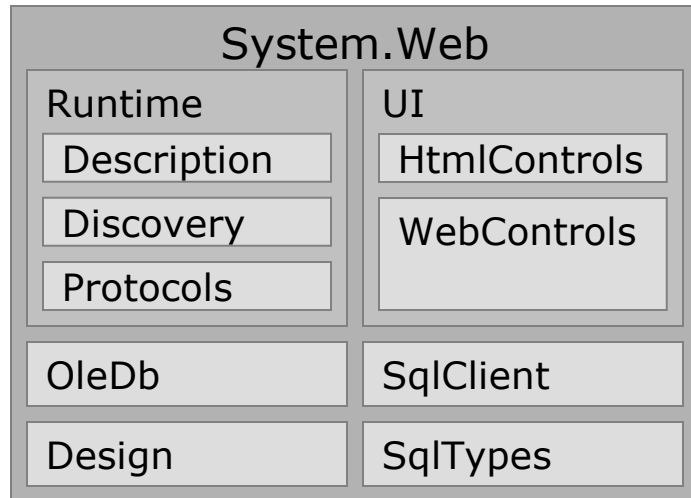
- Konsistentes Programmiermodell
  - alle Anwendungsdienste als objektorientiertes Programmiermodell
- Vereinfachtes Programmiermodell
  - keine Registrierung, COM-Schnittstellen, HRESULTs ...
- Stabile Installationen
  - isolierte Anwendungskomponenten
  - keine >DLL-Hölle< mehr
  - Versionierung von Komponenten
- Vereinfachte Installationen
  - Anwendungsdateien einfach in Zielverzeichnis kopieren
  - keine Registry-Einträge nötig
- Viele verfügbare Plattformen
  - Compiler generiert IL-Code
  - Ausführbar auf Maschinen, die über ECMA-kompatible Versionen der CLR verfügen
- Integration verschiedener Programmiersprachen
  - Typen, die in unterschiedlichen Sprachen geschrieben wurden
  - Common Type System

## 3.7. Das .NET-Konzept Common Language Runtime

- Einfacheres Wiederverwenden von Code
  - durch oben beschriebene Techniken
- Automatische Speicherverwaltung
  - Garbage Collection
- Typsicherheit
  - Zugriff auf Objekte immer auf kompatible Weise
  - Code springt nur an bekannte Stellen (Eintrittspunkt von Methoden)
  - keine Pufferüberläufe
- Komfortables Debuggen
  - Debuggen von Anwendungen unterschiedlicher Sprachen
- Konsistente Fehlerverarbeitung
  - **Alle** Fehler werden über Ausnahmen gemeldet
- Sicherheit
  - basierend auf Herkunft des Codes / der Daten
- Interoperabilität
  - Zugriff auf COM-Komponenten

## 3.7. Das .NET-Konzept

### Base Class Library



- Schnittstelle zum Betriebssystem
- komplett Objektorientiert
- Allen .NET Sprachen stehen dieselben Dienste zur Verfügung
- Zugriff auf Dateisystem, Fensteranzeige, Druckfunktionen, Remoting, Grafik, Datenbankzugriff

## 3.7. Das .NET-Konzept Base Class Library

- CIL ist eine Art "objektorientierter Maschinencode"
- arbeitet Stack-orientiert, keine Register
- unabhängig von CPU
- Verifizierung des Codes durch die CLR bei der Übersetzung in nativen Code
  - nur Speicheradressen lesen, in die vorher Daten geschrieben wurden
  - Methoden mit der korrekten Anzahl von Argumenten aufrufen
  - jedes Argument hat richtigen Typ
- wird zur Laufzeit vom Just-In-Time-Compiler kompiliert
- Caching von bereits übersetzten Typen
- Optimierung anhand ausführender Architektur

```
.method private hidebysig static void Main() cil managed {  
    .entrypoint  
    .maxstack 3  
    .locals ([0] int32 v, [1] object o)  
    IL_0000: ldc.i4.5  
    IL_0001: stloc.0  
    IL_0002: ldloc.0  
    IL_0003: box      [mscorlib]System.Int32  
    ...
```

**Beispiel für IL-Code**

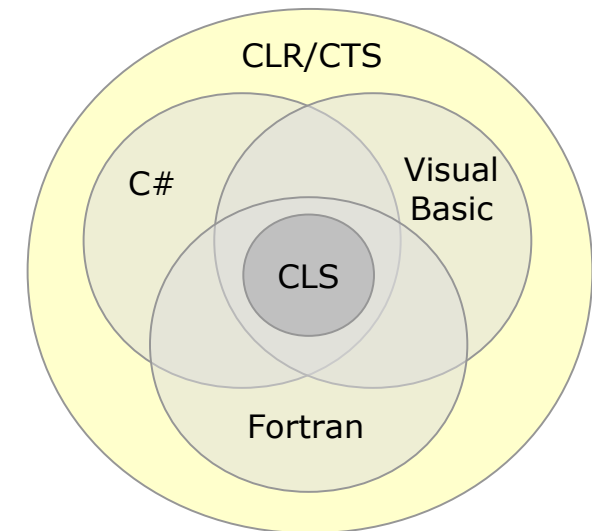
## Common Language Specification

kleinster gemeinsamer Nenner der .NET-Sprachen

- standardisierte Typen
- selbstbeschreibende Typinformationen (Metadaten)
- gemeinsame Ausführungsumgebung

```
using System;
[assembly:CLSCompliant(true)]
// Compiler soll CLS-Kompatibilität prüfen

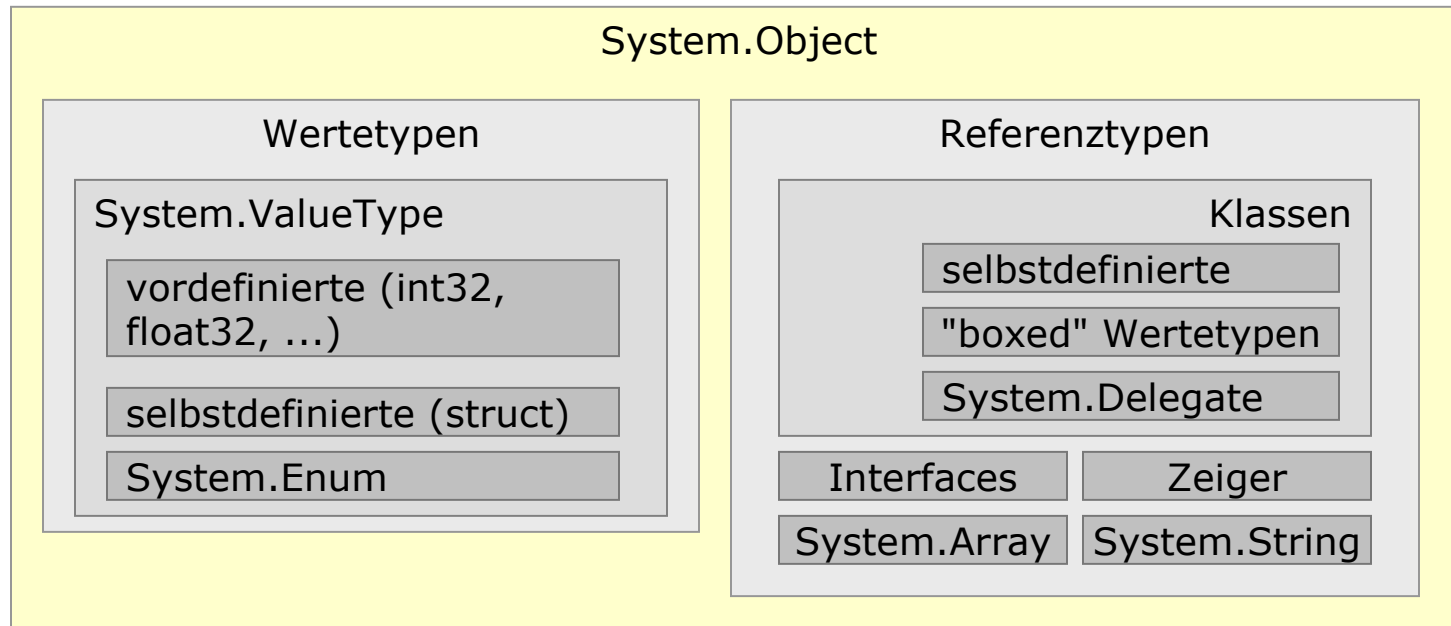
// Fehler, weil Klasse öffentlich ist
public class App {
    // Fehler, weil UInt32 nicht CLS-Kompatibel
    public UInt32 Abc() { return 0; }
    // Fehler, weil keine Unterscheidung zwischen
    // Groß- und Kleinschreibung in CLS
    public void abc() {}
    //Kein Fehler, da Methode privat ist
    private UInt32 ABC() { return 0;}
```



- Vollständige Liste der CLS-Regeln im Abschnitt „Cross-Language Interoperability“ in der Dokumentation des .NET Framework SDK



### Common Type System



- Wertetypen enthalten Werte (liegen auf dem Stack)
- Referenztypen zeigen auf Werte (Werte liegen auf dem Heap)
- Wertetypen können ausdrücklich als Objekte behandelt (und damit auf dem Heap abgelegt) werden: Boxing

## 3.7. Das .NET-Konzept Common Type System (CTS)

### Verweis- und Referenztypen

```
struct MyStruct {
    int i;
    float f;
}
class MyClass {
    int k;
    MyStruct t
}
```

```
int j;
j = 1234;

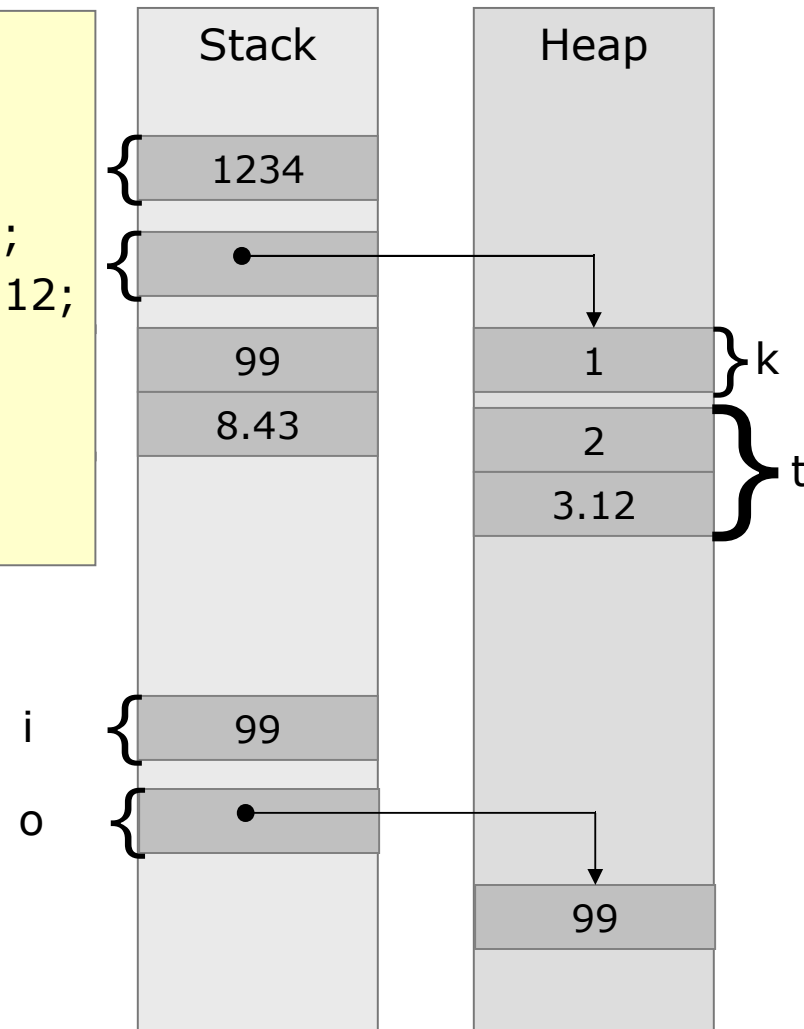
MyClass c = new MyClass();
c.k = 1; c.t.i = 2; c.t.f = 3.12;

MyStruct s;
s.i = 99; s.f = 8.43
```

#### Boxing

```
int i;
i = 99;

object o;
o = i;
```



## 3.7. Das .NET-Konzept Common Type System (CTS)

### Klassendefinition in C#

```
class DatenKlasse  
{  
    int i;  
    float f;  
    string s;  
    int[] ai;  
}
```

### Klassendefinition in VB .NET

```
Private Class DatenKlasse  
    Dim i As Integer  
    Dim f As Single  
    Dim s As String  
    Dim ai() As Integer  
End Class
```



### Ausführbare Komponente auf IL-Ebene

```
.class private auto ansi beforefieldinit DatenKlasse  
extends [mscorlib]System.Object  
{  
    .field private int32 i  
    .field private float32 f  
    .field private string s  
    .field private int32[] ai  
  
    .method public hidebysig specialname rtspecialname  
instance void .ctor() cil managed  
    { ... }  
}
```

## 3.7. Das .NET-Konzept Common Type System (CTS)

- Typsicherheit
  - CLR weiß zur Laufzeit **immer** um den Typen eines Objektes
  - Objekt kann seinen Typ nicht manipulieren
- Konvertierung (Type Casting)
  - Objekt kann in einen seiner Basistypen konvertiert werden (z.B. Int32 -> Object) – implizite Konvertierung
  - wenn ein Objekt in einen abgeleiteten Typen umgewandelt werden soll, muss explizit konvertiert werden

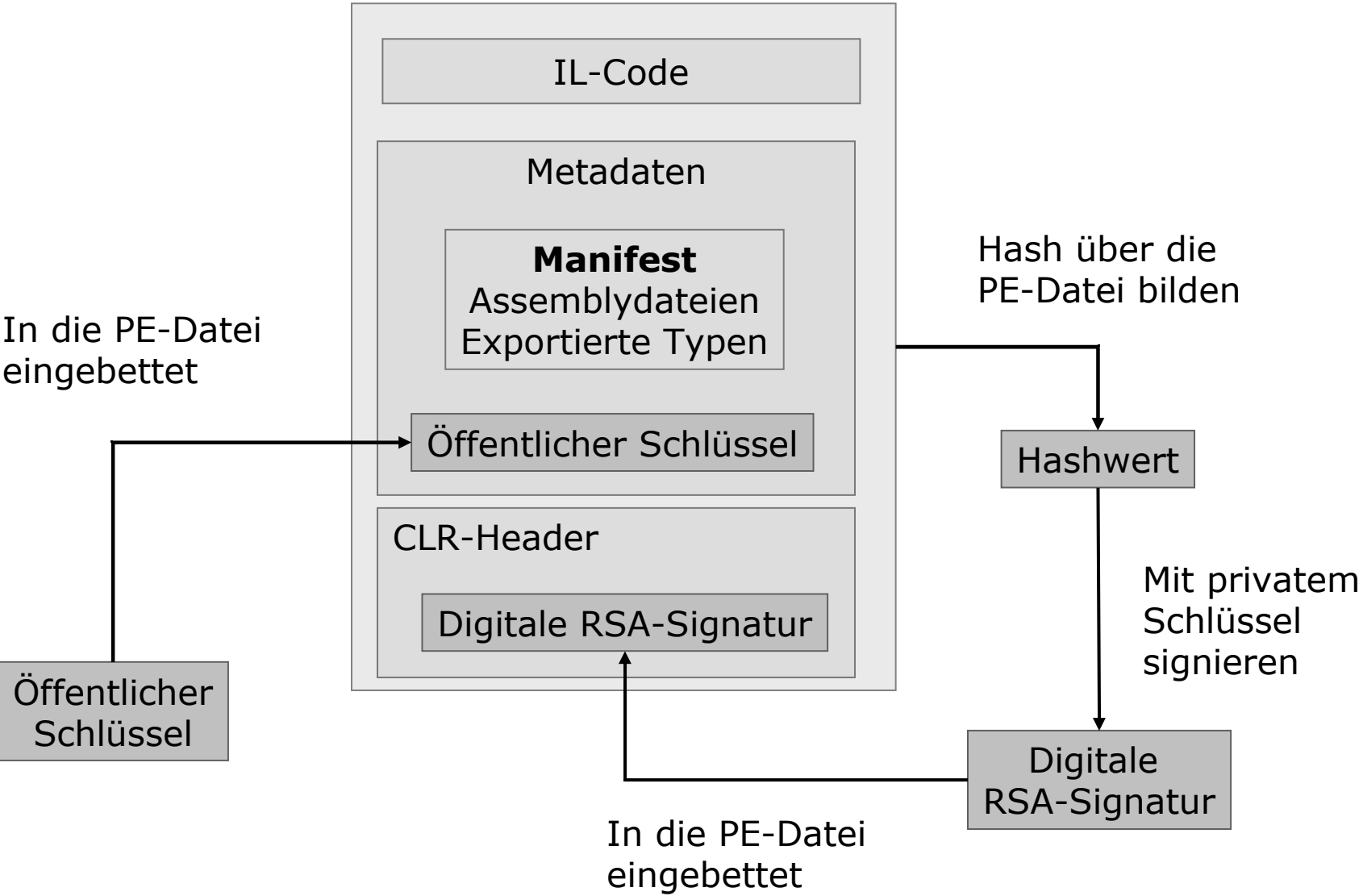
```
class SHK : Student { ... }  
class App {  
    public static void Main() {  
        SHK x = new SHK();  
        EvaluateStudent(x); // Ok, da SHK vom Typ Student abgeleitet wurde  
  
        DateTime newYear = new DateTime(2001, 1, 1);  
        EvaluateStudent(newYear);  
        // Ausnahme in EvaluateStudent, da DateTime nicht von Student abgeleitet wurde  
    }  
    public void EvaluateStudent(Object o) {  
        Student s = (Student) o;  
        // Compiler weiß nicht, ob Konvertierung erfolgen kann, erst CLR zur Laufzeit  
        ...  
    }  
}
```

### Komponenten als Packungseinheiten - Assemblies

- atomare, selbstbeschreibende Einheit, inklusive Metadaten
  - Portable Executable (PE-Datei)
- Metadatenstruktur wird **Manifest** genannt
- "single file assembly" - Manifest ist Teil des eigentlichen Codes - oder
- "multi file assembly" - Manifest ist eigenständige Einheit
- ein Manifest enthält ...
  - Identität der Assembly (Name, Version, ...)
  - die Namen aller Dateien in der Assembly
  - kodierte Hashwerte aller Dateien im Assembly
  - Details der vorhandenen Klassen, Methoden und Eigenschaften
  - Namen und Hashwerte aller referenzierten Assemblies
  - Sicherheitseinstellungen

- starke Namen (strong names) zur eindeutigen Identifizierung einer Assembly
  - zusammengesetzt aus Dateinamen (ohne Erweiterung), Versionsnummer, Kultur, Token für einen öffentlichen Schlüssel
  - festdefinierte Struktur:  
Name\_Versionsnummer\_Kultur\_Schlüsseltoken
- Signierung = Möglichkeit zur Sicherstellung der Unversehrtheit des Codes
- Assemblies können *privat*, *gemeinsam (shared)* oder *global* sein.
- globale Assemblies werden auf dem Rechner mit dem Werkzeug *gacutil* im globalen Assembly-Zwischenspeicher (Global Assembly Cache (GAC)) gespeichert.
  - Vorteil: Assembly kann global genutzt werden
  - Nachteil: Keine einfache Installation mehr möglich

# 3.7. Das .NET-Konzept Assemblies



### Weitergabe von Assemblies

- Kopieren von Dateien in Zielverzeichnis, z.B. per Batch
  - alle abhängigen Verweise und Typen sind in Assembly enthalten
  - referenzierte Assemblies im Anwendungsverzeichnis
- Konventionelle Installation möglich (cab, msi ...)
  - Verknüpfungen auf Desktop, Startleiste ...
  - Einbindung in Softwareverwaltung von Windows
  - zukünftig eventuell Automation von Verknüpfung
- Private Assemblies
  - werden in Anwendungsverzeichnis installiert
  - werden **nur** von jeweiliger Anwendung benutzt
  - es werden nur Typen gebunden, die für die Anwendung passen



### Weitere Features

- Verbindung von managed und unmanaged code über Interop-Technik
  - Einbindung traditioneller COM-Programme über .NET-Kapsel möglich
  - unmanaged code oft an performanzkritischen Stellen
  - Bedeutung relativiert durch die Erfahrung, dass es oft effizienter ist, die Algorithmen zu verbessern statt die Implementierung
- Komplexes Sicherheitskonzept
  - Authentizität des Herstellers
  - Schutz der Programme vor Veränderung
  - codebasiertes und nutzerbasiertes Sicherheitsmodell zur Beschreibung der Vertrauenswürdigkeit des Codes

## 3.7. Das .NET-Konzept

### Weitere Features

### Weitere Features

- Attributbasiertes Programmieren
  - Dependency Injection zur Übernahme von Diensten der Umgebung (Assembly) in das Programm
  - wurde von Java als Annotationen z.B. in EJB 3.0 übernommen
- Basisklassen-Bibliothek deckt alle wichtigen Grundfunktionalitäten ab
  - z.B. Textformatierung, Email-Versand, Codegenerierung
  - Basisklassen zur Anbindung an den Web Service Standard als Grundlage für verteilte Anwendungen
- Integration der Laufzeitumgebung in Windows Server seit 2003
  - explizit in Konkurrenz zu J2EE
  - Kernbestandteil von Windows Vista
  - abgespeckte Version der Laufzeitumgebung für Kompaktgeräte

### Weitere Features

- Bindung verschiedener Programmiersprachen an dieselbe Plattform erlaubt Erstellung einer Applikationen unter Verwendung verschiedener Hochsprachen
  - wird durch sprachunabhängige Editoren wie Eclipse oder Visual Studio .NET
  - Gemeinsame Verwendung von Bibliotheken
  - Wartbarkeit derartiger Projekte ist deutlich komplexer als solcher, die nur in einer Hochsprache geschrieben sind.
  - Ändert sich mglw. mit dem Einsatz generativer Techniken und modellgetriebener Software-Entwicklungsansätze
- Hoher Anklang der Plattform vor allem in marktengen und akademischen Bereichen, wo Programmiermöglichkeiten voll ausgereizt werden.

### Bestandteile des Frameworks

- Common Intermediate Language / Common Language Runtime
  - Zwischensprachencode wird in Maschinencode übersetzt und ausgeführt
  - kleinste Schnittmenge für Sprachen, die von CLR unterstützt werden
- Base Class Library
  - Bibliotheken mit wichtigen Basisfunktionen werden erst auf der Ebene der CIL eingebunden und sind so hochsprachenunabhängig verfügbar
- Common Language Specification / Common Type System
  - Vereinigung der Sprachkonzepte, die im .NET-Framework unterstützt werden, um Datenkompatibilität auf Hochsprachenebene zu sichern
- Assemblies
  - Komponentenkonzept zum Kapseln von Software-Bausteinen
  - mit Metadaten, Integritätscheck, Versionierung
  - Konzept des Managements von Abhängigkeiten zu anderen Assemblies
  - Global einheitliches Namenssystem (GAC)
  - werkzeuggestütztes Verwaltungssystem für Assemblies
  - Administration über Konfigurationsdateien (XML)