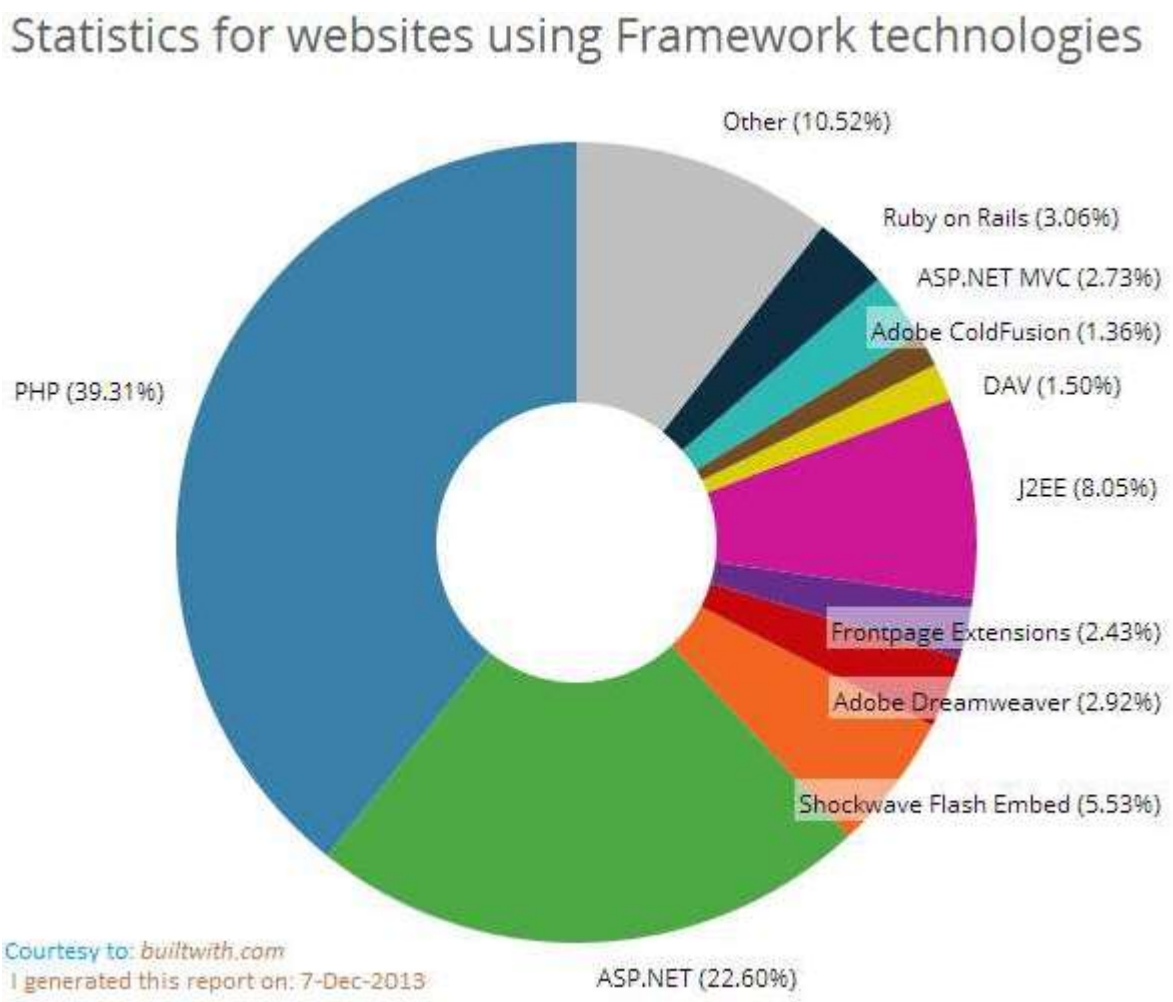
Introduction

Dienstag, 22. November 2016 12:46

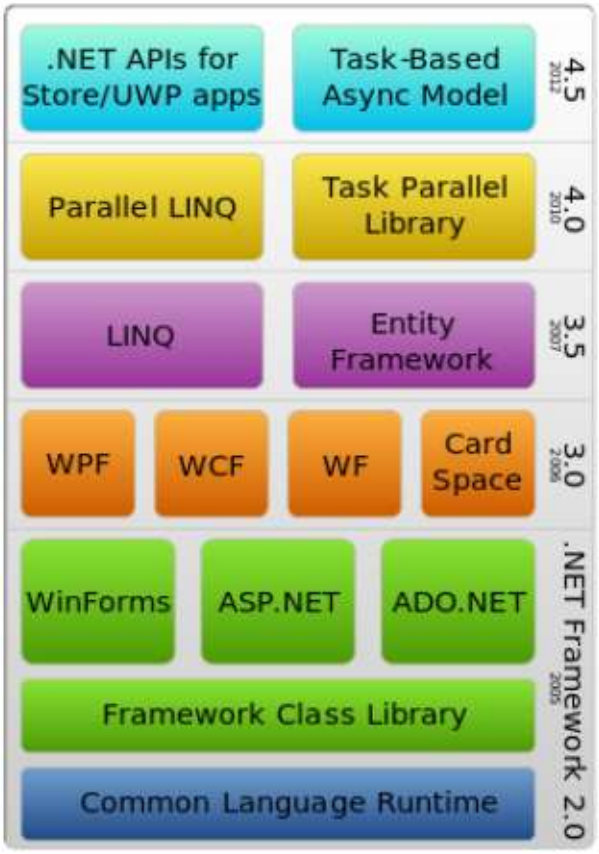
Microsoft .NET Platform

- 2002 eingeführt
- Hauptprogrammierplattform für Windows



Aus <http://codepattern.net/Blog/post/PHP-vs-ASP-NET>

.NET Versionen und Komponenten



Features

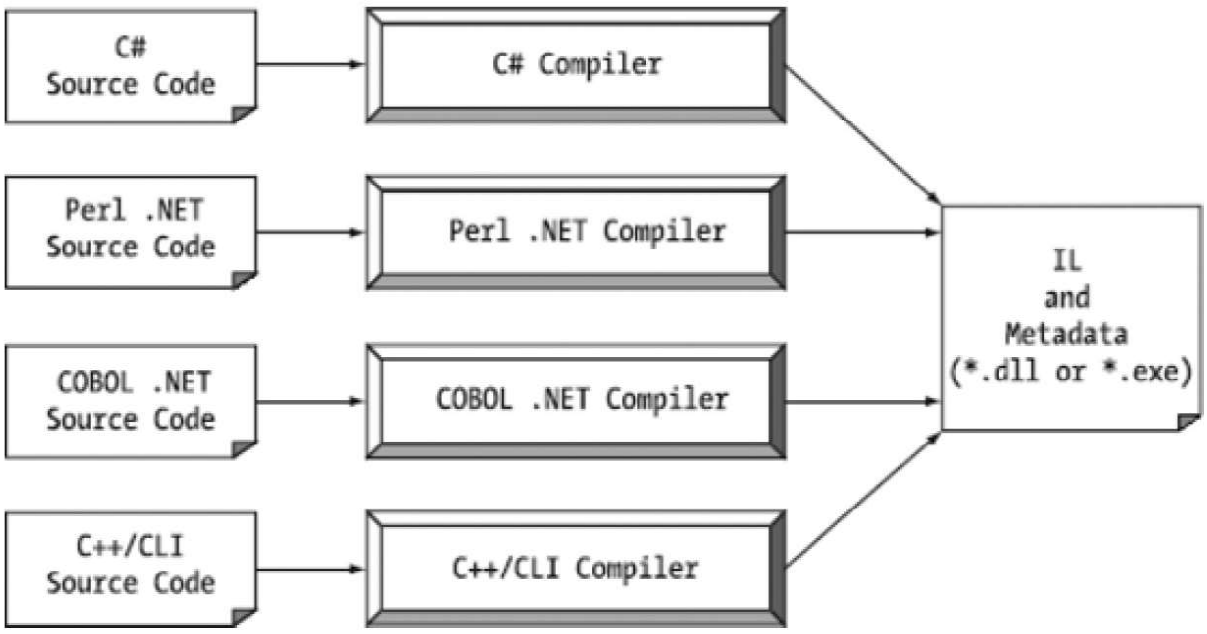
- Komponentenbasiert: Unterschiedliche Komponenten aus unterschiedlichen Programmiersprachen:
    - C#, Visual Basic, F#, JavaScript, C++ (in Visual Studio)
    - Ruby, Python, COBOL, Pascal (externe Compiler)
    - https://en.wikipedia.org/wiki/List_of_CLI_languages
  - Vererbung, Exception Handling, Debugging über mehrere Programmiersprachen hinweg
- Versionskompatibilität: mehrere Versionen der gleichen .dll können nebeneinander existieren
- .NET Distributionen:
  - Microsoft, included in Visual Studio
  - Andere Distributionen (www.mono-project.com, www.dotgnu.org) stellen Compiler und Runtime Environment für Linux Android, IOS, …
  - .NET Core (plattformunabhängig, schlank)
- Erstellte Anwendungen sind Plattform-unabhängig, und laufen unter verschiedenen Betriebssystemen (Windows, MacOS, Linux, Android, …)

Warum mehrere Programmiersprachen ?

Wie tauschen sich die unterschiedlichen Programmiersprachen untereinander aus ?

Compilierung in .NET

- Erzeugt *.dll Bibliotheken und *.exe Dateien -> heißen Assembly
- Diese Dateien haben bis auf ihre Endung nichts mit normalen .exe, .dll Dateien zu tun
- Dateien enthalten Code in einer Intermediate Language (IL) + Metadaten + Manifest
- Enthält keine Plattform-spezifischen Informationen ->plattformunabhängig
- Metadaten beschreiben Typen, bspw. von wem eine Klasse erbt, welche Methoden sie hat, Version, …



C# Code

```
class Calc
{
  public int Add(int x, int y)
  { return x + y; }
}
```

Visual Basic Code

```
Class Calc
  Public Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
    Return x + y
  End Function
End Class
```

Entsprechender IL Code

https://en.wikipedia.org/wiki/List_of_CIL_instructions

```
.method public hidebysig instance int32 Add(int32 x,
   int32 y) cil managed
{
  // Code size 9 (0x9)
  .maxstack  2
  .locals init (int32 V_0)
  IL_0000:  nop
  IL_0001:  ldarg.1
  IL_0002:  ldarg.2
  IL_0003:  add
  IL_0004:  stloc.0
  IL_0005:  br.s IL_0007
  IL_0007:  ldloc.0
  IL_0008:  ret
} // end of method Calc::Add
```

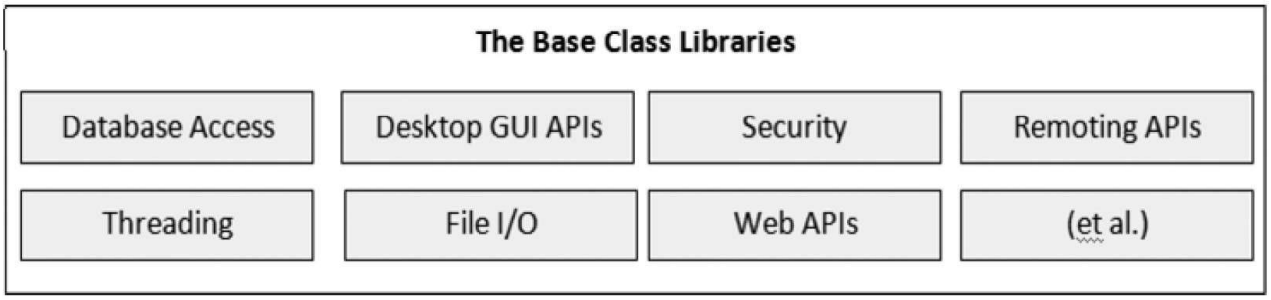
Ausführung des Programms:

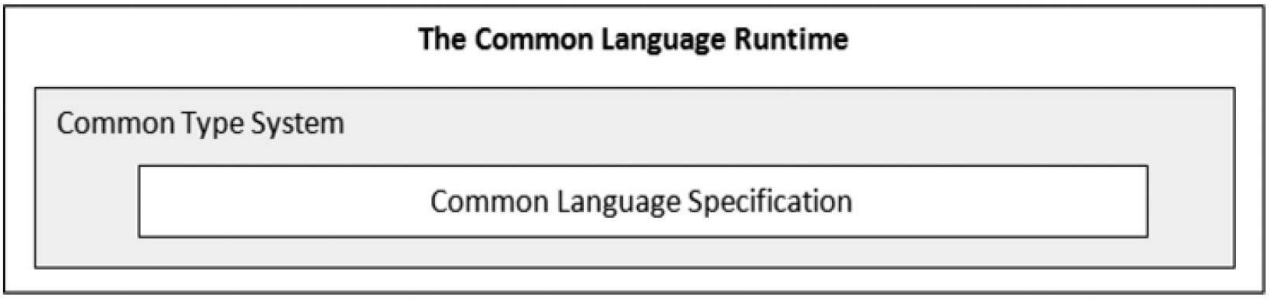
- ruft just-in time compiler (JIT, jitter) auf der den code compiliert und ausführt
- Methoden werden nur einmal kompiliert und im Speicher gelagert, (-> kein Interpreter) so dass mehrfacher Aufruf der gleichen Methode nur eine einzige Kompilierung benötigt.

Teile von .NET

- Common Language Runtime (CLR)
  - Findet, lädt und managed .Net Objekte
  - Nutzt Metadaten der Assembly um Typen zu finden
  - Memory management, thread coordination, …
  - Compiliert nötigen IL Code und führt ihn aus
- Common Type System (CTS)
  - Spezifiziert Datentypen und Programmkonstrukte und deren Interaktion
- Common Language Specification (CLS)

- Common Type System (CTS)
  - Spezifiziert Datentypen und Programmkonstrukte und deren Interaktion
- Common Language Specification (CLS)
  - Definiert eine Teilmenge von CTS, die über alle .NET Sprachen geteilt ist
  - Denn nicht jede .NET Sprache unterstützt alle features und Datentypen der CTS
  - Programmierung nur mit CLS features erzeugt Code Bibliotheken die garantiert mit jeder anderen .NET Sprache kommunizieren können
  - Festgelegt in Regeln
    - Erste Regel: Alle Regeln beziehen sich nur auf die Teile eines Typs, der außerhalb eines Assemblys sichtbar sind
  - C# Compiler kann prüfen, ob die Regeln eingehalten wurden, also ob der Code der CLS entspricht
- Base Class Library
  - Existiert für jede .NET Sprache
  - Beinhaltet Behandlung von I/O, Threads, Grafik, Hardware-Schnittstellen





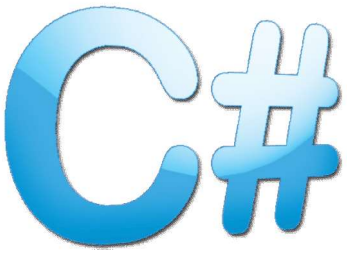
Wie behält man die Übersicht bei vielen verschiedenen Klassen ?

Namespaces

- Zur Organisation aller Typen der .NET Platform über alle Sprachen hinweg
- Namespaces können subspaces (Unterräume) enthalten
  - System, System.IO, …
- Jedes Assembly kann beliebig viele Namespaces enthalten
- Bsp: System namespace enthält System.Int32, System.String, …
- Zugriff auf Typen durch
  - Namespace.Subnamespace.Subnamespace.Typname
  - using keyword: using Namespace.Subnamespace.Subnamespace

- Zugriff auf Typen durch
  - Namespace.Subnamespace.Subnamespace.Typname
  - using keyword: using Namespace.Subnamespace.Subnamespace Typname

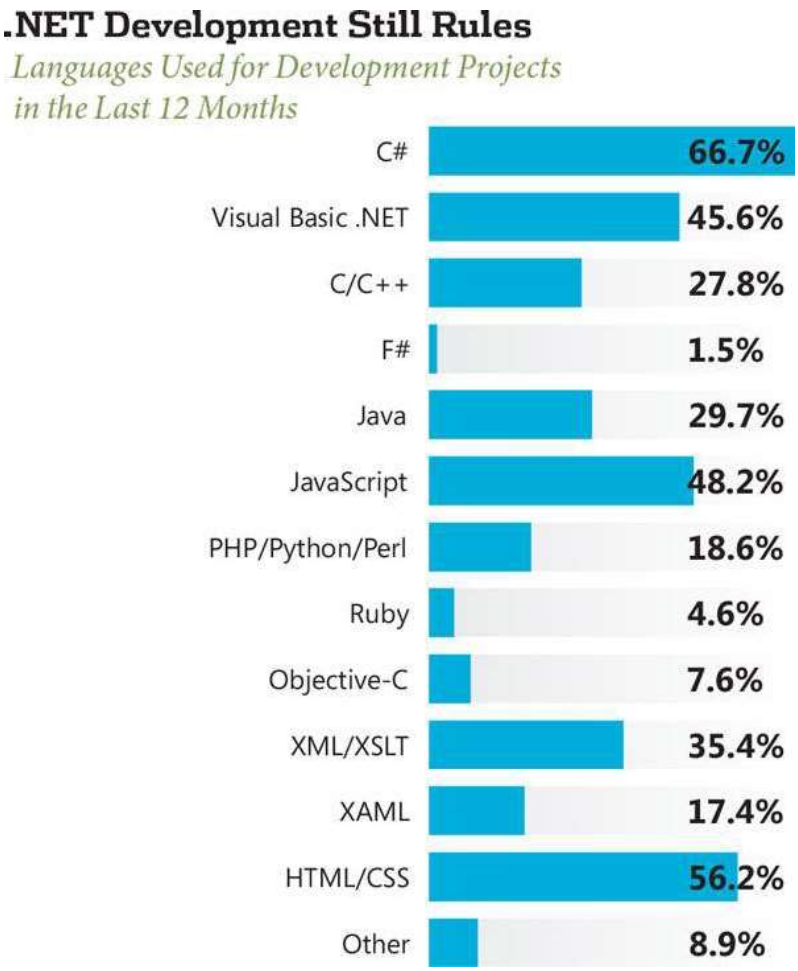
Die Sprache C#



- Syntax sehr ähnlich zu Java
- Vereint Konzepte von Java, Visual Basic, C++
- Idee: die Vorteile aller Sprachen vereinigen
  - Syntaktisch sauber wie Java
  - Einfach wie Visual Basic
  - Flexibel und Mächtig wie C++

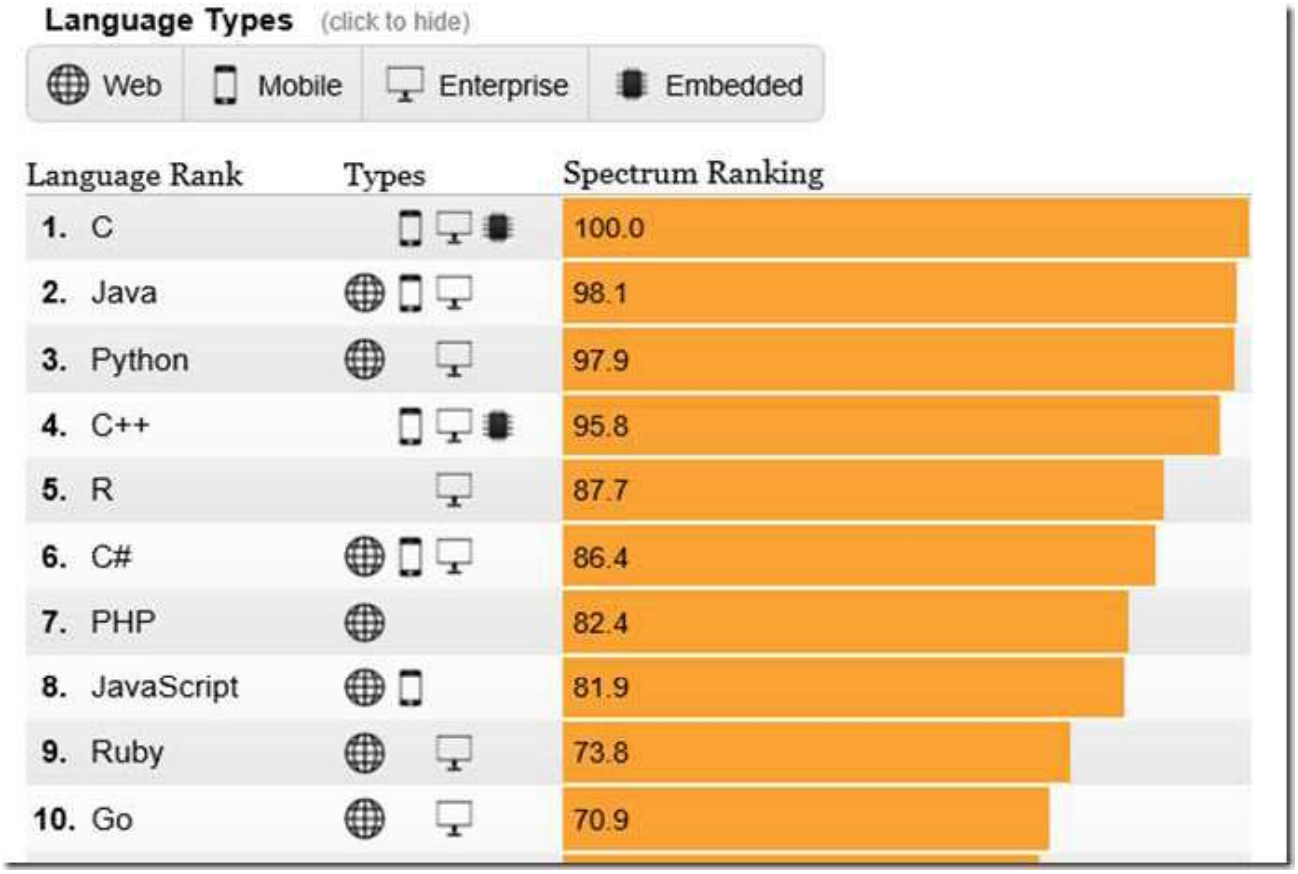
Features

- Keine Pointer nötig
- Automatisches Speichermanagement (Garbage Collector)
- Operatorüberladung
- Attribut-basierte Programmierung
- Generische Programmierung
- Lambda Operatoren
- Einfaches Multithreading mit Async/Await Keywords



Aus <https://www.google.de/search?q=c%23+statistics&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjtiJ30ocnSAhWsIsAKHRvXAKAQ_AUIBigB&biw=1920&bih=918>

Most popular coding languages



Aus <https://www.google.de/search?q=c%23+f%C3%BCr+windows+anwendungen+statistik&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjsr9fgusnSAhXCFsAKHRSBC1wQ_AUICSgC&biw=1920&bih=918>

.NET Core ist eine freie und quelloffene Software-Plattform (innerhalb) der .NET-Plattform

- 2016 veröffentlicht
- Plattform-unabhängig!
- Abgespeckte Version von .NET
- Fast alles was wir hier lernen kann direkt in .NET Core verwendet werden

Gliederung des Kurses

- **Grundlagen C#**
- **Vererbung, Polymorphie, Schnittstellen**
- **Generische Programmierung**

- **Fehlerbehandlung Exceptions**
- **Delegates und Ereignisse**
- **Windows Forms**
- **Parallele Programmierung**
- **Test-basierte Entwicklung**
- **Agile / emergente Entwicklung**

Saxonia Systems