Vorlesung

Anwendungsprogrammierung im CAx-Umfeld

Prof Dr.-Ing. Frank Lobeck

UNIVERSITÄT DUISBURG ESSEN

Universität Duisburg Essen

Campus Duisburg

- https://join.me/vip141111

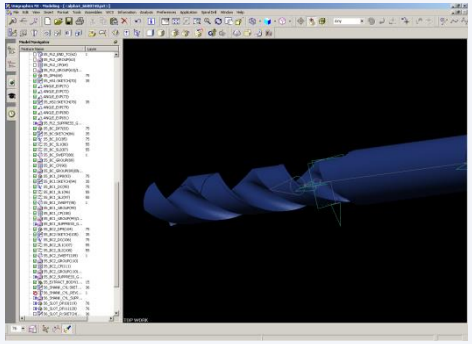
Inhalte

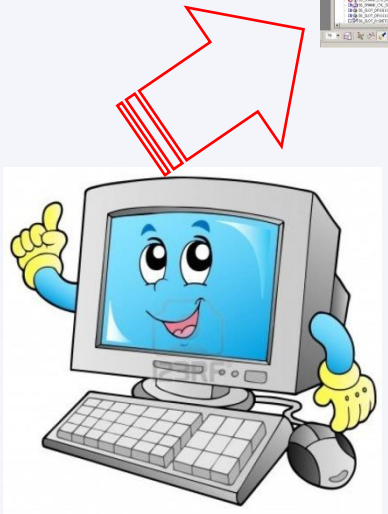
- Einführung
- Grundbegriffe der Objektorientierten Programmierung
- Grundlagen der Programmierung in Visual Basic
- Makro-Technik
- Einbinden von Makros in Visual Studio
- Anwendungsprogrammierung in SolidWorks
- Programmierung in C++ / C#
- Web-Programmierung

Einleitung

Computer – Programme – Programmiersprachen….

- Computer: „black box“, Duales System
- Programm: Tätigkeiten, Aufgaben, Prozesse





Programmiersprache
(von Menschen lesbar)

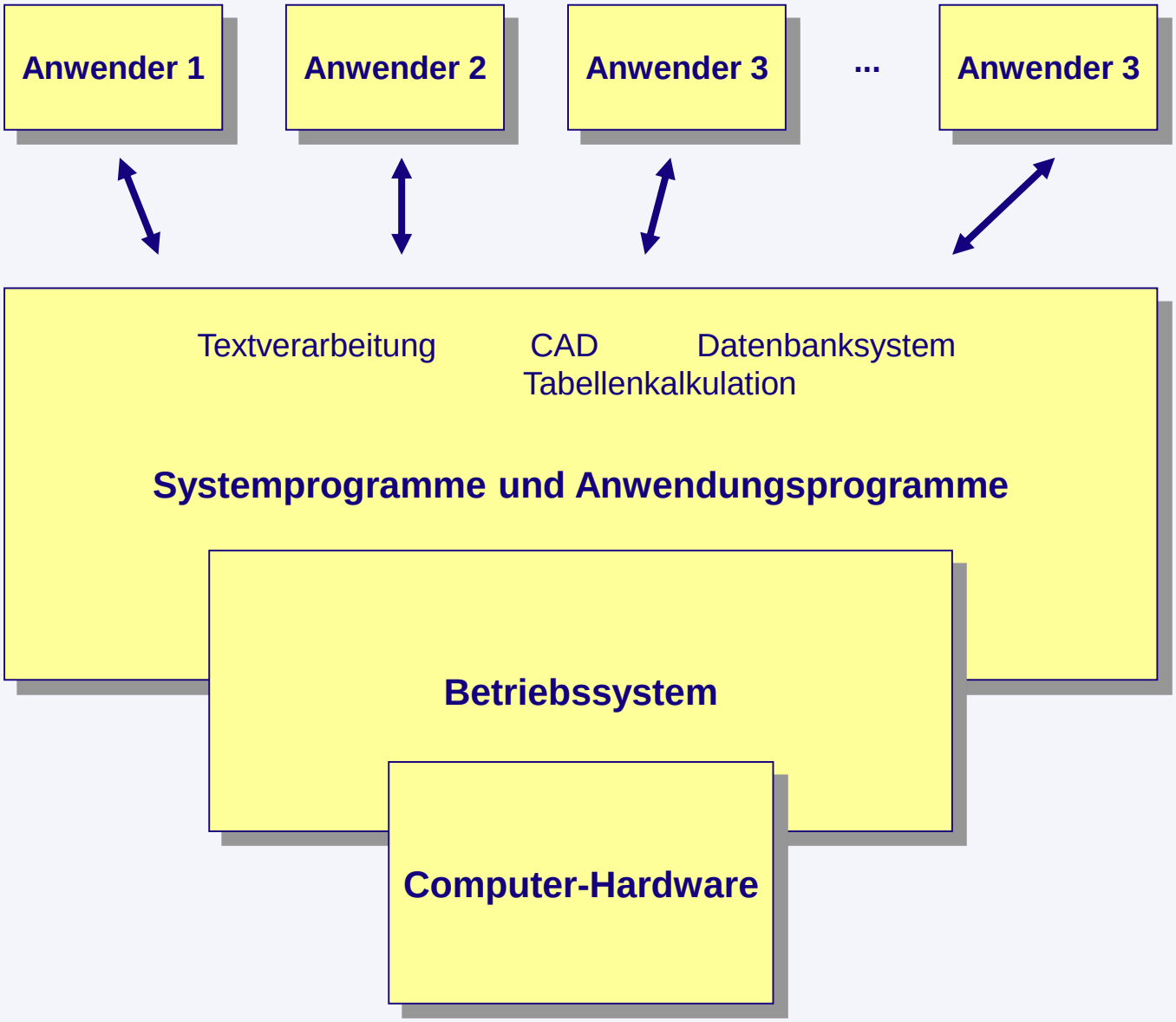
Maschinensprache
(vom Computer lesbar)

Betriebssystem - Begriffsbestimmung

Definition nach DIN 44300:

Betriebssystem (operating system): Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen. Eine Sprache, der ein Betriebssystem gehorcht, heißt Betriebssprache (operating language).

Betriebssystem - Begriffsbestimmung



Betriebssysteme - Klasseneinteilung

Ein-Benutzer	Mehrere	Mehr-Prozessor-Systeme	Verteilte Systeme	Echtzeitsysteme
(single user) Ein-Prozeß (single process)	•Benutzer (multi-user) •Prozesse (multi-processing)			
⇒ Stapelorientiert ⇒ Dialogfähig	⇒ Time sharing ⇒ multi-tasking			

Programmiersprachen

Compiler

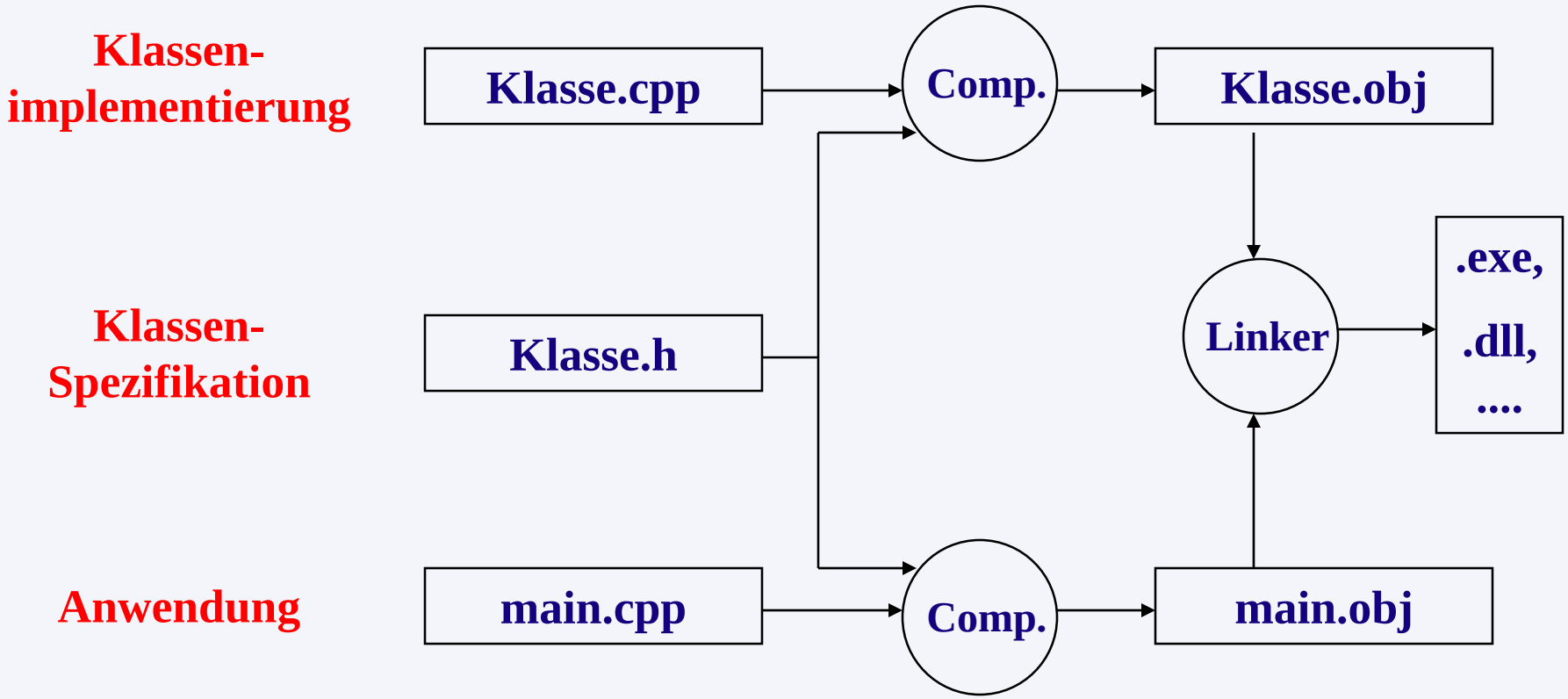
Ein Compiler ist ein Computerprogramm, das ein (anderes) Programm, das in einer bestimmten Programmiersprache geschrieben ist, in eine Form übersetzt, die von einem Computer ausgeführt werden kann. (Wikipedia)

Beispiele für Programmiersprachen

- C
- C++
- Fortran
- Pascal
- Smalltalk

Compiler – Arbeitsweise (Bsp: C++)

Vorgänge beim Erstellen einer Anwendung



Interpreter

Anders als beim Compiler übersetzt der Interpreter nicht vorher, sondern während des Programms den Quelltext. Deswegen ist die Ausführungszeit beim Interpreter deutlich langsamer. Der Interpreter erzeugt keinen speicherbaren Maschinencode.

Beispiele für Programmiersprachen mit Interpreter

- Basic
- PHP
- Perl
- Ruby
- Python
- JavaScript

Andere Formen / Mischformen

Java

Java Programme liegen zunächst als menschenverständlicher Text vor, als sogenannter Quellcode.

Der Java-Compiler übersetzt ihn in einen maschinenverständlichen Code, den sogenannten Java-Bytecode.

Die Maschine, die diesen Bytecode ausführt, ist jedoch typischerweise virtuell – das heißt, der Code wird nicht direkt durch Hardware ausgeführt, sondern durch entsprechende Software auf der Zielplattform

(die sog. Java VM (Java Virtual Machine)

.NET

(sprich: Dot Net)

.NET ist eine von Microsoft entwickelte Plattform Technologie.

.NET besteht aus einer Laufzeitumgebung (CLR, Common Language Runtime), in der die Programme ausgeführt werden.

.NET ist auf verschiedenen Plattformen verfügbar und unterstützt die Verwendung einer Vielzahl von Programmiersprachen.

.NET-Programme werden zum Kompilierungszeitpunkt zunächst in eine Zwischensprache (Common Intermediate Language) übersetzt.

Zur Ausführungszeit wird der Code von der .Net-Laufzeitumgebung in die eigentliche Maschinensprache des Zielsystems übersetzt.

Programmiersprachen:

- VB
- C# (C Sharp)

Merkmale von Visual Basic

Keine Zeilennummerierung

Wie bei modernen Hochsprachen werden die Codezeilen nicht durchnummeriert, eine Voraussetzung für prozedurales Programmieren. Um Sprungbefehle einzusetzen, können Zeilenmarken vergeben werden

Funktionen und Prozeduren

Durch Verwendung von selbstdefinierten Unterprogrammen (Routinen) können Programme modular aufgebaut werden. Solche Routinen (Funktionen und Prozeduren) lassen sich in eigenständigen Dateien zusammenfassen (Bibliotheken).

Objekte mit Eigenschaften und Methoden

Grundlegende Vorgänge werden nicht mehr über Routinen angesprochen, sondern als Objekte mit Eigenschaften und Methoden in den Programmcode eingesetzt.

Grafische Benutzeroberfläche mit ereignisgesteuerter Programmierung

Das Programm setzt sich aus grafischen Fensterelementen (Formen) und darin befindlichen Bedienungselementen (Steuerelemente oder Objekte) zusammen, die eine Benutzeroberfläche erzeugen und wartet auf Aktionen des Anwenders. Eine solche Aktion löst im Programm ein sogenanntes Ereignis aus. Innerhalb einer Ereignisprozedur kann mit linearer Programmierung die gewünschte Aktion ausgeführt werden.

Begriffe

Objekt

Objekte definieren sich durch Eigenschaften und auf sie anwendbare Methoden. VB kennt hier Anwendungen, Fenster, Steuerelemente, Geräte, etc.

Eigenschaften

Eigenschaften definieren bestimmte Zustände (.Color) eines Objektes oder sein Verhalten (.Enabled). Sie können gelesen oder gesetzt werden.

Methoden

Methoden bewirken eine Veränderung des Objektzustandes, sie funktionieren nur mit einem (bestimmten) Objekt-Typ. Methoden sind programmtechnisch innerhalb des Objektes abgelegte Prozeduren.

Anweisungen

Zur internen Steuerung des Programmablaufs: Kontroll- und Schleifenstrukturen, Wertzuweisung, Verbindung zu Hardware-Elementen wie Drucker, Laufwerke, etc.

Beispiele: **Objekt Haus: Eigenschaften: Größe, Farbe, Fenster, ...; Methoden: Ändern, Anmalen, Öffnen, Wechseln**

Objekt Auto: Eigenschaften: Farbe, Größe, Räder, Motor,...; Methoden: Lackieren, Öffnen, Starten, Ändern

Verwendung von Objekten und Eigenschaften

Objektverwendung

Haus.Farbe = rot	**Lies:** ***setze die Eigenschaft Farbe des Objektes Haus auf rot***
Auto.Farbe = rot	**Lies:** ***setze die Eigenschaft Farbe des Objektes Auto auf rot***
Haus.Räder = 4	**Lies:** ***setze die Eigenschaft Anzahl der Räder des Objektes Auto auf 4 => Fehlermeldung***

Objektverkettung

Haus.Fenster.Farbe = rot	***Das Objekt Haus.Fenster bekommt die Eigenschaft rot***
Haus.Fenster.Griff.Farbe = rot	***Das Objekt Haus.Fenster.Griff bekommt die Eigenschaft rot***

Verwendung von Objekten und Methoden

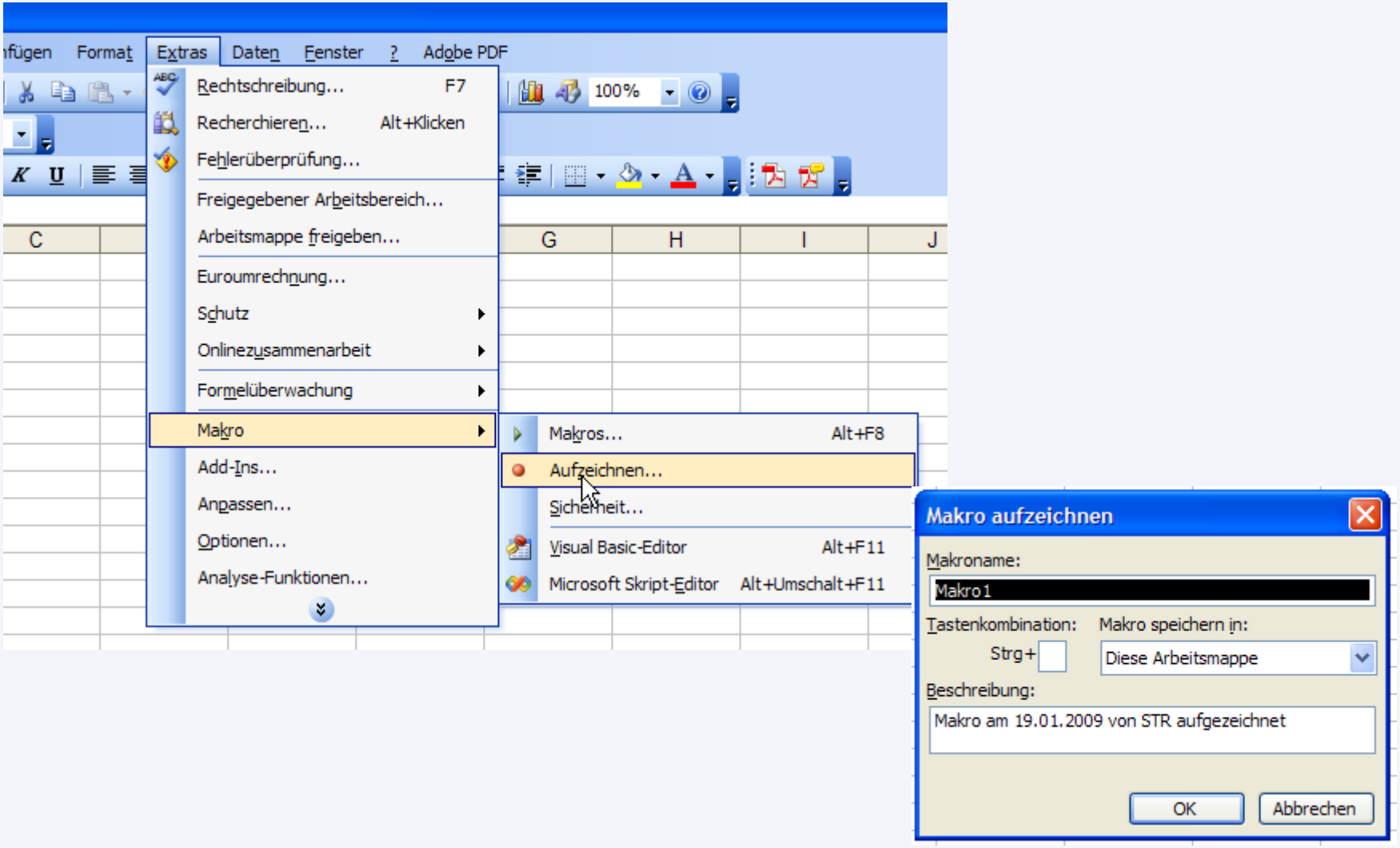
Haus.Anmalen ***Mit dem Objekt Haus wird über die Methode Anmalen eine Aktion ausgeführt***
=> Ergebnis der Methode Anmalen ist identisch mit Setzen der Eigenschaft .Farbe

Makro- Programmierung (Excel)

- Grundlagen der Programmierung in Visual Basic 6.0
- Erstellung von Makros
- Einbinden von Makros in Visual Studio

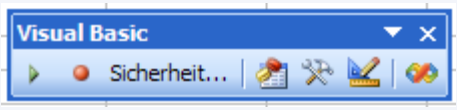
VBA-Makros in Excel

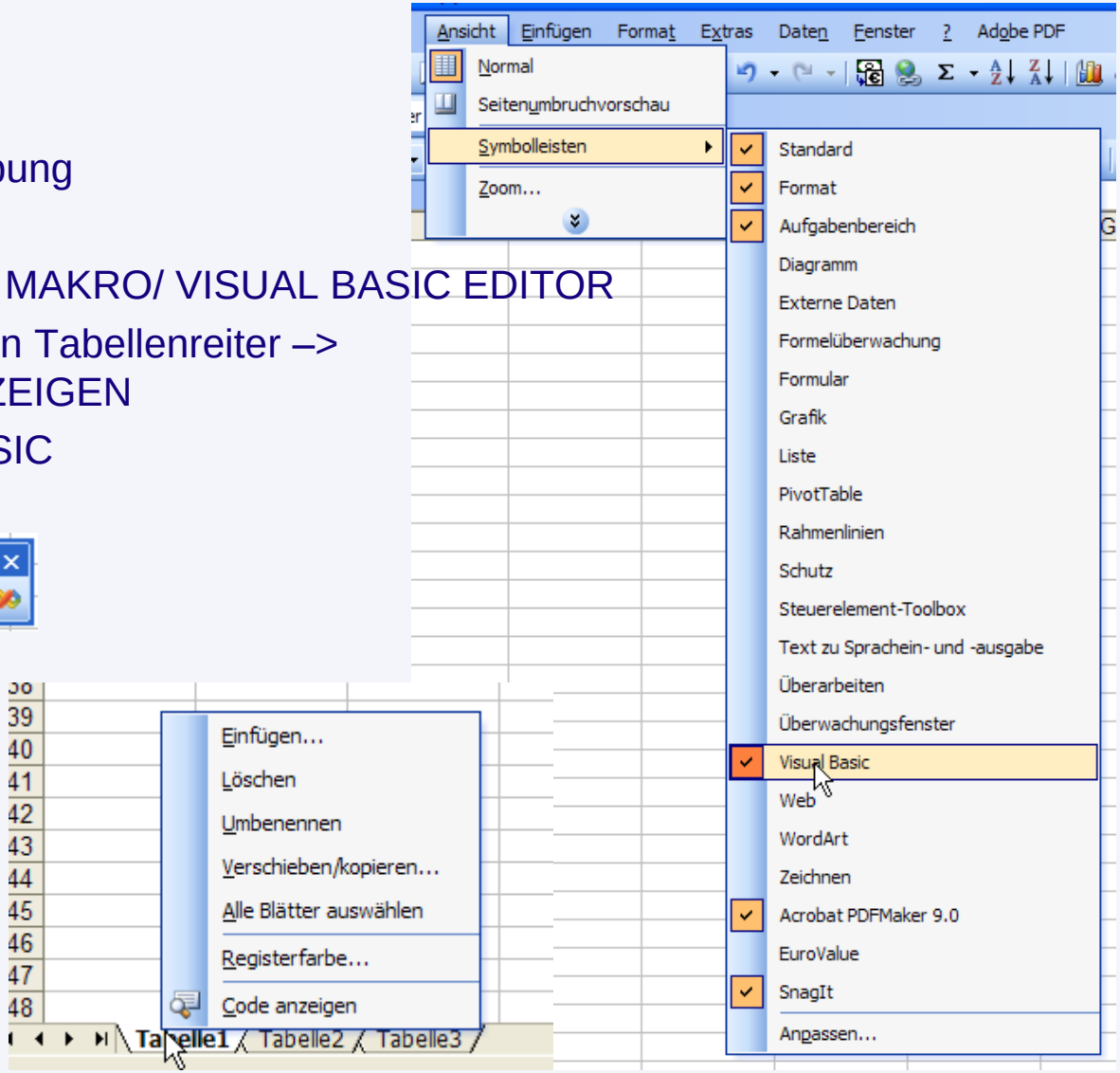
- VBA-Makros aufzeichnen



Makros editieren / Entwicklungsumgebung

- Aufruf der Entwicklungsumgebung
  - ⇒ ALT – F11
  - ⇒ Menü: EXTRAS – Befehl: MAKRO/ VISUAL BASIC EDITOR
  - ⇒ Rechte-Maus-Klick auf den Tabellenreiter –> Kontextmenü: CODE ANZEIGEN
  - ⇒ Symbolleiste VISUAL BASIC



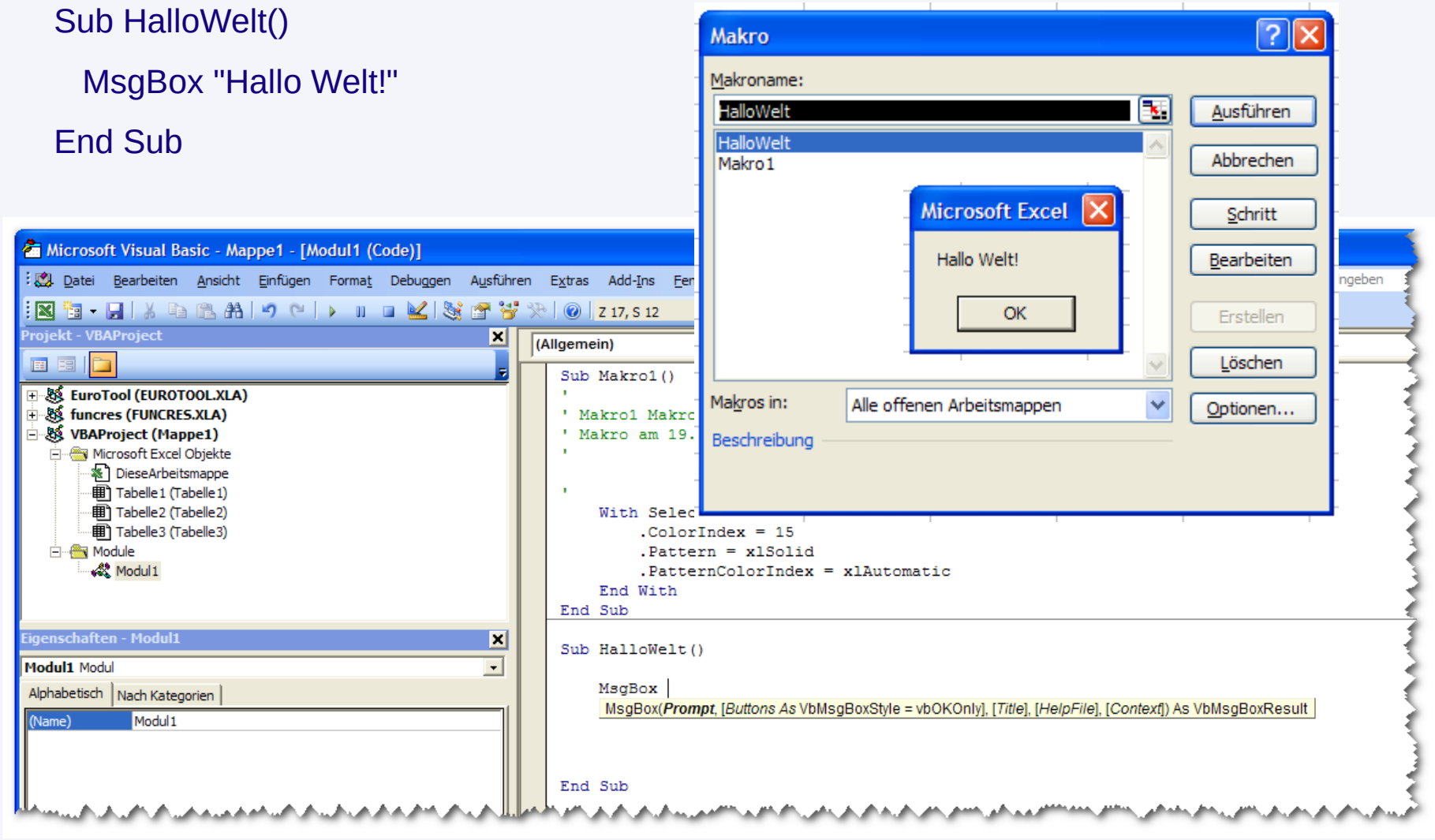


Das erste Makro: „Hallo-Welt“

ALT-F11:

```
Sub HalloWelt()
   MsgBox "Hallo Welt!"
End Sub
```

Ausführen: ALT-F8



MsgBox / InputBox

```
Msgbox
```

```
MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult
```

```
Inputbox
```

```
InputBox(Prompt, [Title], [Default], [XPos], [YPos], [HelpFile], [Context]) As String
```

Verschiedene Konstanten

Siehe auch Zusatzinfo

Die folgenden Konstanten sind in der Klassenbibliothek von Visual Basic für Applikationen definiert und können überall im Code anstelle der tatsächlichen Werte verwendet werden:

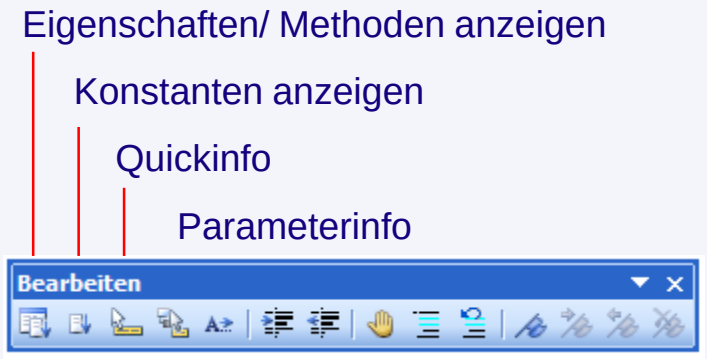
Konstante	Äquivalent	Beschreibung
vbCrLf	**Chr**(13) + **Chr**(10)	Kombination aus Wagenrücklauf und Zeilenvorschub
vbCr	**Chr**(13)	Wagenrücklaufzeichen
vbLf	**Chr**(10)	Zeilenvorschubzeichen
vbNewLine	**Chr**(13) + **Chr**(10) oder, auf dem Macintosh, **Chr**(13)	Plattformspezifisches Zeilenumbruchzeichen; je nachdem, welches für die aktuelle Plattform geeignet ist
vbNullChar	**Chr**(0)	Zeichen mit dem Wert 0
vbNullString	Zeichenfolge mit dem Wert 0	Nicht identisch mit der Null-Zeichenfolge (""); wird verwendet, um externe Prozeduren aufzurufen.
vbObjectError	-2147221504	Benutzerdefinierte Fehlernummern sollten größer als dieser Wert sein. Zum Beispiel: `Err.Raise Number = vbObjectError + 1000`
vbTab	**Chr**(9)	Tabulatorzeichen
vbBack	**Chr**(8)	Rückschrittzeichen
vbFormFeed	**Chr**(12)	Nicht sinnvoll unter Microsoft Windows oder auf dem Macintosh
vbVerticalTab	**Chr**(11)	Nicht sinnvoll unter Microsoft Windows oder auf dem Macintosh

Hilfen

- F1: Hilfe
- F2: Objektkatalog

- IntelliSense
  Durch Betätigung der Tastenkombination STRG + LEERTASTE wird die Intellisense – Autovervollständigung ausgelöst (wenn eine solche an dieser Stelle zur Verfügung steht!)

- Symbolleiste BEARBEITEN



F1-Hilfe: „Datentypen Zusammenfassung“

Datentypen (Zusammenfassung)

Siehe auch Zusatzinfo

Die folgende Tabelle enthält die von Visual Basic unterstützten Datentypen sowie deren Speicherbedarf und Wertebereiche.

Datentyp	Speicherbedarf	Wertebereich
Byte	1 Byte	0 bis 255
Boolean	2 Bytes	**True** oder **False**
Integer	2 Bytes	-32.768 bis 32.767
Long (lange Ganzzahl)	4 Bytes	-2.147.483.648 bis 2.147.483.647
Single (Gleitkommazahl mit einfacher Genauigkeit)	4 Bytes	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte.
Double (Gleitkommazahl mit doppelter Genauigkeit)	8 Bytes	-1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte.
Currency (skalierte Ganzzahl)	8 Bytes	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807
Decimal	14 Bytes	+/-79.228.162.514.264.337.593.543.950.335 ohne Dezimalzeichen; +/-7,9228162514264337593543950335 mit 28 Nachkommastellen; die kleinste Zahl ungleich Null ist +/-0,0000000000000000000000000001.
Date	8 Bytes	1. Januar 100 bis 31. Dezember 9999.
Object	4 Bytes	Beliebiger Verweis auf ein Objekt vom Typ **Object**.
String (variable Länge)	10 Bytes plus Zeichenfolgenlänge	0 bis ca. 2 Milliarden.
String (feste Länge)	Zeichenfolgenlänge	1 bis ca. 65.400
Variant (mit Zahlen)	16 Bytes	Numerische Werte im Bereich des Datentyps **Double**.
Variant (mit Zeichen)	22 Bytes plus Zeichenfolgenlänge	Wie bei **String** mit variabler Länge.
Benutzerdefiniert (mit **Type**)	Zahl ist von Elementen abhängig	Der Bereich für jedes Element entspricht dem Bereich des zugehörigen Datentyps.

Variablen definieren

Option Explicit-Anweisung

Siehe auch Beispiel Zusatzinfo

Wird auf Modulebene verwendet, um die explizite Deklaration aller Variablen in diesem Modul zu erzwingen.

Syntax

Option Explicit

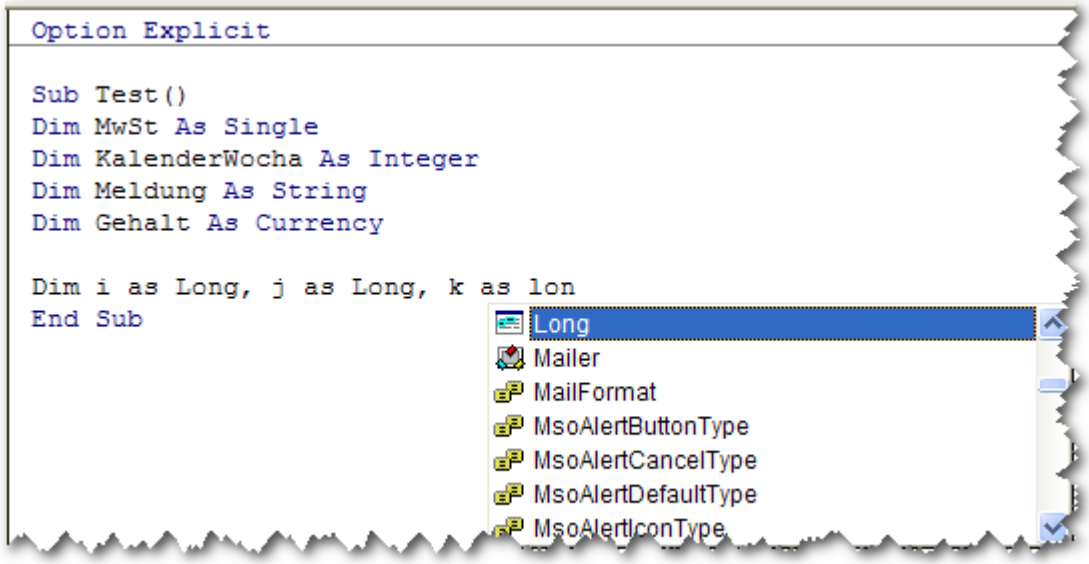
Bemerkungen

Wird die **Option Explicit**-Anweisung verwendet, so muß sie im jeweiligen Modul vor jeder Prozedur stehen.

Wenn Sie die **Option Explicit**-Anweisung in einem Modul verwenden, müssen alle Variablen explizit mit den Anweisungen **Dim**, **Private**, **Public**, **ReDim** oder **Static** deklariert werden. Wenn Sie einen nicht deklarierten Variablennamen verwenden, tritt zur Kompilierungszeit ein Fehler auf.

Wenn Sie die **Option Explicit**-Anweisung nicht verwenden, erhalten alle nichtdeklarierten Variablen den Typ **Variant**, solange mit einer **Def***Typ*-Anweisung kein anderer Standardtyp festgelegt wird.

Anmerkung Mit **Option Explicit** vermeiden Sie falsch geschriebene Namen bereits bestehender Variablen oder Verwechslungen im Code bei unklarem Gültigkeitsbereich von Variablen.



Fallunterscheidung: if

If...Then...Else-Anweisung

Siehe auch Beispiel Zusatzinfo

Führt eine Gruppe von Anweisungen aus, wenn bestimmte Bedingungen erfüllt sind, die vom Wert eines Ausdrucks abhängen.

Syntax

If *Bedingung* **Then** [*Anweisungen*] [**Else** *elseAnweisungen*]

Alternativ können Sie die Block-Syntax verwenden:

If *Bedingung* **Then**
    [*Anweisungen*]
[**ElseIf** *Bedingung-n* **Then**
    [*elseifAnweisungen*] . . .
[**Else**
    [*elseAnweisungen*]]
End If

Teil	Beschreibung
Bedingung	Erforderlich. Ein oder mehrere Ausdrücke der beiden folgenden Arten: Ein numerischer Ausdruck oder ein Zeichenfolgenausdruck, der **True** oder **False ergibt**. Wenn *Bedingung* den Wert Null hat, wird *Bedingung* als **False** interpretiert. Ein Ausdruck der Form **TypeOf** *Objektname* **Is** *Objekttyp*. *Objektname* ist ein beliebiger Objektverweis, und *Objekttyp* ist ein beliebiger gültiger Objekttyp. Der Ausdruck ergibt **True**, wenn *Objektname* dem Objekttyp entspricht, der durch *Objekttyp* angegeben wird. Andernfalls ist das Ergebnis **False**.
Anweisungen	Optional in Form eines Blocks, erforderlich in der einzeiligen Variante, die keinen **Else**-Abschnitt beinhaltet. Eine oder mehrere durch Doppelpunkte getrennte Anweisungen, die ausgeführt werden, wenn *Bedingung* den Wert **True** hat.
Anweisung-n	Optional. Dieselbe Bedeutung wie *Bedingung*.
elseifAnweisungen	Optional. Eine oder mehrere Anweisungen, die ausgeführt werden, wenn die zugehörige Bedingung *(Bedingung-n)* **True ergibt**.
elseAnweisungen	Optional. Eine oder mehrere Anweisungen, die ausgeführt werden, wenn keine der Bedingungen (*Bedingung*-Ausdruck oder *Bedingung-n*-Ausdruck) **True** ergibt.

Fallunterscheidung: select – case

Select Case-Anweisung

Siehe auch Beispiel Zusatzinfo

Führt eine von mehreren Gruppen von Anweisungen aus, abhängig vom Wert eines Ausdrucks.

Syntax

Select Case *Testausdruck*
    [**Case** *Ausdrucksliste-n*
    [*Anweisungen-n*]] . . .
    [**Case Else**
    [*elseAnw*]]
End Select

Die Syntax der **Select Case**-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Testausdruck	Erforderlich. Ein beliebiger numerischer Ausdruck oder Zeichenfolgenausdruck.
Ausdrucksliste-n	Erforderlich, wenn der **Case**-Abschnitt verwendet wird. Eine durch Kommas getrennte Liste in einer oder mehreren der folgenden Formen: *Ausdruck*, *Ausdruck* **To** *Ausdruck*, **Is** *Vergleichsoperator Ausdruck*. Das Schlüsselwort **To** gibt einen Bereich von Werten an. Bei diesem Schlüsselwort muß der kleinere Wert immer links von **To** stehen. Verwenden Sie das Schlüsselwort **Is** in Kombination mit Vergleichsoperatoren (außer **Is** und **Like**), um einen Bereich von Werten anzugeben. Wenn Sie das Schlüsselwort **Is** nicht angeben, wird es automatisch eingefügt.
Anweisungen-n	Optional. Eine oder mehrere Anweisungen, die ausgeführt werden, wenn *Testausdruck* mit irgendeinem Teil in *Ausdrucksliste-n* übereinstimmt.
elseAnw	Optional. Eine oder mehrere Anweisungen, die ausgeführt werden, wenn *Testausdruck* mit keinem der Ausdrücke im **Case**-Abschnitt übereinstimmt.

Schleifen: Do - Loop

Do...Loop-Anweisung

Siehe auch Beispiel Zusatzinfo

Wiederholt einen Block mit Anweisungen, solange eine Bedingung den Wert **True** hat oder bis eine Bedingung den Wert **True** erhält.

Syntax

Do [{**While** | **Until**} *Bedingung*]
    [*Anweisungen*]
    [**Exit Do**]
    [*Anweisungen*]

Loop

Sie können auch die folgende, ebenfalls zulässige Syntax verwenden:

Do
    [*Anweisungen*]
    [**Exit Do**]
    [*Anweisungen*]

Loop [{**While** | **Until**} *Bedingung*]

Die Syntax für die **Do Loop**-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Bedingung	Optional. Numerischer Ausdruck oder Zeichenfolgenausdruck, der entweder **True** oder **False** ergibt. Hat *Bedingung* den Wert Null, so wird *Bedingung* als **False** interpretiert.
Anweisungen	Eine oder mehrere Anweisungen, die wiederholt werden, solange oder bis *Bedingung* **True** wird.

Schleifen: For - Next

For...Next-Anweisung

Siehe auch Beispiel Zusatzinfo

Wiederholt eine Gruppe von Anweisungen so oft wie angegeben.

Syntax

For *Zähler* = *Anfang* **To** *Ende* [**Step** *Schritt*]
    [*Anweisungen*]
    [**Exit For**]
    [*Anweisungen*]

Next [*Zähler*]

Die Syntax für die **For...Next**-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Zähler	Erforderlich. Numerische Variable, die als Schleifenzähler dient. Eine boolesche Variable oder ein Element eines Datenfeldes ist nicht zulässig.
Anfang	Erforderlich. Startwert von *Zähler*.
Ende	Erforderlich. Endwert von *Zähler*.
Schritt	Optional. Betrag, um den *Zähler* bei jedem Schleifendurchlauf verändert wird. Falls kein Wert angegeben wird, ist die Voreinstellung für *Schritt* eins.
Anweisungen	Optional. Eine oder mehrere Anweisungen zwischen **For** und **Next**, die so oft wie angegeben ausgeführt werden.

Schleifen: While - Wend

While...Wend-Anweisung

Siehe auch Beispiel Zusatzinfo

Führt eine Reihe von Anweisungen aus, solange eine gegebene Bedingung den Wert **True** hat.

Syntax

While *Bedingung*
[*Anweisungen*]
Wend

Die Syntax der **While...Wend**-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Bedingung	Erforderlich. Numerischer Ausdruck oder Zeichenfolgenausdruck, der **True** oder **False** ergibt. Wenn *Bedingung* den Wert Null hat, wird *Bedingung* als **False** interpretiert.
Anweisungen	Optional. Eine oder mehrere Anweisungen, die ausgeführt werden, solange *Bedingung* **True** ergibt.

```
While Not ActiveSheet.Range("A" & i) = ""
    Select Case ActiveSheet.Range("C" & i)
     Case "OK"
        Range("A" & i, "I" & i).Select
    Case Else
     MsgBox "Es fehlt der Wert in Zeile " & i
   End Select
i = i + 1
Wend
```

With - Anweisung

With-Anweisung

Siehe auch Beispiel Zusatzinfo

Führt eine Reihe von Anweisungen für ein einzelnes Objekt oder einen benutzerdefinierten Typ aus.

Syntax

With *Objekt*
    [*Anweisungen*]
End With

Die Syntax der **With**-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Objekt	Erforderlich. Name eines Objekts oder eines benutzerdefinierten Typs.
Anweisungen	Optional. Eine oder mehrere Anweisungen, die für *Objekt* ausgeführt werden sollen.

Bemerkungen

Mit der **With**-Anweisung können Sie eine Reihe von Anweisungen für ein bestimmtes Objekt ausführen, ohne den Namen des Objekts mehrmals angeben zu müssen. Wenn Sie zum Beispiel mehrere Eigenschaften eines bestimmten Objekts verändern möchten, sollten Sie die Zuweisungsanweisungen für die Eigenschaft in einer **With**-Kontrollstruktur unterbringen. Sie brauchen dann den Namen des Objekts nicht bei jeder einzelnen Zuweisung, sondern nur einmal zu Beginn der Kontrollstruktur anzugeben. Das folgende Beispiel veranschaulicht die Verwendung der **With**-Anweisung, um mehreren Eigenschaften desselben Objekts Werte zuzuweisen.

```
With Bezeichnungsfeld1
    .Height = 2000
    .Width = 2000
    .Caption = "Schönen Tag noch"
End With
```

Interior

Range.Interior

Diagrammobjekt.Interior

Das Objekt beschreibt Farbe und Muster des Innenbereichs (des Hintergrunds). Die wichtigsten Eigenschaften sind Color, Pattern und PatternColor.

Bsp.:

```
Selection.Interior.ColorIndex = 4
Selection.Interior.Pattern = xlSolid
```

Farbtabellen I

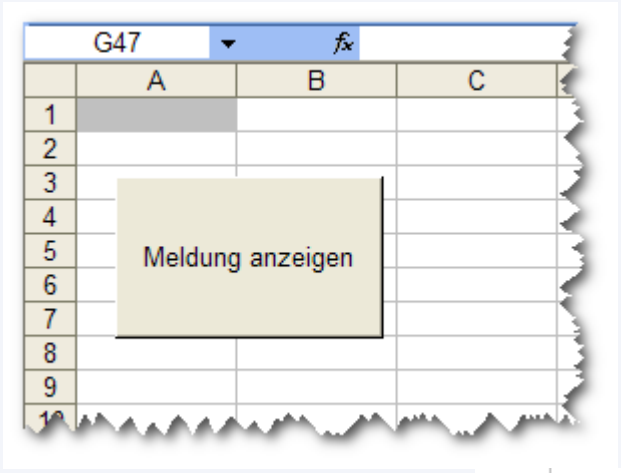
interior	font	HTML	bgcolor=	Red	Green	Blue	Color
Black	[Color 1]	#000000	#000000	0	0	0	**[Black]**
White		#FFFFFF	#FFFFFF	255	255	255	**[White]**
Red	[Color 3]	#FF0000	#FF0000	255	0	0	**[Red]**
Green	[Color 4]	#00FF00	#00FF00	0	255	0	**[Green]**
Blue	[Color 5]	#0000FF	#0000FF	0	0	255	**[Blue]**
Yellow	[Color 6]	#FFFF00	#FFFF00	255	255	0	**[Yellow]**
Magenta	[Color 7]	#FF00FF	#FF00FF	255	0	255	**[Magenta]**
Cyan	[Color 8]	#00FFFF	#00FFFF	0	255	255	**[Cyan]**
[Color 9]	[Color 9]	#800000	#800000	128	0	0	[Color 9]
[Color 10]	[Color 10]	#008000	#008000	0	128	0	[Color 10]
[Color 11]	[Color 11]	#000080	#000080	0	0	128	[Color 11]
[Color 12]	[Color 12]	#808000	#808000	128	128	0	[Color 12]
[Color 13]	[Color 13]	#800080	#800080	128	0	128	[Color 13]
[Color 14]	[Color 14]	#008080	#008080	0	128	128	[Color 14]
[Color 15]	[Color 15]	#C0C0C0	#C0C0C0	192	192	192	[Color 15]
[Color 16]	[Color 16]	#808080	#808080	128	128	128	[Color 16]
[Color 17]	[Color 17]	#9999FF	#9999FF	153	153	255	[Color 17]
[Color 18]	[Color 18]	#993366	#993366	153	51	102	[Color 18]
[Color 19]	[Color 19]	#FFFFCC	#FFFFCC	255	255	204	[Color 19]
[Color 20]	[Color 20]	#CCFFFF	#CCFFFF	204	255	255	[Color 20]
[Color 21]	[Color 21]	#660066	#660066	102	0	102	[Color 21]
[Color 22]	[Color 22]	#FF8080	#FF8080	255	128	128	[Color 22]
[Color 23]	[Color 23]	#0066CC	#0066CC	0	102	204	[Color 23]
[Color 24]	[Color 24]	#CCCCFF	#CCCCFF	204	204	255	[Color 24]
[Color 25]	[Color 25]	#000080	#000080	0	0	128	[Color 25]
[Color 26]	[Color 26]	#FF00FF	#FF00FF	255	0	255	[Color 26]
[Color 27]	[Color 27]	#FFFF00	#FFFF00	255	255	0	[Color 27]
[Color 28]	[Color 28]	#00FFFF	#00FFFF	0	255	255	[Color 28]
[Color 29]	[Color 29]	#800080	#800080	128	0	128	[Color 29]
[Color 30]	[Color 30]	#800000	#800000	128	0	0	[Color 30]
[Color 31]	[Color 31]	#008080	#008080	0	128	128	[Color 31]
[Color 32]	[Color 32]	#0000FF	#0000FF	0	0	255	[Color 32]
[Color 33]	[Color 33]	#00CCFF	#00CCFF	0	204	255	[Color 33]
[Color 34]	[Color 34]	#CCFFFF	#CCFFFF	204	255	255	[Color 34]

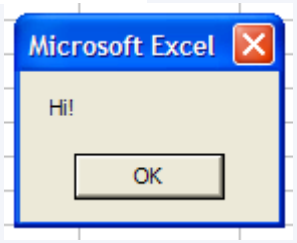
Farbtabellen II

[Color 35]	[Color 35]	#CCFFCC	#CCFFCC	204	255	204	[Color 35]
[Color 36]	[Color 36]	#FFFF99	#FFFF99	255	255	153	[Color 36]
[Color 37]	[Color 37]	#99CCFF	#99CCFF	153	204	255	[Color 37]
[Color 38]	[Color 38]	#FF99CC	#FF99CC	255	153	204	[Color 38]
[Color 39]	[Color 39]	#CC99FF	#CC99FF	204	153	255	[Color 39]
[Color 40]	[Color 40]	#FFCC99	#FFCC99	255	204	153	[Color 40]
[Color 41]	[Color 41]	#3366FF	#3366FF	51	102	255	[Color 41]
[Color 42]	[Color 42]	#33CCCC	#33CCCC	51	204	204	[Color 42]
[Color 43]	[Color 43]	#99CC00	#99CC00	153	204	0	[Color 43]
[Color 44]	[Color 44]	#FFCC00	#FFCC00	255	204	0	[Color 44]
[Color 45]	[Color 45]	#FF9900	#FF9900	255	153	0	[Color 45]
[Color 46]	[Color 46]	#FF6600	#FF6600	255	102	0	[Color 46]
[Color 47]	[Color 47]	#666699	#666699	102	102	153	[Color 47]
[Color 48]	[Color 48]	#969696	#969696	150	150	150	[Color 48]
[Color 49]	[Color 49]	#003366	#003366	0	51	102	[Color 49]
[Color 50]	[Color 50]	#339966	#339966	51	153	102	[Color 50]
[Color 51]	[Color 51]	#003300	#003300	0	51	0	[Color 51]
[Color 52]	[Color 52]	#333300	#333300	51	51	0	[Color 52]
[Color 53]	[Color 53]	#993300	#993300	153	51	0	[Color 53]
[Color 54]	[Color 54]	#993366	#993366	153	51	102	[Color 54]
[Color 55]	[Color 55]	#333399	#333399	51	51	153	[Color 55]
[Color 56]	[Color 56]	#333333	#333333	51	51	51	[Color 56]

Beispiel Schaltflächen einfügen

```
Sub Schaltflaeche()
Dim Schalter As Button
On Error Resume Next

    Sheets("Tabelle1").Buttons("Schalter").Delete
    Set Schalter = Sheets("Tabelle1").Buttons.Add(15, 30, 100, 60)
    With Schalter
        .Name = "Schalter"
        .Caption = "Meldung anzeigen"
        .OnAction = "Meldung“
    End With
    Set Schalter = Nothing

End Sub

Sub Meldung()
    MsgBox "Hi!"
    ActiveSheet.Buttons("Schalter").Delete
End Sub

Sub LoeschenButton()
   ActiveSheet.Buttons("Schalter").Delete
End Sub
```



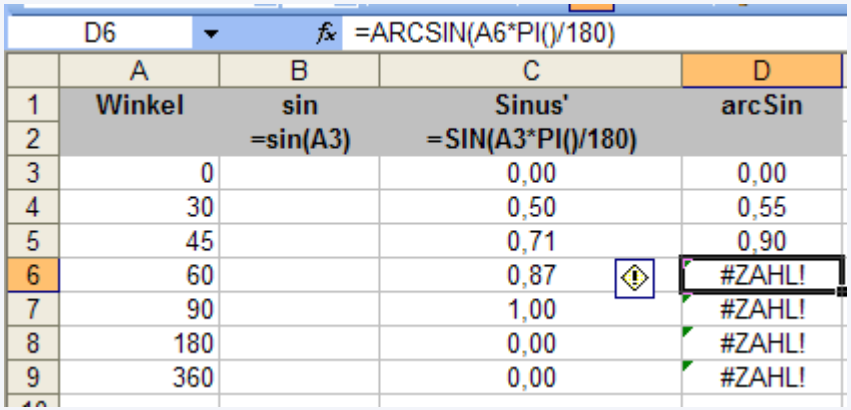


Winkelfunktionen/ Eigene Funktionen

Achtung:

Excel rechnet Winkelbezüge im Bogenmaß: 2* PI()= 360°!

=> sin‘ (90°) => sin (90° * PI()/180)



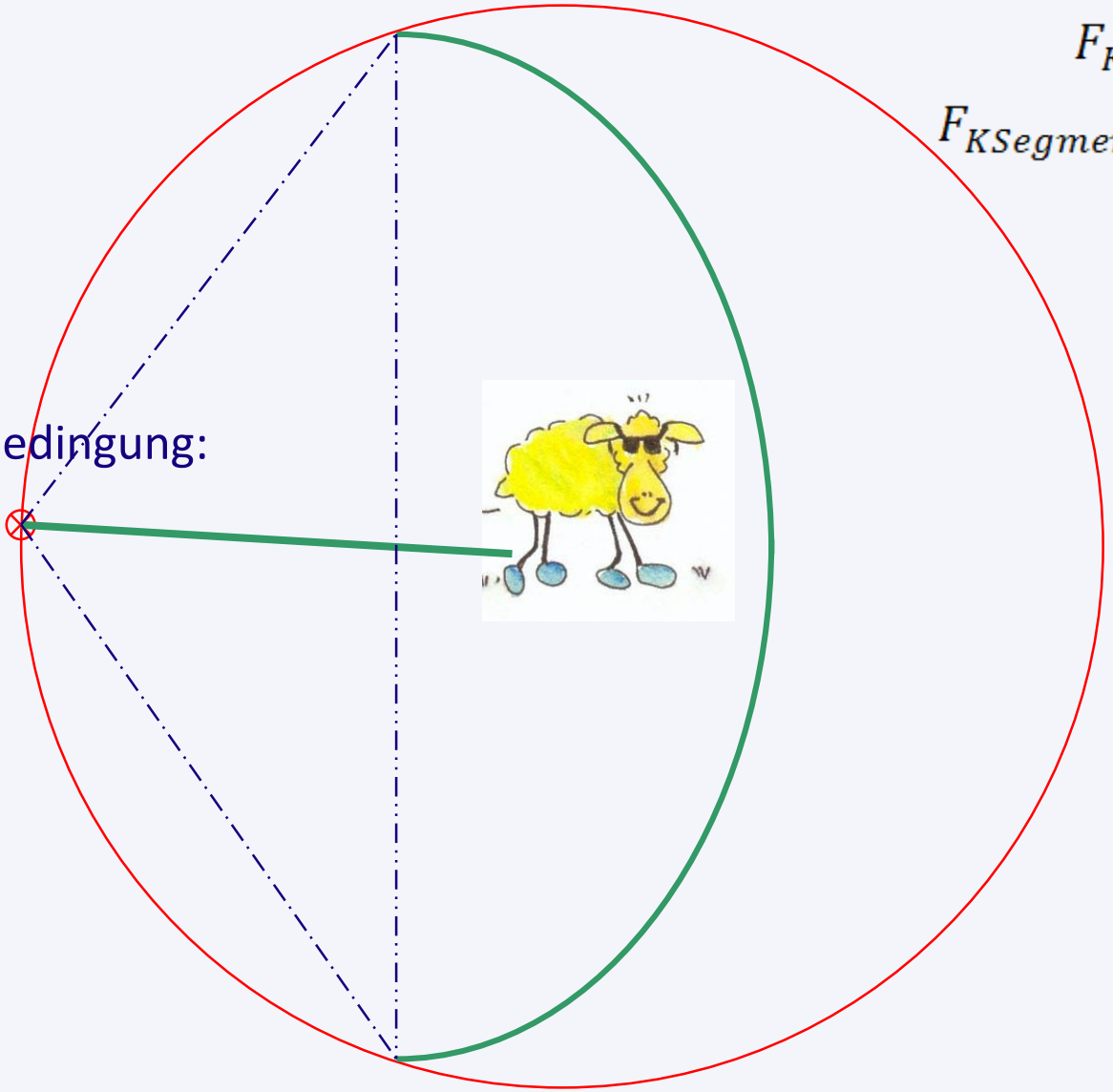
	A	B	C	D
1	Winkel	sin	Sinus'	arcSin
2		=sin(A3)	=SIN(A3*PI()/180)	
3	0		0,00	0,00
4	30		0,50	0,55
5	45		0,71	0,90
6	60		0,87	#ZAHL!
7	90		1,00	#ZAHL!
8	180		0,00	#ZAHL!
9	360		0,00	#ZAHL!

```
Function ASin(value As Double) As Double
    If Abs(value) <> 1 Then
        ASin = Atn(value / Sqr(Abs(1 - value * value)))
    Else
        ASin = 1.5707963267949 * Sgn(value)
    End If
End Function
```

„Schafrätsel“

- Wir haben eine kleine, RUNDE, schön gelegene und mit saftigem Gras bestandene Wiese. Der Besitzer hat ein kleines, weißes, wuscheliges Schaf. Wenn er es einfach auf der Wiese laufen lassen würde, hätte es bald das ganze Gras gefressen, wäre dick und rund und würde am Schluss kollabieren. Um das zu verhindern, beschließt der Bauer/Besitzer, dass es nur die Hälfte des Grases fressen soll, sprich die halbe Wiese abgrasen darf. Da das besagte Wolltier eine dementsprechende Aufforderung aber nicht verstehen würde und der Bauer auch keine Lust hat, eine Abgrenzung zu errichten, bindet er das Schaf einfach am Zaun (um die ganze Wiese verläuft ein Zaun!) fest und macht das Seil genau so lang, dass das Schaf die halbe Fläche erreichen kann.
  Frage: Wie lang ist denn das Seil, mit dem das Schaf festgebunden ist?

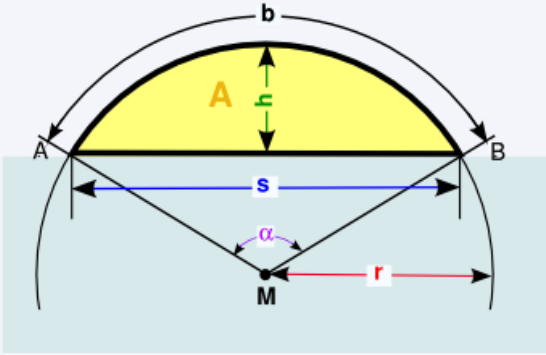
Mathematik



Bedingung:

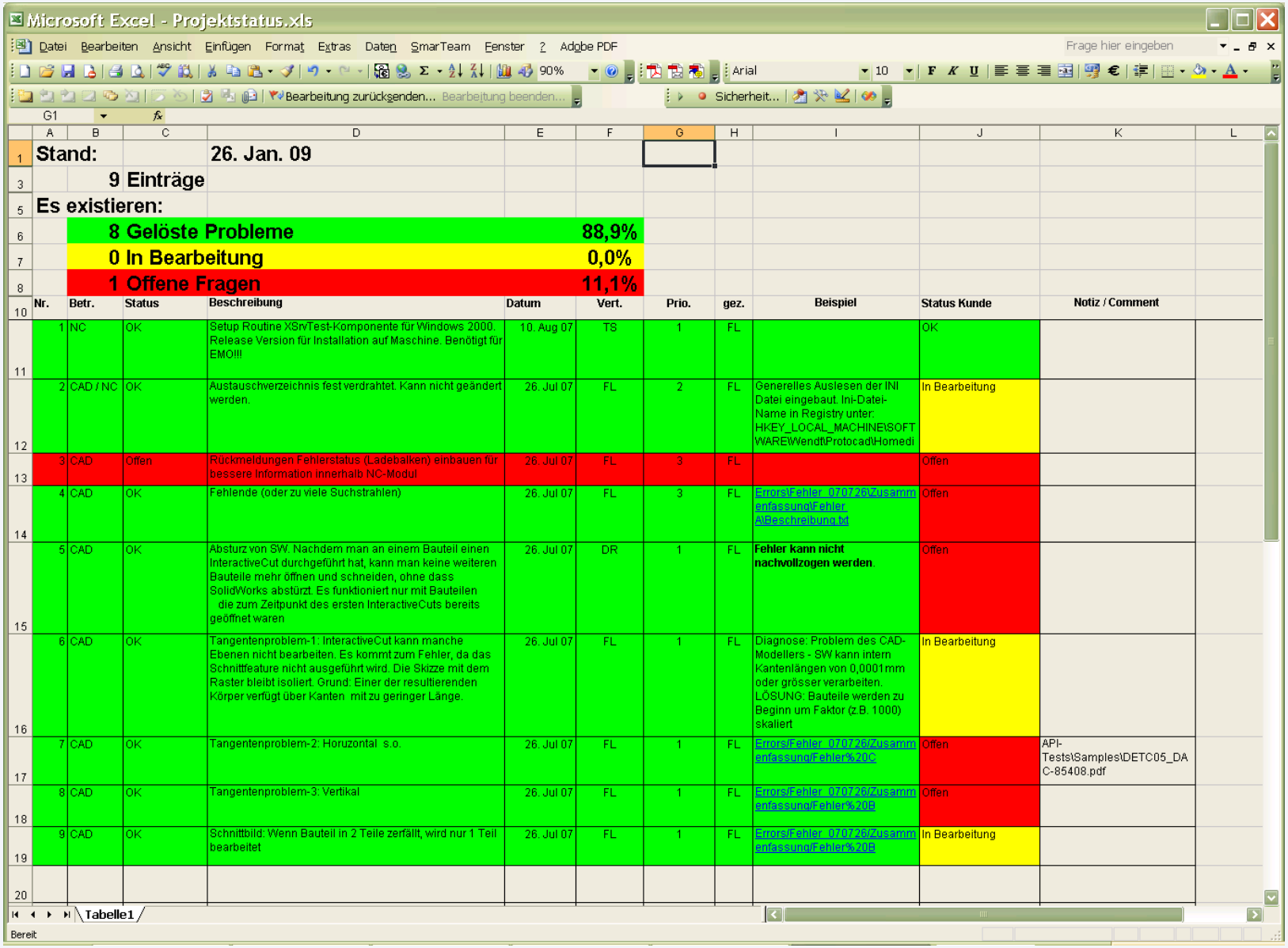
$$F_{KTeilkreis} = \pi r^2 * \frac{\alpha}{360^\circ}$$

$$F_{KSegment} = F_{Teilkreis} - 2 * F_{Dreieck}$$



$$\frac{F_{Wiese}}{2} = F_{KSegment}$$

Aufgabe: Projektmanagement



Microsoft Excel - Projektstatus.xls

Datei Bearbeiten Ansicht Einfügen Format Extras Daten SmarTeam Fenster ? Adobe PDF

Frage hier eingeben

Arial 10 90%

Bearbeitung zurücksenden... Bearbeitung beenden... Sicherheit...

G1

	A	B	C	D	E	F	G	H	I	J	K	L
1	**Stand:**			**26. Jan. 09**								
3		**9**	**Einträge**									
5	**Es existieren:**											
6		**8**	**Gelöste Probleme**			**88,9%**						
7		**0**	**In Bearbeitung**			**0,0%**						
8		**1**	**Offene Fragen**			**11,1%**						
10	**Nr.**	**Betr.**	**Status**	**Beschreibung**	**Datum**	**Vert.**	**Prio.**	**gez.**	**Beispiel**	**Status Kunde**	**Notiz / Comment**	
11	1	NC	OK	Setup Routine XSrvTest-Komponente für Windows 2000. Release Version für Installation auf Maschine. Benötigt für EMO!!!	10. Aug 07	TS	1	FL		OK		
12	2	CAD / NC	OK	Austauschverzeichnis fest verdrahtet. Kann nicht geändert werden.	26. Jul 07	FL	2	FL	Generelles Auslesen der INI Datei eingebaut. Ini-Datei-Name in Registry unter: HKEY_LOCAL_MACHINE\SOFTWAREWendt\Protocad\Homedi	In Bearbeitung		
13	3	CAD	Offen	Rückmeldungen Fehlerstatus (Ladebalken) einbauen für bessere Information innerhalb NC-Modul	26. Jul 07	FL	3	FL		Offen		
14	4	CAD	OK	Fehlende (oder zu viele Suchstrahlen)	26. Jul 07	FL	3	FL	Errors\Fehler_070726\Zusammenfassung\Fehler A\Beschreibung.txt	Offen		
15	5	CAD	OK	Absturz von SW. Nachdem man an einem Bauteil einen InteractiveCut durchgeführt hat, kann man keine weiteren Bauteile mehr öffnen und schneiden, ohne dass SolidWorks abstürzt. Es funktioniert nur mit Bauteilen die zum Zeitpunkt des ersten InteractiveCuts bereits geöffnet waren	26. Jul 07	DR	1	FL	**Fehler kann nicht nachvollzogen werden.**	Offen		
16	6	CAD	OK	Tangentenproblem-1: InteractiveCut kann manche Ebenen nicht bearbeiten. Es kommt zum Fehler, da das Schnittfeature nicht ausgeführt wird. Die Skizze mit dem Raster bleibt isoliert. Grund: Einer der resultierenden Körper verfügt über Kanten mit zu geringer Länge.	26. Jul 07	FL	1	FL	Diagnose: Problem des CAD-Modellers - SW kann intern Kantenlängen von 0,0001mm oder grösser verarbeiten. LÖSUNG: Bauteile werden zu Beginn um Faktor (z.B. 1000) skaliert	In Bearbeitung		
17	7	CAD	OK	Tangentenproblem-2: Horuzontal s.o.	26. Jul 07	FL	1	FL	Errors/Fehler_070726/Zusammenfassung/Fehler%20C	Offen	API-Tests\Samples\DETC05_DAC-85408.pdf	
18	8	CAD	OK	Tangentenproblem-3: Vertikal	26. Jul 07	FL	1	FL	Errors/Fehler_070726/Zusammenfassung/Fehler%20B	Offen		
19	9	CAD	OK	Schnittbild: Wenn Bauteil in 2 Teile zerfällt, wird nur 1 Teil bearbeitet	26. Jul 07	FL	1	FL	Errors/Fehler_070726/Zusammenfassung/Fehler%20B	In Bearbeitung		
20												

Tabelle1

Bereit

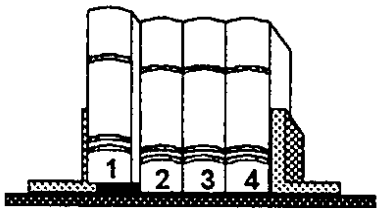
Programmierung

- Grundbegriffe der Objektorientierten Programmierung
- Grundlagen der Programmierung in Visual Basic 6.0
- Erstellung von Makros
- Einbinden von Makros in Visual Studio

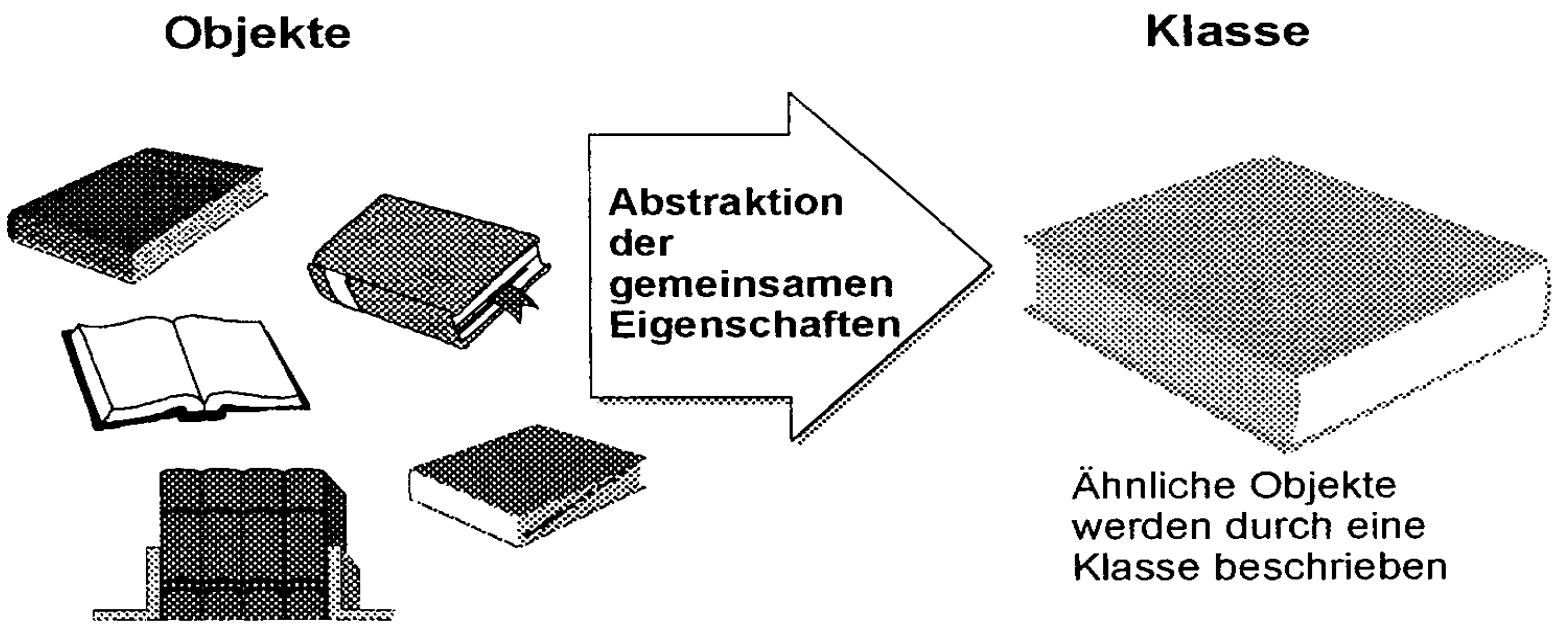
Objekt

Ein Objekt

- ist die Abstraktion eines “Begriffs”,
- *hat* eine eigene Identität,
- zeigt ein für seine Art typisches Verhalten,
- hat zu jedem Zeitpunkt einen bestimmten Zustand,
- der für das Verhalten in bestimmten Situationen ausschlaggebend sein kann.



Objekt und Klasse

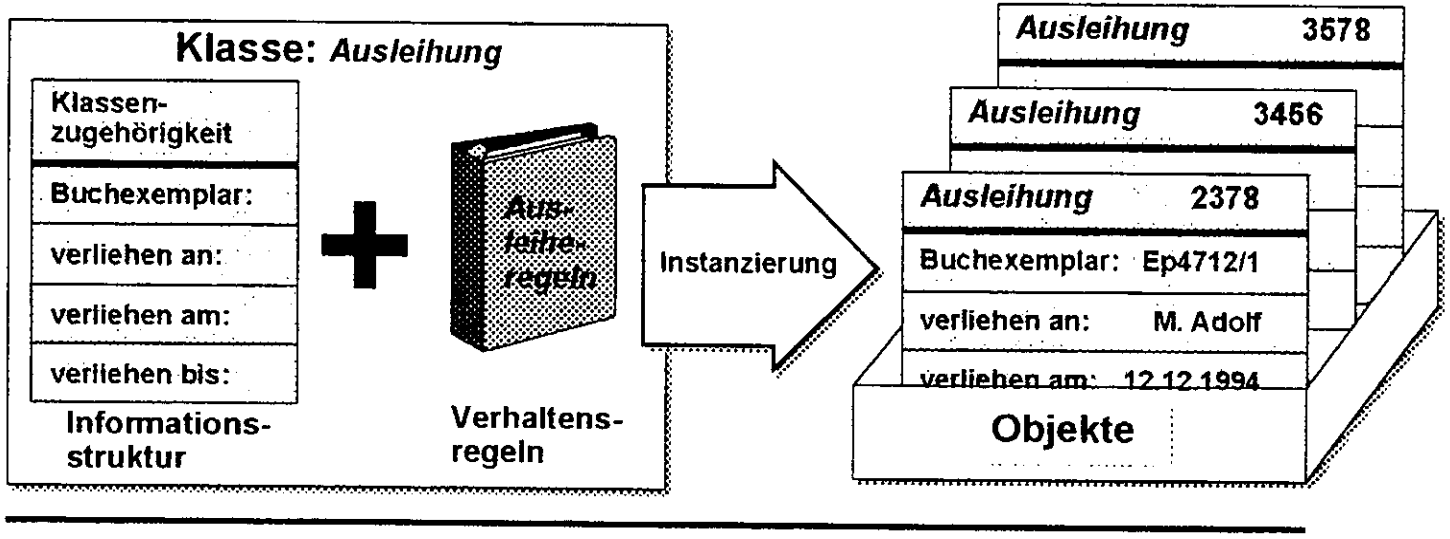


Klasse

- beschreibt den internen Aufbau der Objekte

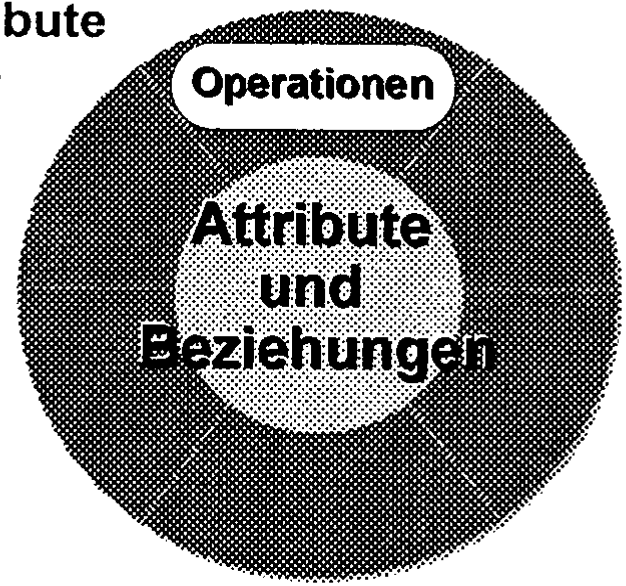
u n d

- ihr mögliches (artgerechtes) Verhalten;
- ist die Vorlage für beliebig viele gleichartige Objekte (Instanzierung).

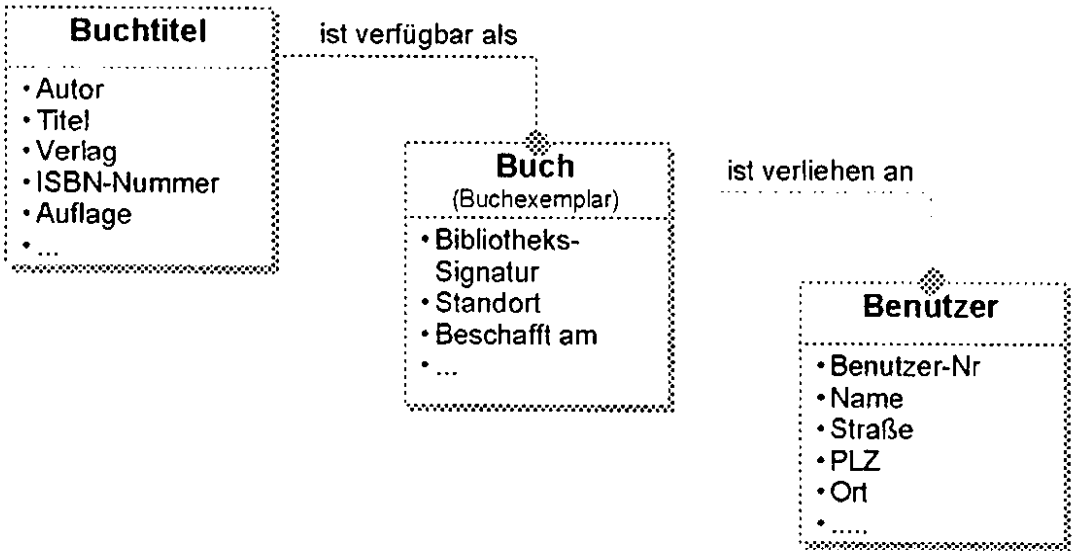


Attribute und Beziehungen

- Die strukturellen (statischen) Eigenschaften eines Objektes werden durch die Spezifikation in der Klasse bestimmt.
- Sie werden durch Attribute und Beziehungen ausgedrückt und
- bieten Platz für die Informationen, die den Zustand der jeweiligen Objekte beschreiben.



Attribute und Beziehungen

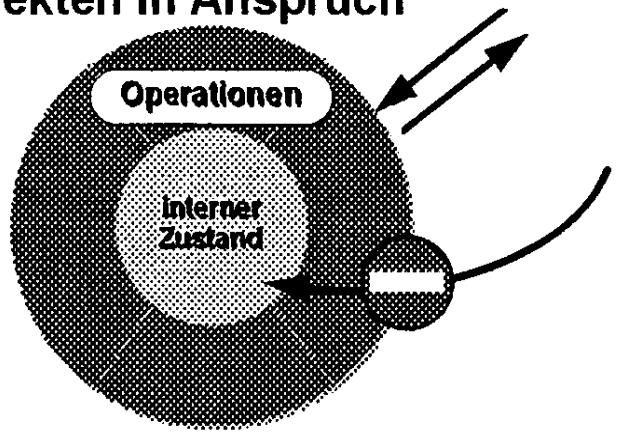


Operationen

- ⇨ Dienste, Services oder Methoden (Smalltalk)
- ❑ sind der verhaltensorientierte Teil,
- ❑ sind die dynamischen Eigenschaften,
- ❑ sind die Dienste, die ein Objekt oder eine Klasse bereithält,
- ❑ bestimmen, was man mit den Objekten machen kann (Nützlichkeit).

Datenkapselung durch Operationen

- Operationen sind die einzige Möglichkeit, mit den privaten Daten eines Objektes in Verbindung zu treten
  - um den Zustand des Objektes zu verändern oder
  - Auskunft über den Zustand zu erhalten
- ⇨ Prinzip der Datenkapselung
- Umgekehrt sind alle Operationen öffentlich zugänglich und können von anderen Objekten in Anspruch genommen werden.



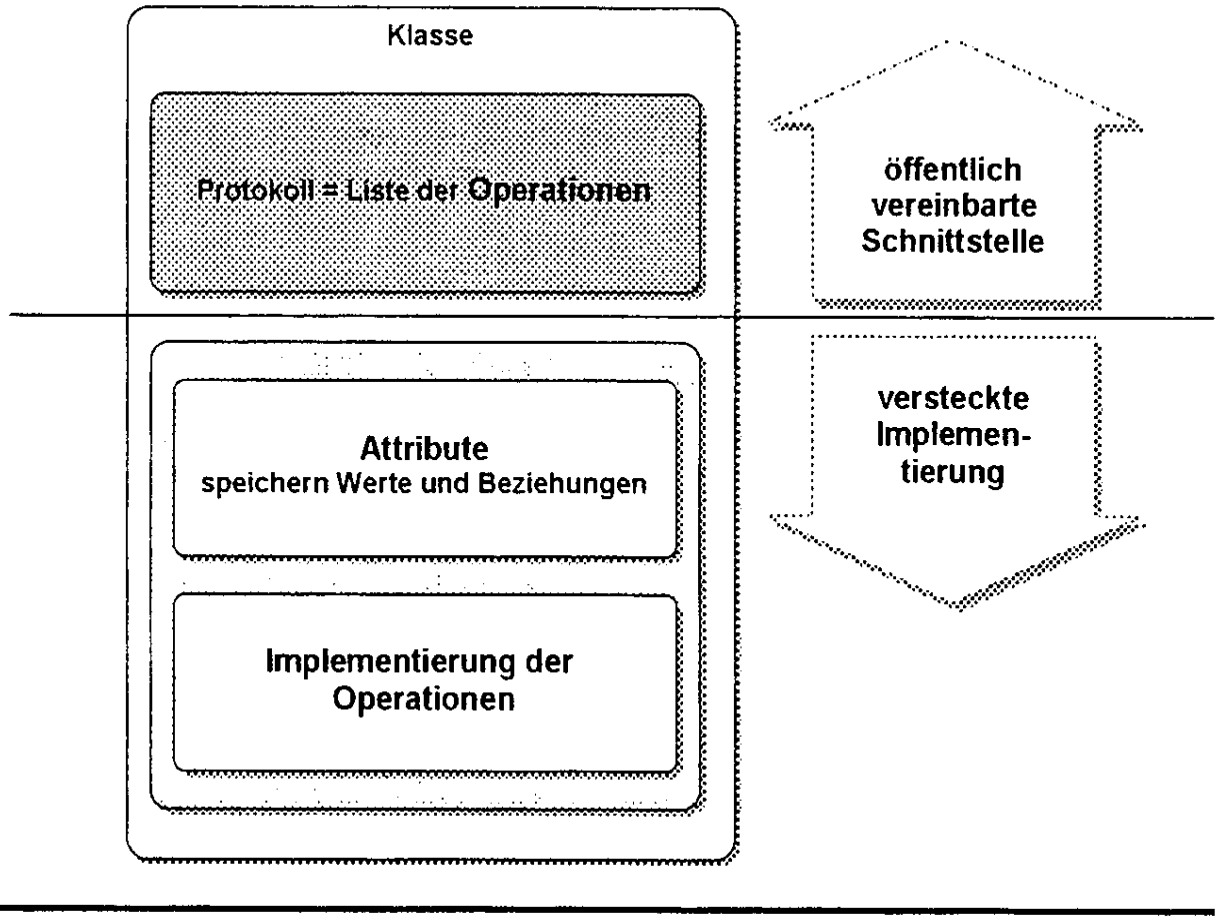
Datenabstraktion

- Nur die Schnittstelle wird veröffentlicht
- Zur Schnittstelle gehören
  - Name der Operationen
  - Parameter
  - Rückgabewert
  - Vorbedingung, d.h. welcher Zustand muß vorliegen, damit die Operation sinnvoll angewendet werden kann und
  - Nachbedingung, d.h. welcher Zustand liegt nach der Ausführung der Methode vor.
  - evtl. mögliche Fehlerzustände
- ⇨ Implementation Hiding
  die konkrete Implementierung ist nicht bekannt

Objekte und Client/Server-Prinzip

- ⇨ **Programming by Contract**
- ❑ **Der Server (Objekt bzw. seine Klasse) verpflichtet sich,**
  - ○ die Spezifikation der Schnittstelle als implementierte Leistung bereitzustellen und
  - ○ die Clients verlassen sich darauf.
- ❑ **ein Client darf keine Annahmen über die interne Implementierung im Server treffen.**
- ❑ **Server dürfen keine Annahmen über die Art der Clients oder den jeweiligen Verwendungskontext treffen.**

Schematischer Aufbau einer Klasse



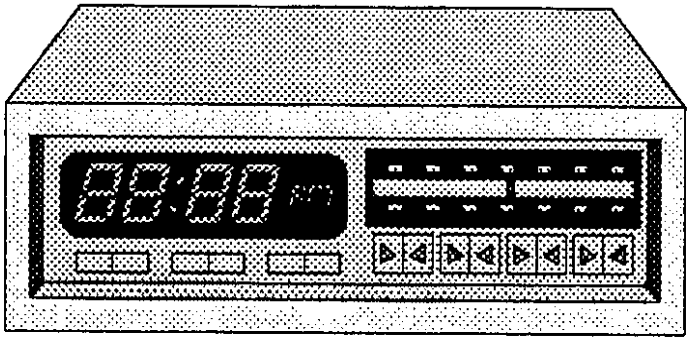
Operationen und Wiederverwendbarkeit

Eine Klasse ist umso nützlicher,

- je mehr Operationen sie anbietet,
- je elementarer die Operationen sind,
- je allgemeingültiger die Operationen sind,
- je selbständiger die Klasse ist,
- je vollständiger der Lebenszyklus ist,
- je genauer die angebotene Leistung die geforderte Leistung bereits abdeckt.

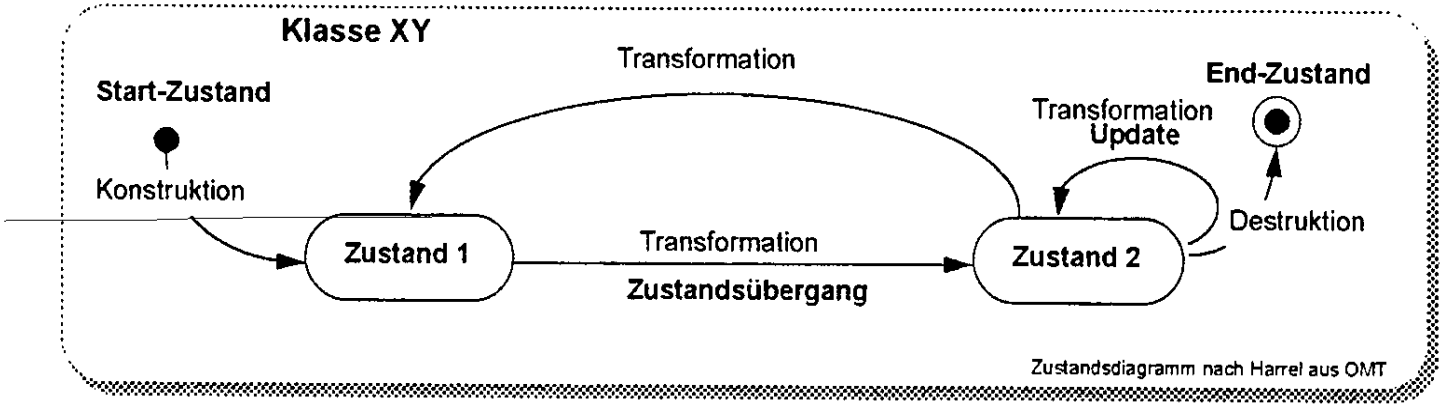
Protokoll

- Die Summe aller Operationen, die ein Objekt bzw. seine Klasse als Schnittstelle anbietet, heißt *Protokoll*.
- Das Protokoll ist öffentlich.
- Die darin enthaltenen Operationen können von anderen Objekten als Dienste in Anspruch genommen werden.



Objekt als Zustandsautomat

Das Protokoll muß den gesamten Lebenszyklus eines Objektes abdecken

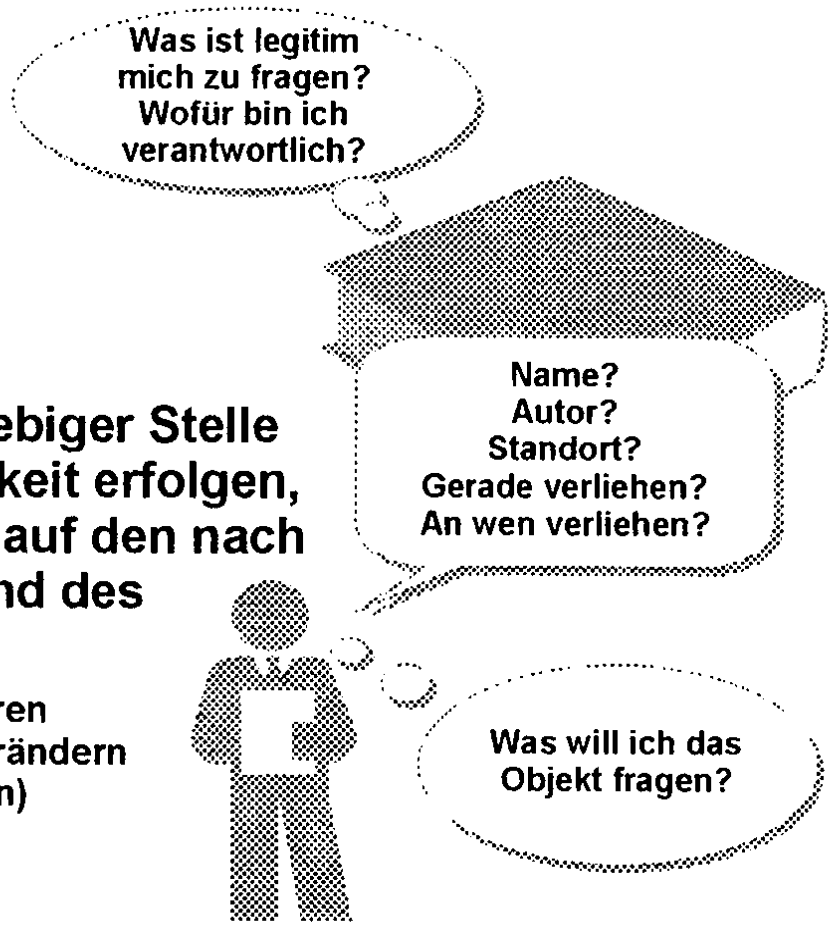


- **Zustandsänderung**
  - Konstruktion
  - Transformation / Zustandsübergang
  - Destruktion

Was will ich einem Objekt bzw. seiner Klasse alles befehlen?

Objekte als Datenspeicher

- eine Abfrage ist
  - eine Informationsabfrage
    - originär
    - abgeleitet
  - eine Zustandsabfrage
- Abfragen dürfen an beliebiger Stelle und in beliebiger Häufigkeit erfolgen, dies bleibt ohne Einfluß auf den nach außen sichtbaren Zustand des befragten Objektes
  - Abfragen können den inneren Zustand eines Objektes verändern (z.B. Caching-Mechanismen)



Methoden vs. Operationen

- Operationen spezifizieren die dynamischen Eigenschaften
- Das Protokoll ist die Summe aller Operationen

- Operationen werden durch Methoden implementiert
- Prozeduren und Funktionen sind somit entweder
  - Methoden und als Operationen im Protokoll verzeichnet und damit öffentlich zugänglich

  oder

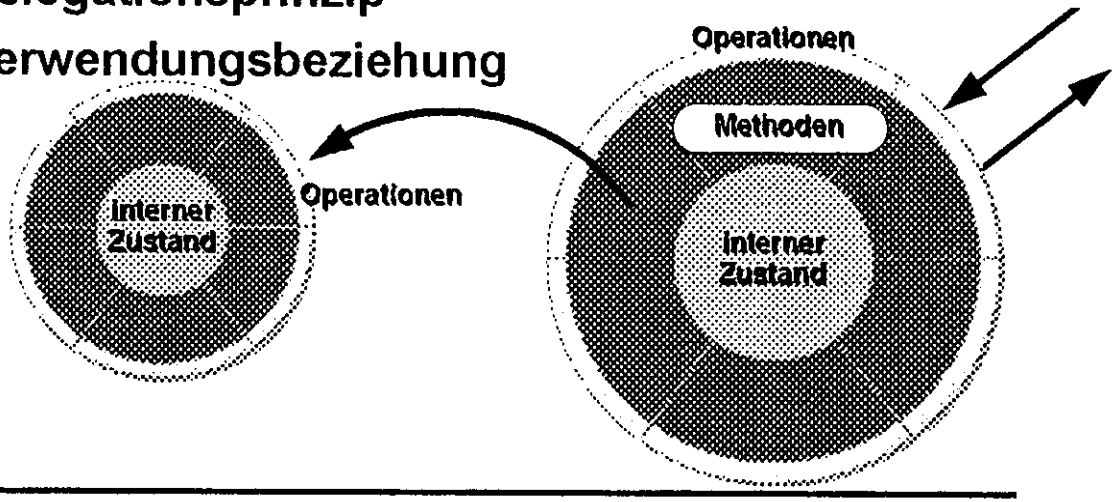
  - nach den Prinzipen strukturierter Programmierung entworfene private Unterprogramme
- Methoden tun nichts, was andere bereits tun

Methoden

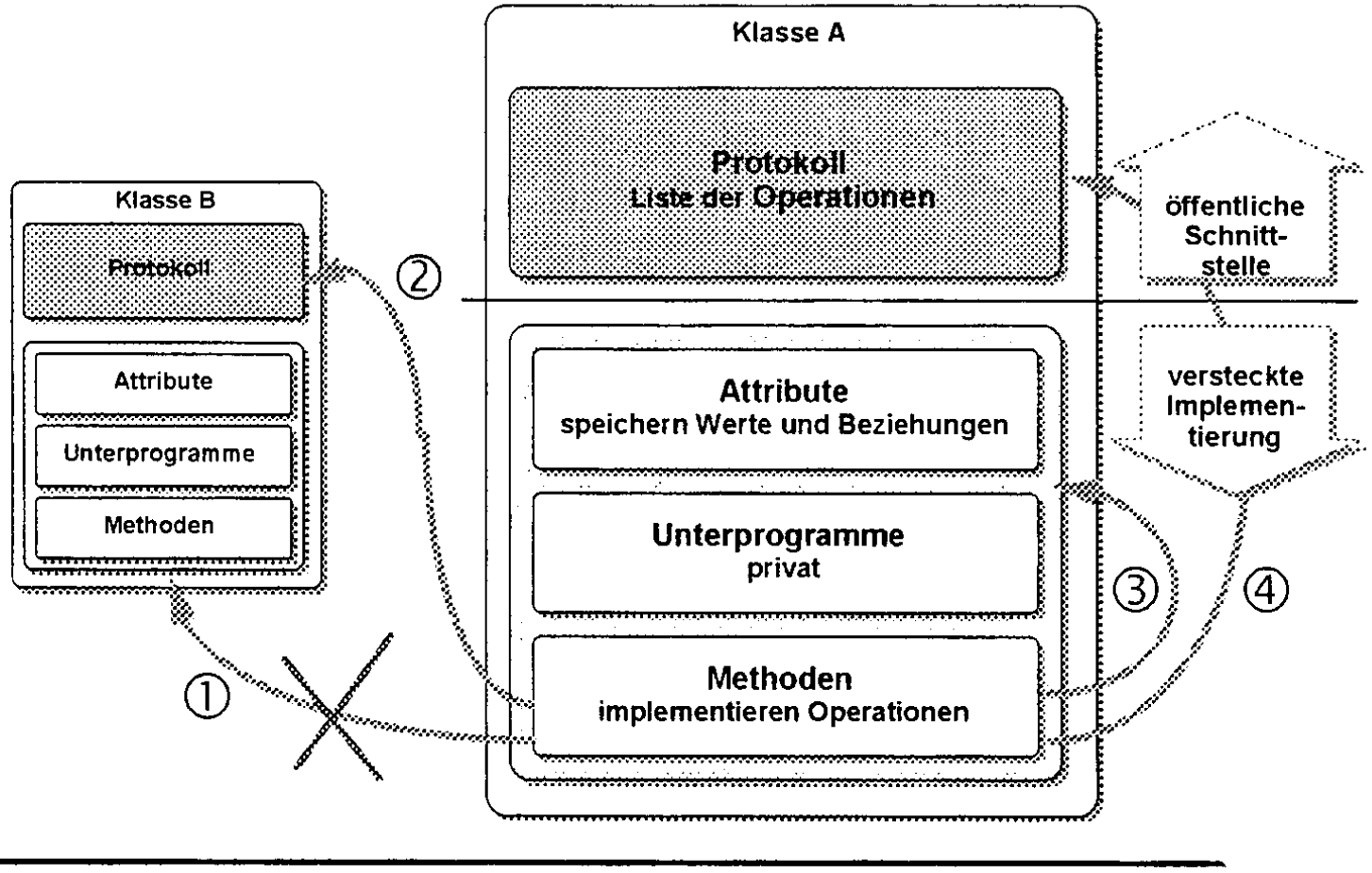
- Methoden bedienen sich klasseneigener (oder konventioneller) Unterprogramme
- oder benutzen Leistungen, die von anderen Objekten (der selben oder einer anderen Klasse) als Operationen angeboten werden.

⇨ Delegationsprinzip

⇨ Verwendungsbeziehung



Schematischer Aufbau einer Klasse (verfeinert)



Finden von ...

Unterstreiche alle Substantive und bilde daraus Klassen, unterstreiche alle Verben und mache aus ihnen Operationen.

Substantive	Verben	Adjektive
Buch		
Buchtitel		
Benutzer		
Bibliothekar		
Reservierung	reservieren	reserviert
Ausleihung	ausleihen	ausgeliehen
Mahnung	mahnen	gemahnt
Rückgabe	zurückgeben	zurückgegeben
....		

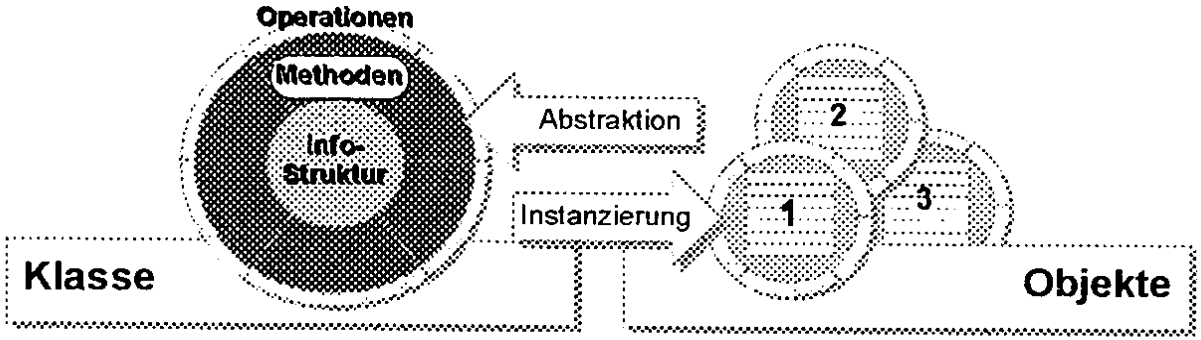
Ereignisse

Objekte / Klassen

Operationen

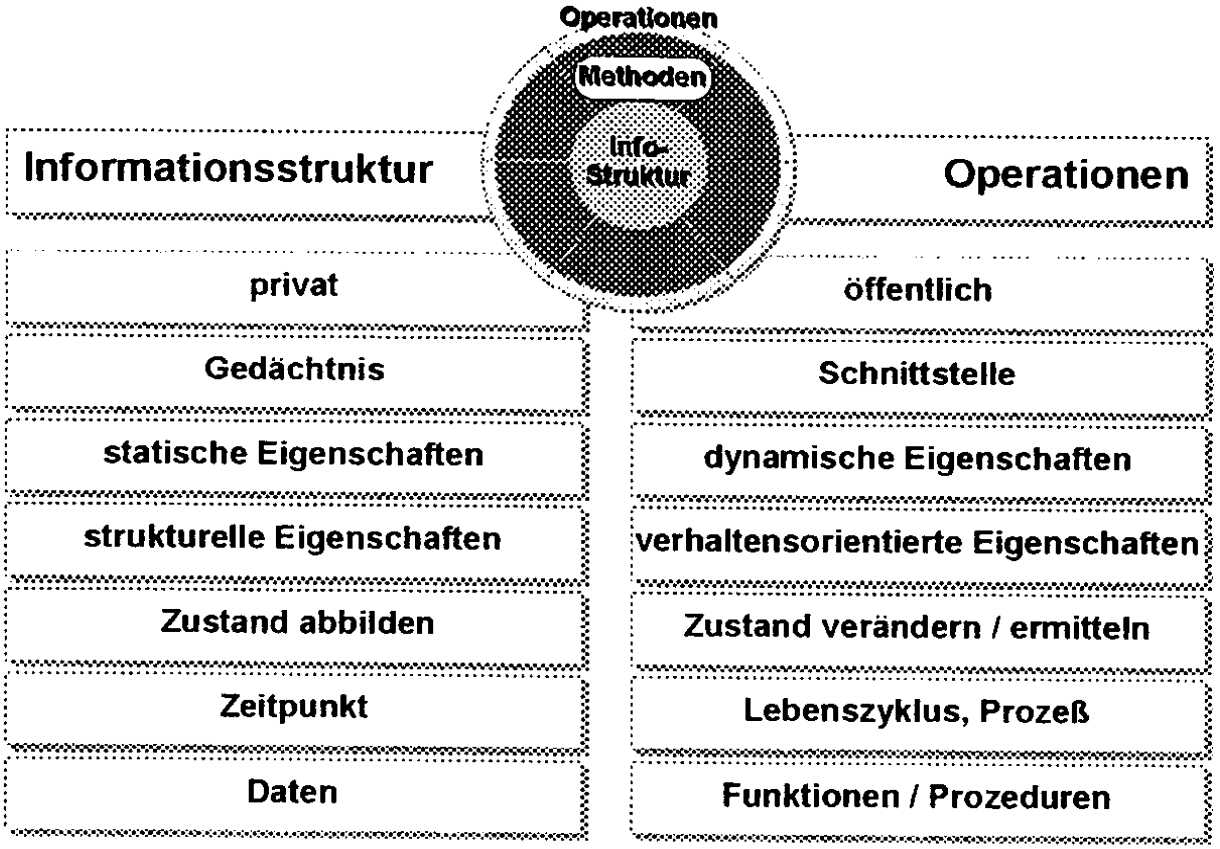
Zustände / Attribute

Klasse vs. Objekt



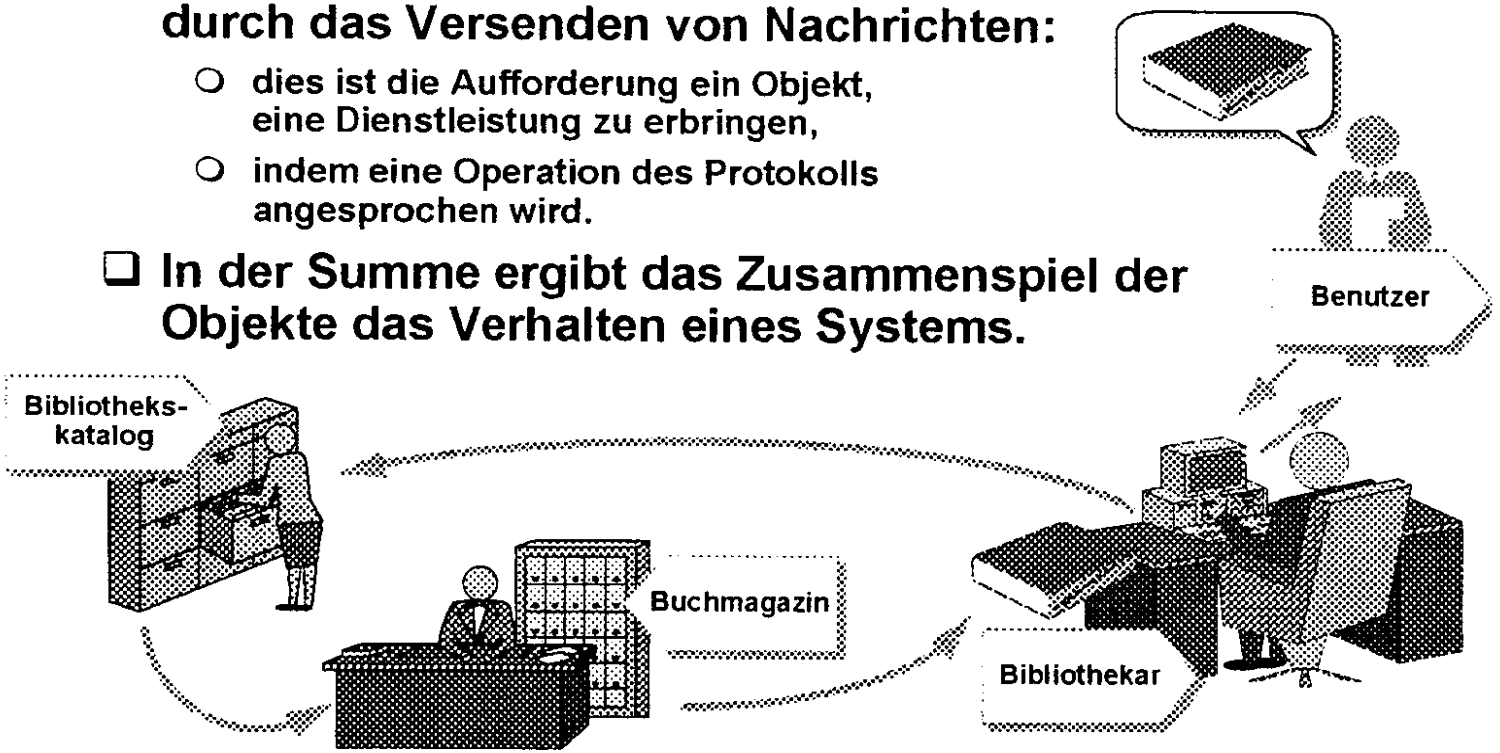
Klasse	Objekte
Muster, Bauplan	Exemplar
Informationsstruktur	individueller Zustand
Operationen & Methoden	versteht Operationen
kann beliebig Objekte erzeugen	kennt Klassenzugehörigkeit
einmal im System vorhanden	beliebig oft vorhanden
hat eindeutigen Namen	hat eigene Identität

Eigenschaften eines Objektes



Nachricht

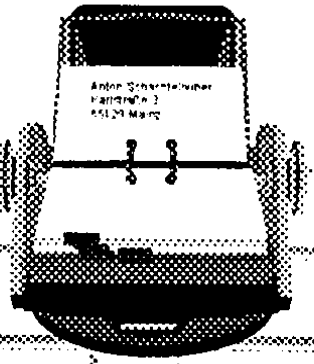
- Objekte kommunizieren untereinander durch das Versenden von Nachrichten:
  - dies ist die Aufforderung ein Objekt, eine Dienstleistung zu erbringen,
  - indem eine Operation des Protokolls angesprochen wird.
- In der Summe ergibt das Zusammenspiel der Objekte das Verhalten eines Systems.



Nachricht

Benutzerkarte.ändereAdresse
('Karlstraße 3', '55129','Mainz');

C++



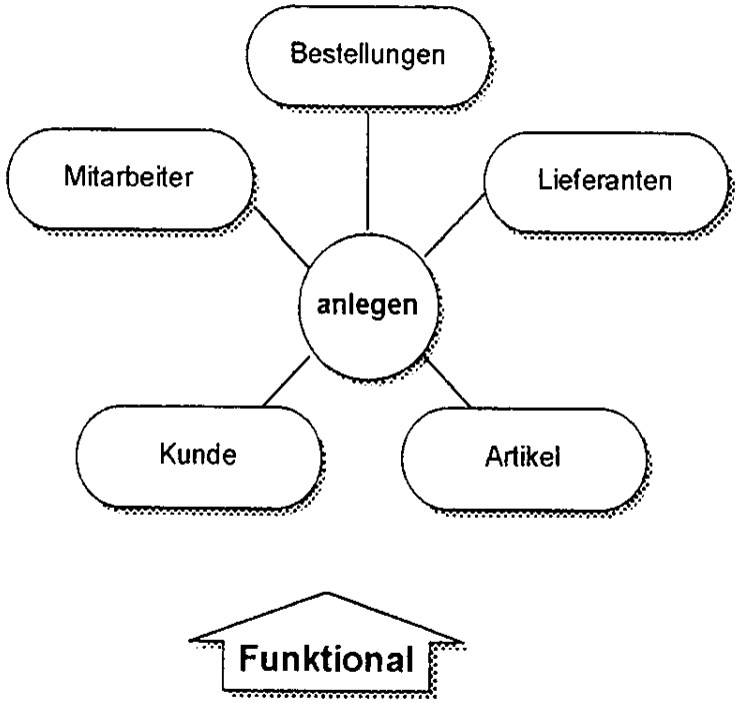
Benutzer	Klasse
Benutzerkarte	Objekt
ändereAdresse (straße, plz, ort)	Signatur = Operation (Parameter,.)
"Karlstraße 3", "55129" "Mainz"	Argumente

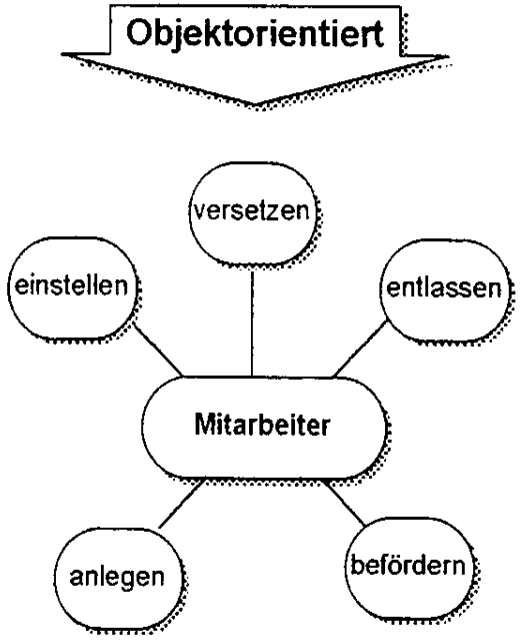
Selektor

Smalltalk

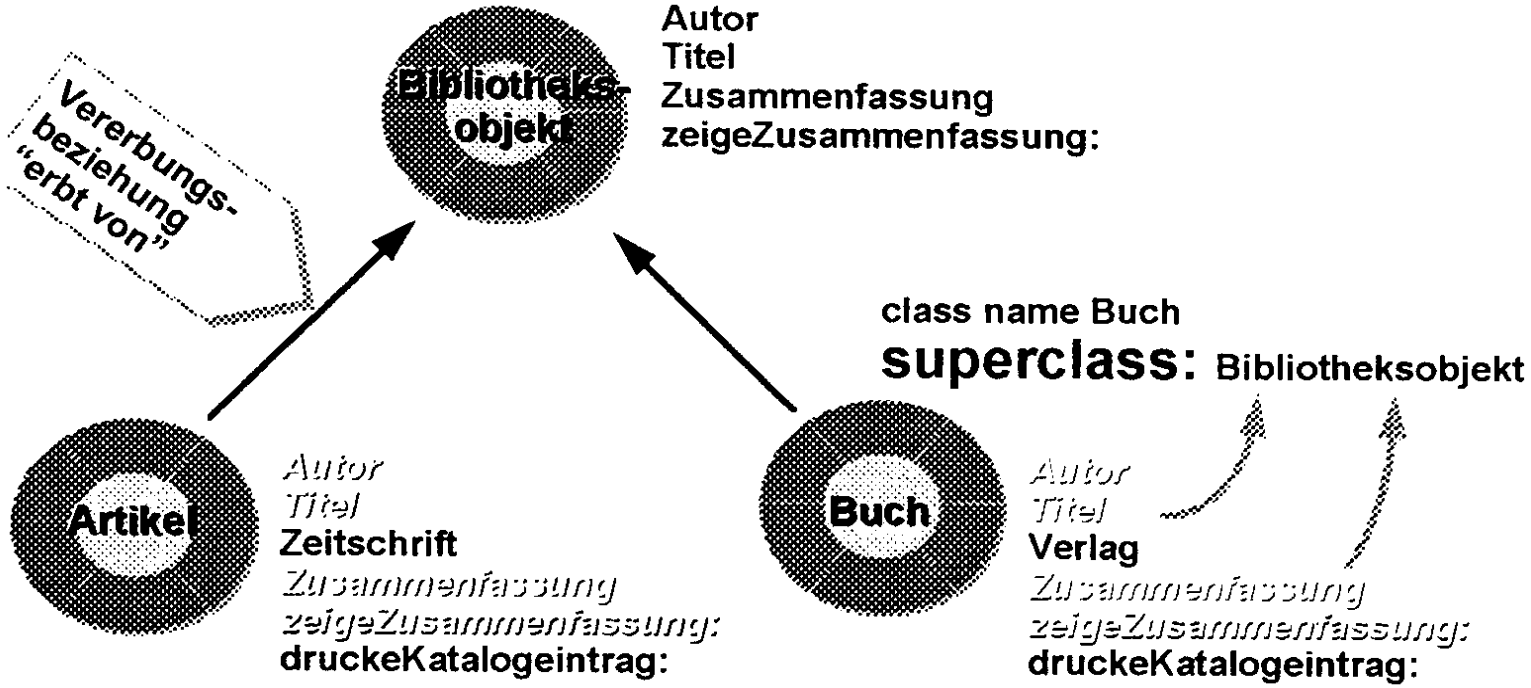
Benutzerkarte ändereStraße: 'Karlstraße 3'
änderePostleitzahl: '55129' ändereOrt: 'Mainz';

Objektparadigma





Generalisierung und Spezialisierung



Vererbung

- Mechanismus zur Wiederverwendung von in der Elternklasse spezifizierten Eigenschaften durch die Kindklasse(n).
- Die Vererbung unterstützt
  - Generalisierung / Spezialisierung
  - Abstraktion / Konkretisierung
  - Erweiterung von Klassen, die nicht verändert werden sollen oder dürfen (z.B. gekaufte oder alte Klassen)

is_a
is_Kind_of

- Kindklasse spezifiziert nur die Unterschiede in den Eigenschaften
  - hinzufügen
  - verändern
  - konkretisieren
  - umbenennen
  - unterdrücken

superclass
inherit

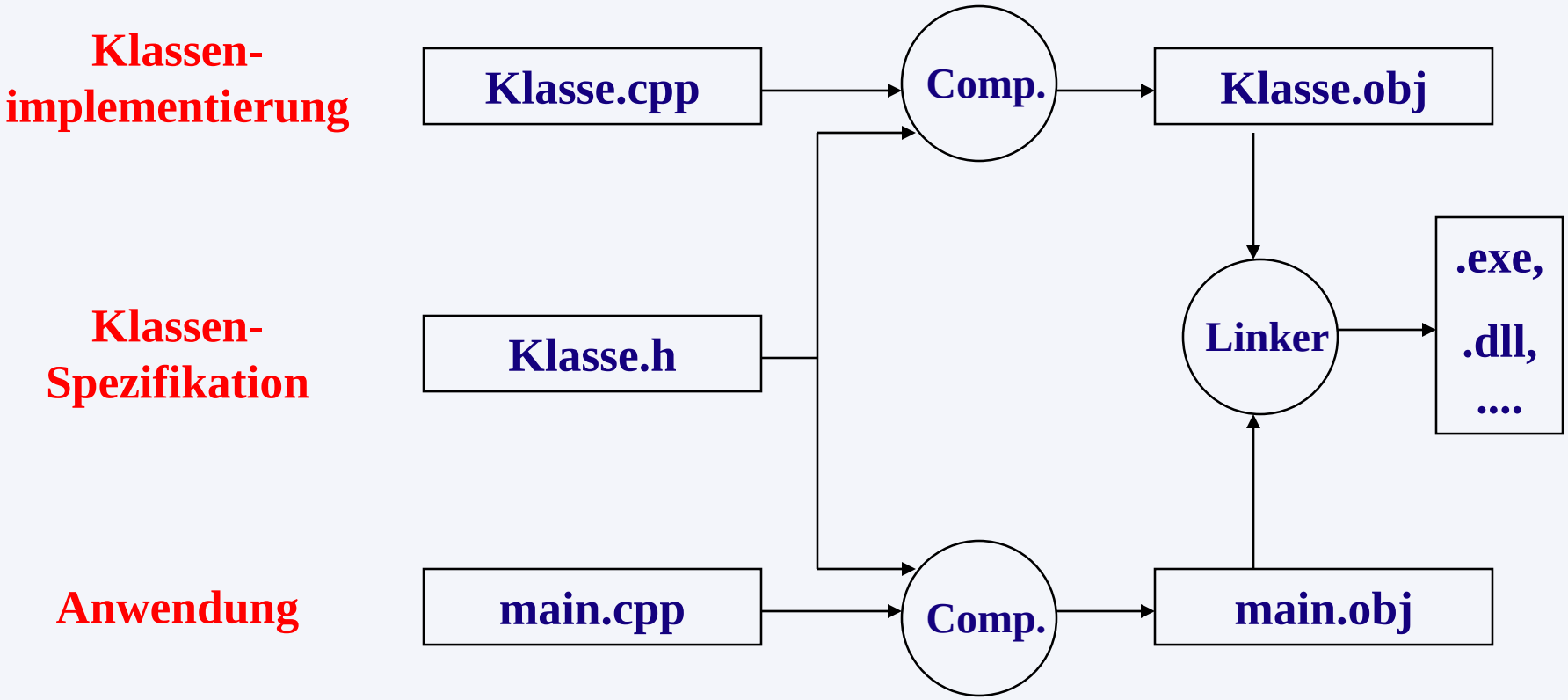
Dateinamen

Für C++-Code existieren zwei Arten von Dateien:

Header-Dateien **verwalten Definitionen, die in verschiedenen Modulen gebraucht werden. Sie enthalten Deklarationen von Konstanten, Variablen, Funktionen und Datentypen. Endungen sind typischerweise .h oder .hpp.**

Quelldateien **sind die eigentlichen Module. Hier befinden sich alle Implementierungen. Typische Endungen sind .cpp, .cc oder .cxx.**

Vorgänge beim Erstellen einer Anwendung



Deklarationen

Um eine Klasse in C++ verwenden zu können, muß sie erst deklariert werden. Dies geschieht innerhalb einer Struktur, die in eine Header-Datei geschrieben wird. Sie enthält sowohl alle *Komponenten* als auch alle *Funktionen.*

Eine Header Datei wird durch *Präprozessor*-Anweisungen eingeschlossen, die vermeiden, daß eine Header-Datei mehrfach eingebunden wird.

```
#ifndef _Klasse_h    // nur beim ersten Durchlauf erfüllt

#define _Klasse_h   // wird beim ersten Durchlauf definiert

.....

#endif
```

Beispiel einer Deklaration an der Klasse *Bruch*

```
#ifndef _Klasse_h    // nur beim ersten Durchlauf erfüllt
#define _Klasse_h   // wird beim ersten Durchlauf definiert
class Bruch{
private:                    // private Deklarationen
        int zaehler, nenner;
public:                     // öffentliche Schnittstelle
        Bruch(int,int);         // Konstruktor
        void print();           //Ausgabe
};

.....
#endif
```

Das Schlüsselwort *Class*

- Eine Klassendeklaration beginnt immer mit dem Schlüsselwort *Class*
- Es folgt der *Identifier* und der Name der Klasse
- Der Klassenrumpf wird in Klammern eingeschlossen
- Abgeschlossen wird die Deklaration durch ein Semikolon

Zugriffschlüsselworte

- Innerhalb einer Klassenstruktur befinden sich Schlüsselworte, die den Zugriff auf Komponenten regeln.
- Sie legen fest, auf welche Komponenten nur innerhalb der Klasse und auf welche Komponenten von außerhalb zugegriffen werden darf.
- Das Schlüsselwort *private* läßt nur Zugriffe innerhalb der Klasse zu. Dies ist die Defaulteinstellung für alle Komponenten.
- Das Schlüsselwort *public* deklariert Schnittstellen nach außen. Alle hier aufgeführten Komponenten sind öffentlich.

Zugriffschlüsselworte

- Elementfunktionen werden in der Regel öffentlich deklariert
- Alle anderen Komponenten (Daten) sollten privat deklariert werden.
- Durch diese Vorgehensweise können Daten nur über definierte Schnittstellen manipuliert werden.
- Es kann Klasseninterne Hilfsfunktionen geben, die ebenfalls nur privat deklariert werden.

Elementfunktionen

- Elementfunktionen der Klasse Bruch ist die Funktion *print().*

- class Bruch {

```
        .......
        void print ();                  // Ausgabe eines Bruches
        .......
};
```

- *Print()* gibt einen Bruch auf dem Bildschirm aus.
- *Print()* liefert keinen Wert zurück, der Typ ist void.
- Der auszugebende Wert wird nicht übergeben. Dieser ist der Elementfunktion bekannt.

Konstruktoren

- Konstruktoren legen fest, auf welche Weise Objekte erzeugt (konstruiert) werden.
- Konstruktoren sind spezielle Funktionen; die den Namen der Klasse tragen.
- Ein Konstruktor wird immer dann aufgerufen, wenn ein neues Objekt einer Klasse gebildet werden soll. Das ist z.B. bei der Deklaration der Fall.
- Konstruktoren können überladen werden.
- Konstruktoren können Parameter übergeben werden.
- Konstruktoren haben keinen Rückgabewert.

Konstruktoren

- Beispiel für Konstruktoren der Klasse Bruch

```
class Bruch {

        Bruch();                // Default-Konstruktor

        Bruch(int);                     // Konstruktor mit Zähler

      Bruch(int,int);         // Konstruktor mit Zähler und
                //                      Nenner
};
```

Konstruktoren

- Beispiel für Konstruktoren der Klasse Bruch *in der Datei Bruch.cpp*

```
Bruch::Bruch (int i, int k)
 {
        zaehler = i;
        nenner = k;
}
```

```
void main ()
 {
        Bruch Zahl ( 2, 3);
        Zahl.print ();

}
```

Default Parameter

Deklaration:

```
Class bruch
{
…
        bruch ( int i, int k = 1); // Default-Par. Nur in Deklaration
};

Bruch::Bruch (int i, int k /* = 1 */)
 {
        zaehler = i;
        nenner = k;
}
```

```
void main ()
 {
        Bruch Zahl ( 2 );
        Zahl.print ();

}
```

Überladen von Funktionen

- In C++ dürfen mehrere Funktionen den gleichen Namen besitzen.
- Diese Eigenschaft wird als *Überladen* (overloading) bezeichnet.
- Sinnvollerweise sollten Funktionen gleichen Namens ähnliche Aufgaben erfüllen.
- Die aufzurufende Funktion wird durch die Anzahl und Typen der übergebenen Parameter erkannt.
- Eine Unterscheidung nur durch den Rückgabetyp ist nicht zulässig.

Beispiel

•Deklaration :

```
int quadrat (int);                  //Quadrat eines int‘s

double quadrat (double);                    //Quadrat eines doubles‘s
```

•Aufruf :

```
x = quadrat (3);

y = quadrat (4.3789);
```

Operatoren für Klassen

- Es besteht die Möglichkeit, Operatoren für eigene Klassen zu definieren.

```
Bruch a, b, c;                    //Deklaration dreier Brüche

if  (a<b) {                 //Prüfen, ob a kleiner b ist

            c = a * b ; }
```

- Operator * zum multiplizieren zweier Brüche
- Operator < zum Vergleich zweier Brüche

Deklaration von Operator-Funktionen

```
Class Bruch {
                int zaehler, nenner;
                public :
                Bruch (int, int);                       //Konstruktor
                Bruch operator * (Bruch)        //Multiplikation
                Bruch operator < (Bruch)        //Vergleich
};
```

Implementieren von Operator-Funktionen

```
Bruch Bruch :: operator * (Bruch b) {
                    Bruch ergebnis;                 //Ergebnis-Bruch
                    // Produkt zuweisen
                    ergebnis.zaehler = zaehler * b.zaehler;
                    ergebnis. nenner = nenner * b.nenner;
                    return ergebnis ;
}

besser :
return Bruch(zaehler * b.zaehler, nenner * b.nenner);
Aufruf:
Bruch b1, b2;
b1 * b2;  // ist das selbe wie:  b1.operator * (b2);
```

Inline-Funktionen

- Funktionen können „inline“ deklariert werden.

- „inline“ bedeutet, daß „innerhalb einer Anweisung“ mit einem Funktionsaufruf, dieser durch die Anweisung ersetzt werden kann.

- Um das zu ermöglichen, wird die Funktion in der Header Datei deklariert. Dafür existieren zwei Möglichkeiten :

1. Die Funktion wird mit dem Schlüsselwort inline in der Header Datei dekariert.

2. Die Funktion wird bei der Deklaration in der Klassenstruktur direkt implementiert.

Beispiele für Inline-Funktionen

Zu 1.

```
Class Bruch{          .....
                      Bruch operator * (Bruch);
};
inline Bruch Bruch::operator * (Bruch) {
          return Bruch(zaehler * b.zaehler, nenner * b.nenner);
}
```

Beispiele für Inline-Funktionen

Zu 2.

```
Class Bruch{        .....
                    void print() {
                            printf(„%d/%d\n“,zaehler,nenner);
                    }
                    .....
};
```

Copy-Konstruktoren

Deklaration eines Objektes mit Initialisierung eines temporären zweiten Objekts:

```
Bruch a;
.........
Bruch x = a*a;        //Deklaration und Initialisierung von x
```

Gleichbedeutend mit:

```
Bruch a;
.........
Bruch x (a*a);        //Deklaration und Initialisierung von x
```

Copy-Konstruktoren

- Ein Objekt wird erzeugt, indem ein Konstruktor aufgerufen wird, der als Parameter wiederum ein Objekt enthält.
- Einen so aufgerufenen Konstruktor nennt man *Copy-Konstruktor.*
- Ein Copy-Konstruktor erzeugt eine Kopie eines Objektes.
- Der Copy-Konstruktor kopiert komponentenweise.
- Ein Copy-Konstruktor muß nicht explizit deklariert werden, er ist für jede Klasse automatisch definiert.

Copy-Konstruktoren und Zuweisungen

- Es existiert ein Unterschied zwischen einer Deklaration und gleichzeitiger Initialisierung und einer Deklaration mit späterer Zuweisung

- Initialisierung

```
Bruch tmp;
.............
Bruch x = tmp ;       //Copy-Konstruktor
```

- Zuweisung

```
Bruch tmp;
.............
Bruch x;              //Default-Konstruktor
x = tmp ;             //Zuweisung
```

Copy-Konstruktoren und Parameterübergabe

- Der Aufruf

  x *= a

  ist gleichbedeutend mit

  x.operator *= (a)

- Vom Objekt a, das als Parameter übergeben wird, wird eine Kopie innerhalb der Operator-Funktion angelegt.

```
Bruch Bruch :: operator *= (Bruch b) {
        .....................
        Multiplikation

        .................
        return *this            // Kopie vom ersten Operanden zurückliefern
}
```

Copy-Konstruktoren und Parameterübergabe

- Der Parameter b in der Implementierung ist eine Kopie, die vom Copy-Konstruktor angelegt wurde

- Das Objekt b kann ohne Auswirkung auf die Aufruf-Funktion manipuliert werden.

- Alternative ist die unter C übliche Methode Zeiger zu übergeben. Die Parameter sind dann Adressen zu den verändernden Objekten.

Beispiel für Multiplikation :

```
x * &a
```

Referenzen

- Eine als Referenz deklarierte Variable ist nichts weiter als ein zweiter Name, der für ein Objekt vergeben wird.

- Deklariert wird eine Referenz durch ein zusätzliches &.

- Beispiel :

```
int& r = a; //r ist Referenz (zweiter Name) für a
```

Eine Referenz ist vom Typ her KEIN Zeiger, sondern hat den gleichen Typ wie das Objekt.

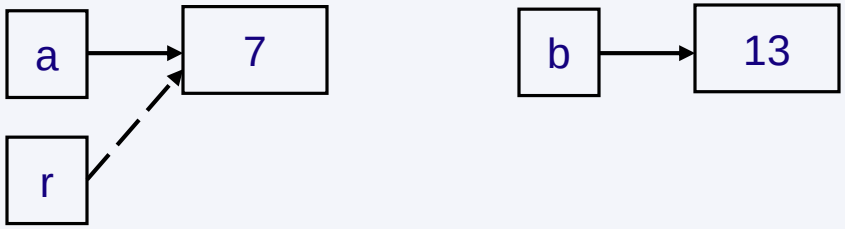
Nach obiger Deklaration ist es völlig unerheblich, ob für folgende Operationen r oder a verwendet werden.

Referenzen

- Beispiel für Referenzen:

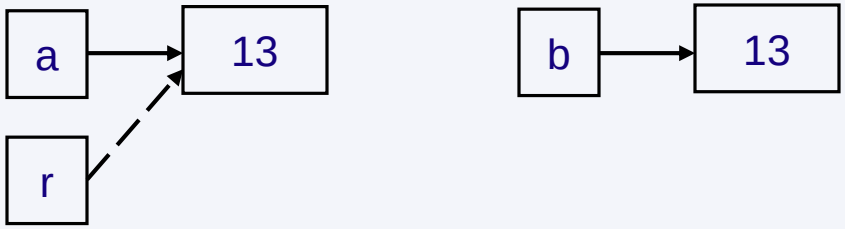
```
int a=7;                // int mit 7 initialisiert
int b=13;  // int mit 13 initialisiert
int& r=a;  // Referenz (zweiter Name) für a
```

- Mit der Deklaration der Referenz r wird diese auch mit a initialisiert:



- Wird r verwendet, ist dies gleichbedeutend mit a:

```
r=b;          // r bzw. a wird der Wert von b zugewiesen
```



Referenzen

•Auf diese Weise läßt sich beispielsweise eine Funktion, die zwei Argumente vertauscht folgendermaßen implementieren:

```
void swap(int& a, int& b) {
        int tmp;
        tmp=a;
        a=b;
        b=tmp;
}
```

•Die Funktion wird dann aufgerufen durch:

```
main(){
        int x=7, y=13;
        swap(x,y);          //vertauscht Werte von x und y
        .........
}
```

Referenzen

- Vorteil:

  Es müssen zum Verändern von Argumenten keine Zeiger mehr übergeben werden.

- Nachteil:

  Es gibt keine Sicherheit mehr, daß bei einem Funktionsaufruf die übergebenen Argumente nicht mehr verändert werden können, wenn nicht Zeiger übergeben werden.

- Vorsicht:

  Die Tatsache, daß es sich um Referenzen und nicht um Kopien handelt hängt nur von der Deklaration der Parameter ab.

Automatische Typumwandlung

Automatische Typumwandlung kann verwendet werden, wenn ein Konstruktor existiert, der nur einen Parameter benötigt.

Beispiel:

```
Bruch x;
while(x<1000){                    //statt: while (x<Bruch(1000)){
```

Außerdem ist eine Typumwandlung auch dann möglich, wenn der Konstruktor mit default-Argumenten besetzt ist.

Beispiel:

```
class Bruch{
                    .......
                    Bruch(int=0, int=1);
};
```

Automatische Typumwandlung

Automatische Typumwandlung ist nicht möglich für den umgekehrten Fall:

Beispiel:

```
Bruch x;
while(1000<x){
```

Der Vergleich x<1000 wird interpretiert als x.operator<(1000)

Durch Typumwandlung erfolgt daraus automatisch:

```
x.operator<(Bruch(1000))
```

Der Vergleich 1000<x wird interpretiert als 1000.operator<(x) //Fehler

Friend Funktionen

Um eine Typumwandlung auch für den Fall 1000<x möglich zu machen, gibt es die Möglichkeit den Operator als global zu deklarieren:

Beispiel:

```
bool operator < (Bruch&, Bruch&)
```

Der Vergleich 1000<x wird interpretiert als

```
Bruch(1000).operator<x
```

Da der Operator aber global deklariert wurde, kann außerhalb eines Objekts nicht auf Klassenelemente zugegriffen werden.

Lösung:
Die Funktion wird mit Hilfe des Schlüsselwortes „friend“ als Freund der Klasse deklariert und besitzt somit auch Zugriff auf private Klassenkomponenten.

Friend Funktionen

Definition der Klasse:

```
class Bruch{
                friend bool operator<(Bruch&,Bruch&);
                ..... };
```

Dadurch wird folgendes möglich:

```
main(){
        Bruch x;
        if (1000<x)          //geht auch
        ..... }
```

Implementierung des Operators:

```
inline Bruch operator <(Bruch&, Bruch&){  //nicht Bruch::
        return a.zaehler*b.nenner<b.zaehler*a.nenner;
..... };
```

Konvertierungsfunktionen

Konstruktoren, die mit einem Parameter aufgerufen werden, definieren, wie ein Objekt einer Klasse aus einem Objekt eines fremden Typs erzeugt werden kann.

Es gibt zusätzlich die umgekehrte Möglichkeit, ein Objekt in einen fremden Datentyp umzuwandeln. Dies geschieht mit Hilfe sogenannter Konvertierungsfunktionen.

Beispiel:

```
class Bruch{
        operator double ();
        ..... };
```

Implementierung:

```
inline Bruch::operator double () {
        return (double)zaehler/(double)nenner;
};
```

Konvertierungsfunktionen

Durch die Konvertierungsfunktion kann so ein Bruch auch immer dann eingesetzt werden, wenn ein double gebraucht wird.

Beispiel:

```
main(){
        Bruch(1,4);
        cout << sqrt(x);        //sqrt erfordert double
        ..... };
```

Implementierung:

```
inline Bruch::operator double () {
                return (double)zaehler/(double)nenner;
};
```

Dynamische Speicherverwaltung

Für dynamische Speicherverwaltung werden in C++ die Operatoren *new* und *delete* eingeführt.
Mit new wird ein Objekt explizit angelegt, mit delete wird es zerstört.

Die mit new und delete angelegten Objekte können sowohl zu einer Klasse gehören, als auch einen fundamentalen Datentyp besitzen.

Die neuen Operatoren ersetzen die in C bekannten Speicherplatzfunktionen malloc und free.

Als Operatoren sind new und delete Teile des Sprachumfangs von C++ und gehören nicht zu einer Standardumgebung.

New liefert im Gegensatz zu malloc immer einen Zeiger auf den richtigen Typ zurück. Der Rückgabewert muß also nicht explizit umgewandelt werden.

Der Operator „new“

Mit dem Operator new kann explizit Speicherplatz für ein Objekt angelegt werden.

Als Operand hinter new wird einfach ein Typ des Objektes angegeben, das explizit angelegt werden soll. Zurückgeliefert wird ein Zeiger auf dieses Objekt.

Beispiel:

```
float* fp = new float;          //legt float an
Bruch *bp = new Bruch;                  //legt Bruch an
```

Sofern es sich um eine Klasse handelt und ein Konstruktor definiert ist, wird dieser auch aufgerufen.

Beispiel:

```
Bruch *bp1 = new Bruch(100);        //legt Bruch an
Bruch *bp2 = new Bruch(77,2);               //legt Bruch an
```

Der Operator „new“

Der Aufruf new gibt einen Wert zurück, der eine Aussage darüber macht, ob die Speicherreservierung erfolgreich war.

Wird ein Wert 0 bzw. NULL zurückgeliefert, so ist kein Objekt eingerichtet worden.

Beispiel:

```
Bruch *bp = new Bruch;
if (bp==NULL){
        //Fehler, kein Objekt
}
```

```
Bruch *bp = new Bruch;
if (!bp){
//Fehler, kein Objekt
}
```

Der Operator „delete“

Der Operator delete dient dazu, ein mit new allokierten Speicherplatz wieder freizugeben.

Als Operand muß der von new zurückgelieferte Zeiger auf das Objekt (die Adresse des Objekts) übergeben werden.

Beispiel:

```
float* fp = new float;              //float anlegen
Bruch* bp = new Bruch;                      //Bruch anlegen
.....................
.....................
delete fp;                          //float freigeben
delete bp;                          //Bruch freigeben
.........
Bruch *bp1 = new Bruch(77,3)
Bruch *bp2 = bp1;
delete bp2;
```

Statische Klassenkomponenten

Klassenkompenenten haben die Eigenschaft, zu EINER Instanz einer Klasse zu gehören. Es treten aber immer wieder Aufgaben auf, bei denen ALLE Instanzen einer Klasse gemeinsame Komponenten besitzen müssen.

Ein Beispiel dafür ist Information über die Anzahl der existierenden Objekte einer Klasse. Eine Variable die sich nur auf eine Instanz bezieht, ist nicht in der Lage diese Information zu verwalten.

Beispiel:

```
class Person{
        ........
        Public:
                static long Anzahl(){
                        return personenanzahl;
                }
        ......
};
```

Statische Klassenkomponenten

Ein Beispiel soll das Arbeiten mit statischen Komponenten verdeutlichen:

Beispiel:

```
#include <stdio.h>
#include ´´Person.h´´
main()  {
        cout << „Personenzahl :“ << Person::anzahl()<<endl;
        Person nico („Hubert“,“Manfred“);
        cout << „Personenzahl :“ << Person::anzahl()<<endl;
        }
```

Die Personenzahl wird hier ohne Bindung an ein Objekt der Klasse ermittelt. Auf diese Weise kann die Anzahl auch ermittelt werden, ohne daß überhaupt eine Person exisitiert.

Vererbung und Polymorphie

Vererbung ermöglicht es, Eigenschaften einer Klasse von einer anderen abzuleiten, wobei nur noch Neuerungen implimentiert werden müssen. Dies führt neben einer Einsparung an Code zu einem Konsistenzvorteil, da gemeinsame Eigenschaften nicht nicht an mehreren Stellen stehen und nur einmal geprüft oder geändert werden müssen.

Polymorphie ermöglicht es, mit Oberflächen zu arbeiten. Verschiedenartige Objekte können zeitweise unter einem Oberbegriff zusammengefaßt werden, ohne daß ihre unterschiedlichen Eigenschaften verlorengehen.

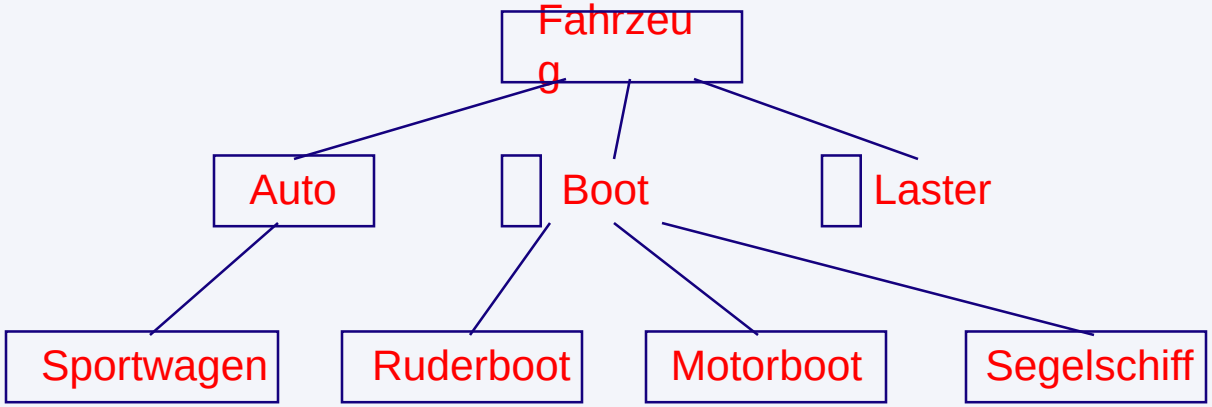
Vererbung ist eine Vorraussetzung für Polymorphie, da für jeden Oberbegriff eine Klasse definiert werden muß, von der die Klassen, die sich unter diesem Oberbegriff zusammengefassen lassen, jeweils abgeleitet werden.

Begriffe in C++

Wenn eine Klase die Eigenschaften einer anderen übernimmt, dann spricht man von Vererbung.

Die Klasse von der geerbt wird, nennt man Basisklasse. Die Klasse die erbt, nennt man die abgeleitete Klasse.

Da eine abgeleitete Klasse selbst Basisklasse sein kann, kann eine ganze Klassenhierarchieentstehen, wie folgendes Beispiel zeigt:



Mächtigkeit von Vererbung:

- Einfache Vererbung (engl.: single inheritance)
- Mehrfachvererbung (engl.: multiple inheritance)

Bei einfacher Vererbung kann eine abgeleitete Klasse nur eine Basisklasse besitzen, bei Mehrfachvererbung sind mehrere Basisklassen möglich. Bei einfach Vererbung gibt es also nur baumartige Hierarchien; bei Mehrfachvererbung sind es gerichtete Graphen.

Programmierung

- Grundbegriffe der Objektorientierten Programmierung
- Grundlagen der Programmierung in Visual Basic 6.0
- Erstellung von Makros
- Einbinden von Makros in Visual Studio

Merkmale der SolidWorks API (*Application Programming Interface*)

- OLE- Programmieroberfläche mit Hunderten von Funktionen
- Aufruf der Funktionen über C, C++, Visual Basic, VBA (Excel, Access) und SW-Makrodateien
- Vollständige Online-Dokumentation

Merkmale der Verwendung von Visual Basic:

- Visual Basic-Code ist ähnlich dem Code im Makro- Programmmodul

  => Aufgezeichneter Makro-Code dient als Grundlage für die zu erstellende Anwendung

- VB- Makros können in SolidWorks direkt aufgerufen werden, wenn sie nur API-Aufrufe enthalten. Eine vorherige Kompilierung ist nicht notwendig (Extras,Makro,Ausführen).
- SolidWorks erkennt nur VB-Befehle bis Version 3.0. Programme, die Befehle von VB >3.0 enthalten, müssen als ausführbares VB-Programm kompiliert oder von einer VBA-Anwendung gestartet werden.
- Programme können als separate .exe-Datei ausgeführt werden. Hierfür muß das Programm im Visual Studio als *.exe gespeichert werden: Datei, *.exe erstellen
- DLL-Implementierungen über VB werden nicht unterstützt

Merkmale von Visual Basic

Keine Zeilennummerierung

Wie bei modernen Hochsprachen werden die Codezeilen nicht durchnummeriert, eine Voraussetzung für prozedurales Programmieren. Um Sprungbefehle einzusetzen, können Zeilenmarken vergeben werden

Funktionen und Prozeduren

Durch Verwendung von selbstdefinierten Unterprogrammen (Routinen) können Programme modular aufgebaut werden. Solche Routinen (Funktionen und Prozeduren) lassen sich in eigenständigen Dateien zusammenfassen (Bibliotheken).

Objekte mit Eigenschaften und Methoden

Grundlegende Vorgänge werden nicht mehr über Routinen angesprochen, sondern als Objekte mit Eigenschaften und Methoden in den Programmcode eingesetzt.

Grafische Benutzeroberfläche mit ereignisgesteuerter Programmierung

Das Programm setzt sich aus grafischen Fensterelementen (Formen) und darin befindlichen Bedienungselementen (Steuerelemente oder Objekte) zusammen, die eine Benutzeroberfläche erzeugen und wartet auf Aktionen des Anwenders. Eine solche Aktion löst im Programm ein sogenanntes Ereignis aus. Innerhalb einer Ereignisprozedur kann mit linearer Programmierung die gewünschte Aktion ausgeführt werden.

Begriffe

Objekt

Objekte definieren sich durch Eigenschaften und auf sie anwendbare Methoden. VB kennt hier Anwendungen, Fenster, Steuerelemente, Geräte, etc.

Eigenschaften

Eigenschaften definieren bestimmte Zustände (.Color) eines Objektes oder sein Verhalten (.Enabled). Sie können gelesen oder gesetzt werden.

Methoden

Methoden bewirken eine Veränderung des Objektzustandes, sie funktionieren nur mit einem (bestimmten) Objekt-Typ. Methoden sind programmtechnisch innerhalb des Objektes abgelegte Prozeduren.

Anweisungen

Zur internen Steuerung des Programmablaufs: Kontroll- und Schleifenstrukturen, Wertzuweisung, Verbindung zu Hardware-Elementen wie Drucker, Laufwerke, etc.

Beispiele: **Objekt Haus: Eigenschaften: Größe, Farbe, Fenster, ...; Methoden: Ändern, Anmalen, Öffnen, Wechseln**

Objekt Auto: Eigenschaften: Farbe, Größe, Räder, Motor,...; Methoden: Lackieren, Öffnen, Starten, Ändern

Verwendung von Objekten und Eigenschaften

Objektverwendung

Haus.Farbe = rot	**Lies:** ***setze die Eigenschaft Farbe des Objektes Haus auf rot***
Auto.Farbe = rot	**Lies:** ***setze die Eigenschaft Farbe des Objektes Auto auf rot***
Haus.Räder = 4	**Lies:** ***setze die Eigenschaft Anzahl der Räder des Objektes Auto auf 4 => Fehlermeldung***

Objektverkettung

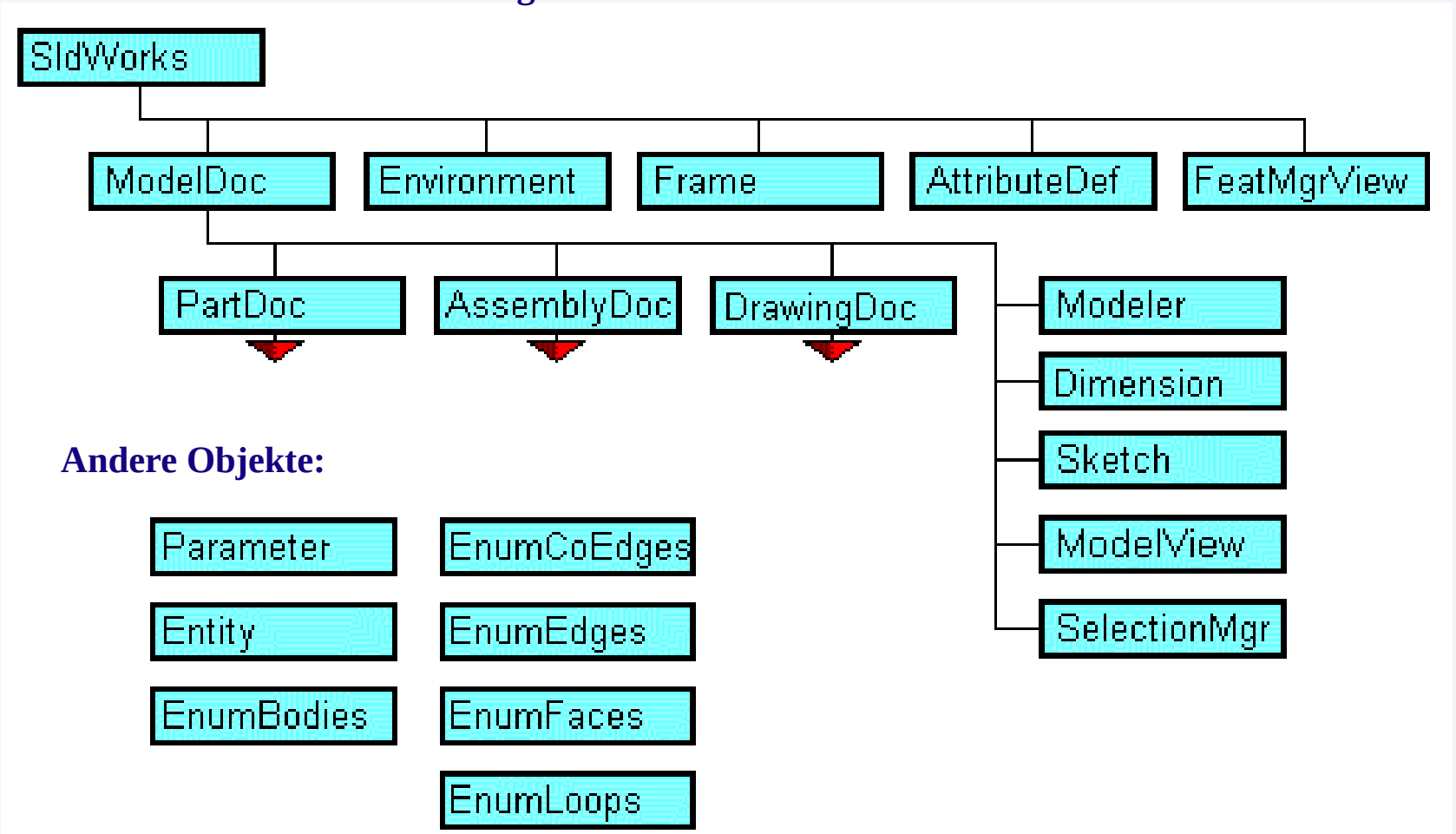
Haus.Fenster.Farbe = rot	***Das Objekt Haus.Fenster bekommt die Eigenschaft rot***
Haus.Fenster.Griff.Farbe = rot	***Das Objekt Haus.Fenster.Griff bekommt die Eigenschaft rot***

Verwendung von Objekten und Methoden

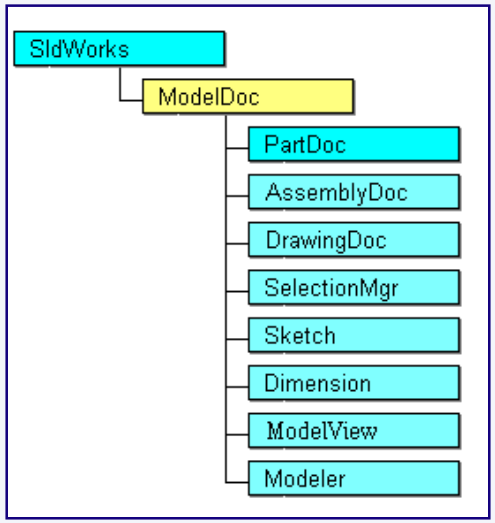
Haus.Anmalen ***Mit dem Objekt Haus wird über die Methode Anmalen eine Aktion ausgeführt => Ergebnis der Methode Anmalen ist identisch mit Setzen der Eigenschaft .Farbe***

Objekttypen der SolidWorks- API (Auswahl)

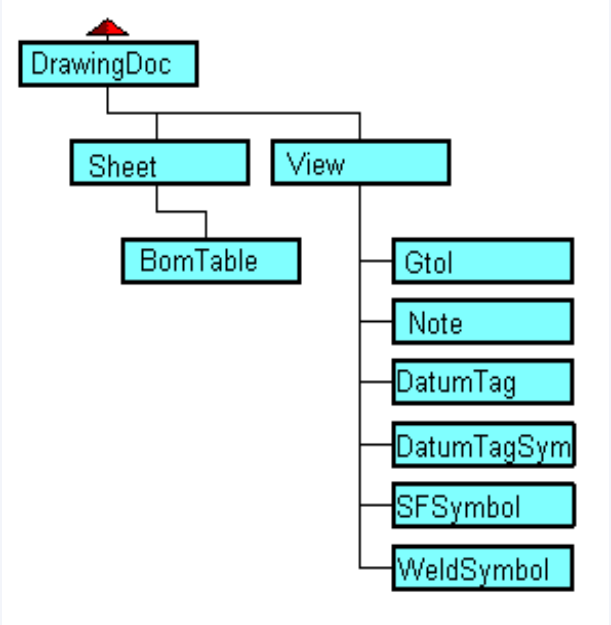
Dieses Diagramm zeigt eine Auswahl von Objekten in SolidWorks. Die Darstellung zeigt eine mögliche Gliederung aber keine hierarchische Ableitung. Die Pfade von und nach Objekten können sich von dieser Darstellung unterscheiden.

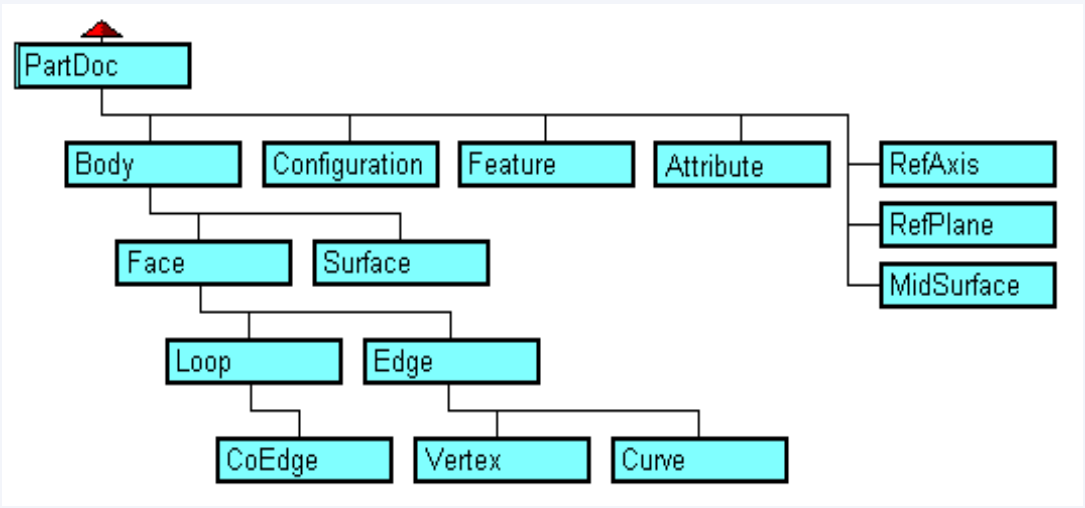


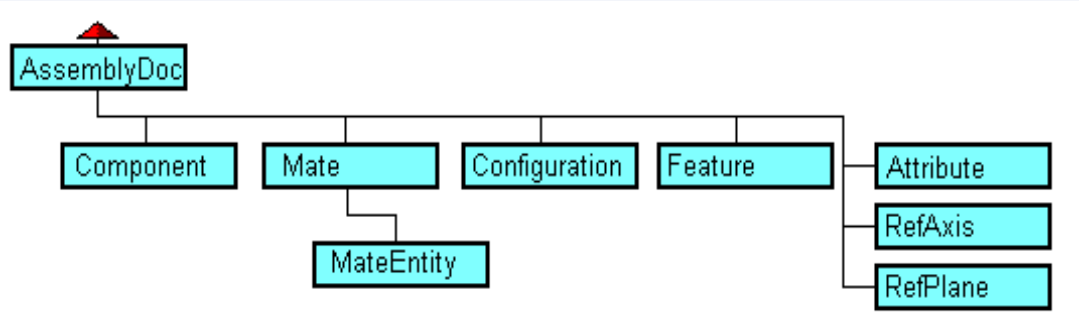
Objekt-Typen im ModelDoc-Bereich (Auswahl)



Das ModelDoc-Objekt stellt Funktionalitäten auf der Modell-Ebene zur Verfügung. Es umfaßt u.a. bemaßte Volumenkörper, Ansichtssteuerung, Skizzenoperationen, Parametersteuerung,Objekt- Auswahl, Öffnen und Speichern von Dokumenten, Erzeugung und Änderung von Features und die Erzeugung von Drahtmodellen.



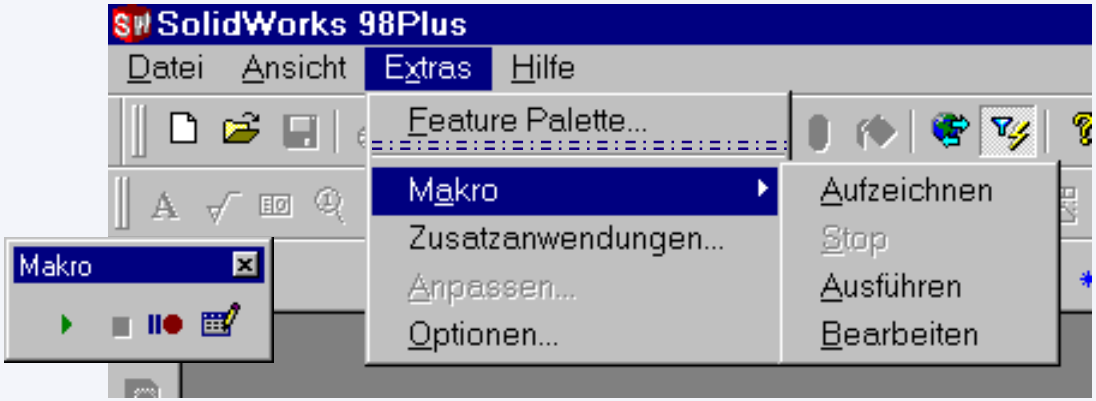




Makros in SolidWorks

Das Erstellen von VB-Programmen, die in SolidWorks Arbeitsabläufe automatisieren sollen, kann dadurch erleichtert werden, dass die dafür notwendigen Befehle zunächst mit einem Makro aufgezeichnet werden.

Dieser Programmcode kann anschließend in das VB-Projekt z.B. in einer Ereignisprozedur eingefügt werden.



Möglichkeiten der Arbeit mit Makros:

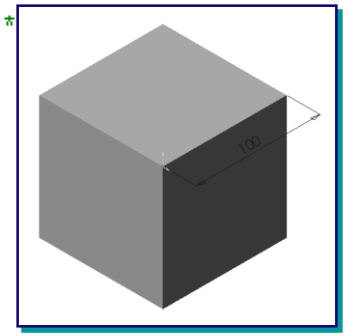
- **Aufzeichnen:** alle Befehle werden von Beginn der Aufzeichnung (im Hauptspeicher) protokolliert bis über “Stop“ der Vorgang beendet wird. Danach wird nach einem Dateinamen gefragt. In diese Datei wird der Makrocode als formatierter ASCII-Text geschrieben
- **Stop:** Beendet die Aufzeichnung von Makros
- **Ausführen:** Führt die Makrobefehle, die sich in einer anzugebenden Datei befinden, aus.
- **Bearbeiten:** Öffnet eine auszuwählende Datei im VBA- Editor aus. Falls kein Visual Studio vorhanden ist, so muß das VB-Programm mit diesem Editor erstellt werden.

Beispiel: Aufzeichnung der Erstellung eines Würfels

Macro2 - Macro21 (Code)

(Allgemein) | main

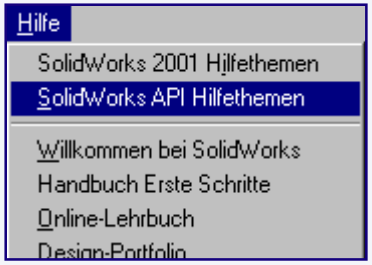
```
' ************************************************************************************
' C:\TEMP\swx65\Macro1.swb - macro recorded on 01/18/02 by krell
' ************************************************************************************
Dim swApp As Object
Dim Part As Object
Dim boolstatus As Boolean
Dim longstatus As Long
Dim Annotation As Object
Dim Gtol As Object
Dim DatumTag As Object
Dim FeatureData As Object
Dim Feature As Object
Dim Component As Object

Sub main()

Set swApp = CreateObject("SldWorks.Application")
Set Part = swApp.NewDocument("D:\Programme\SolidWorks\lang\german\Tutorial\Part.prtdot", 0, 0#, 0#)
Set Part = swApp.ActivateDoc("Teil14")
Part.InsertSketch
Part.SketchRectangle 0, 0, 0, 0.1002784263959, 0.09333747884941, 0, 1
Part.ClearSelection
Part.SelectByID "Line3", "SKETCHSEGMENT", 0.0549796108291, 0.09333747884941, 0
Part.AddDimension 0.0465774, 0.100644, 0
Part.ClearSelection
Part.Parameter("D1@Skizze1").SystemValue = 0.1
Part.SelectByID "Line2", "SKETCHSEGMENT", 0, 0.0611899323181, 0
Part.AddDimension -0.0294077, 0.0487693, 0
Part.ClearSelection
Part.Parameter("D2@Skizze1").SystemValue = 0.1
Part.FeatureExtrusion 1, 0, 0, 0, 0, 0.1, 0.01, 0, 0, 0, 0, 0.01745329251994, 0.01745329251994, 0, 0
```

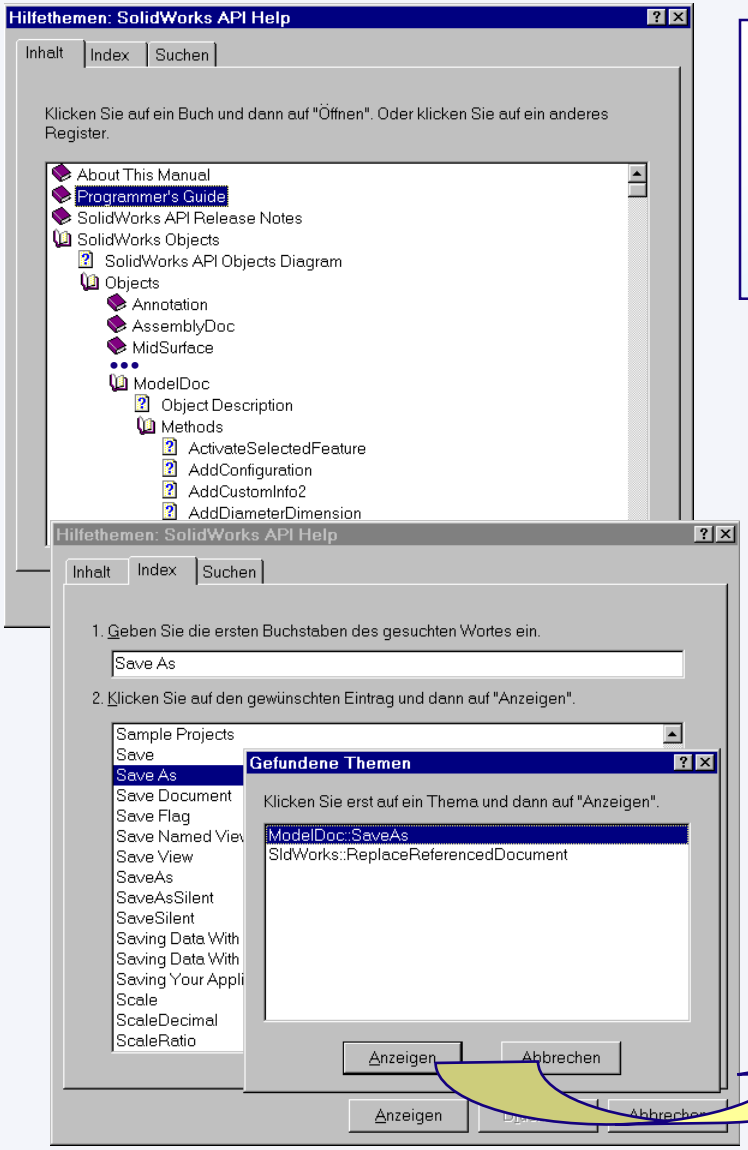


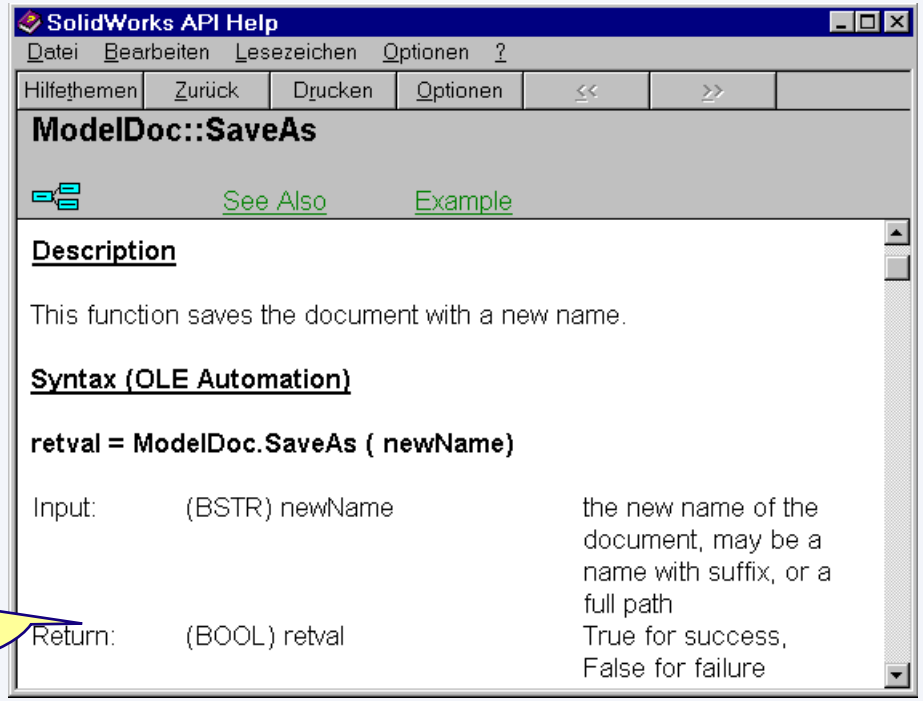
/ Programmierung

Verwendung der Online-Hilfe

Eine vollständige Online-Dokumentation zu API-Befehlen ist in der SolidWorks- Hilfe zu finden:

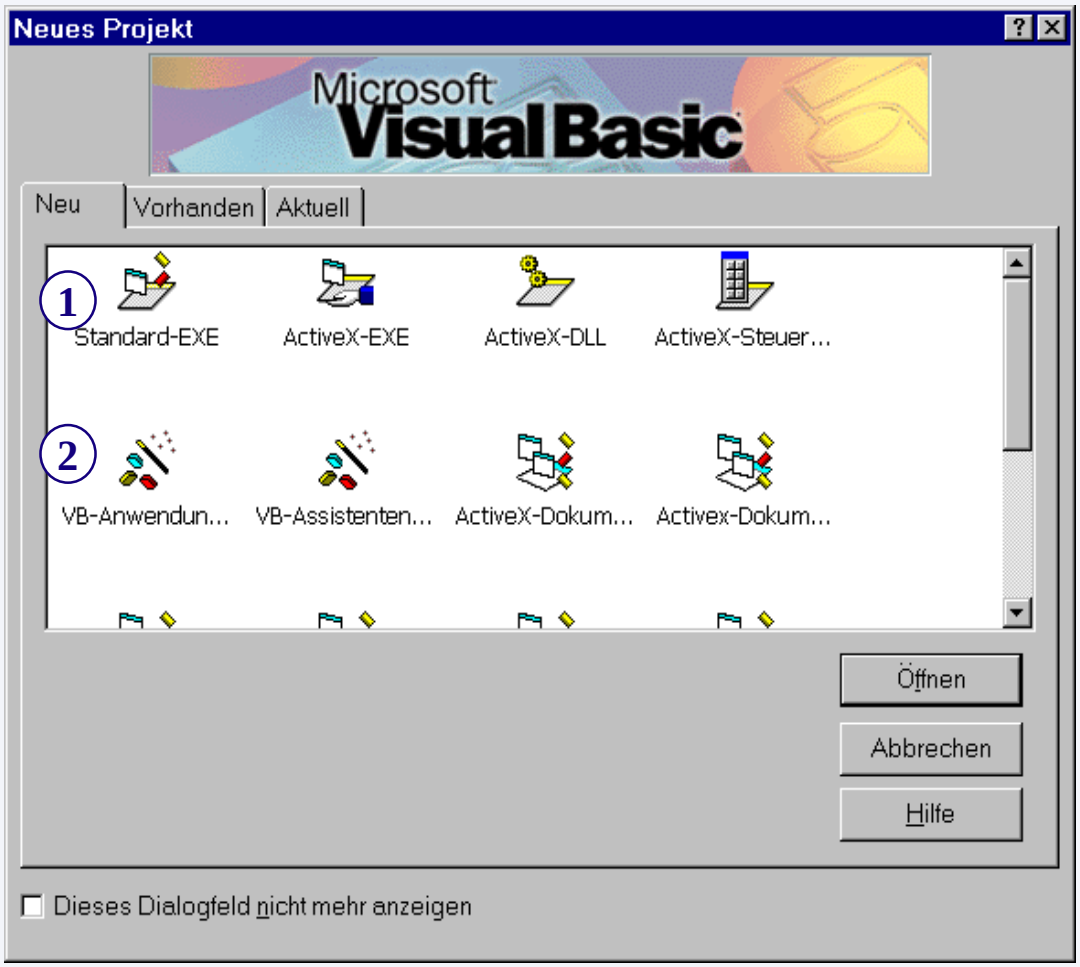






Starten der Anwendung MVB 6.0

Start, Programme, Microsoft Visual Studio, Microsoft Visual Basic 6.0:



1. Register Neu

Für (1) wird ein leeres Formular geöffnet.

Mit (2) wird der Assistent geöffnet, um eine neue Anwendung zu erstellen.

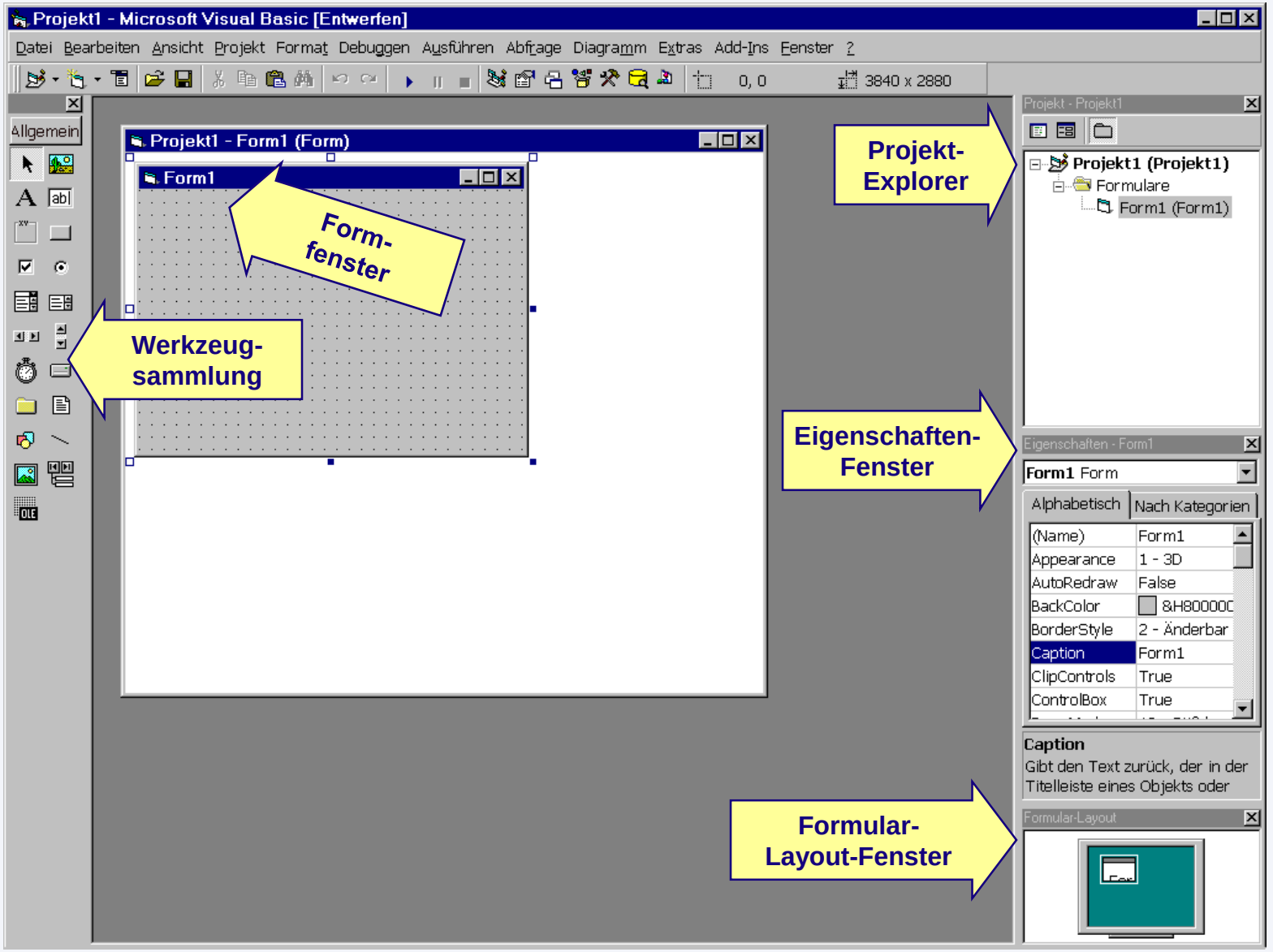
2. Register vorhanden

Für das Öffnen eines bestehenden Projektes

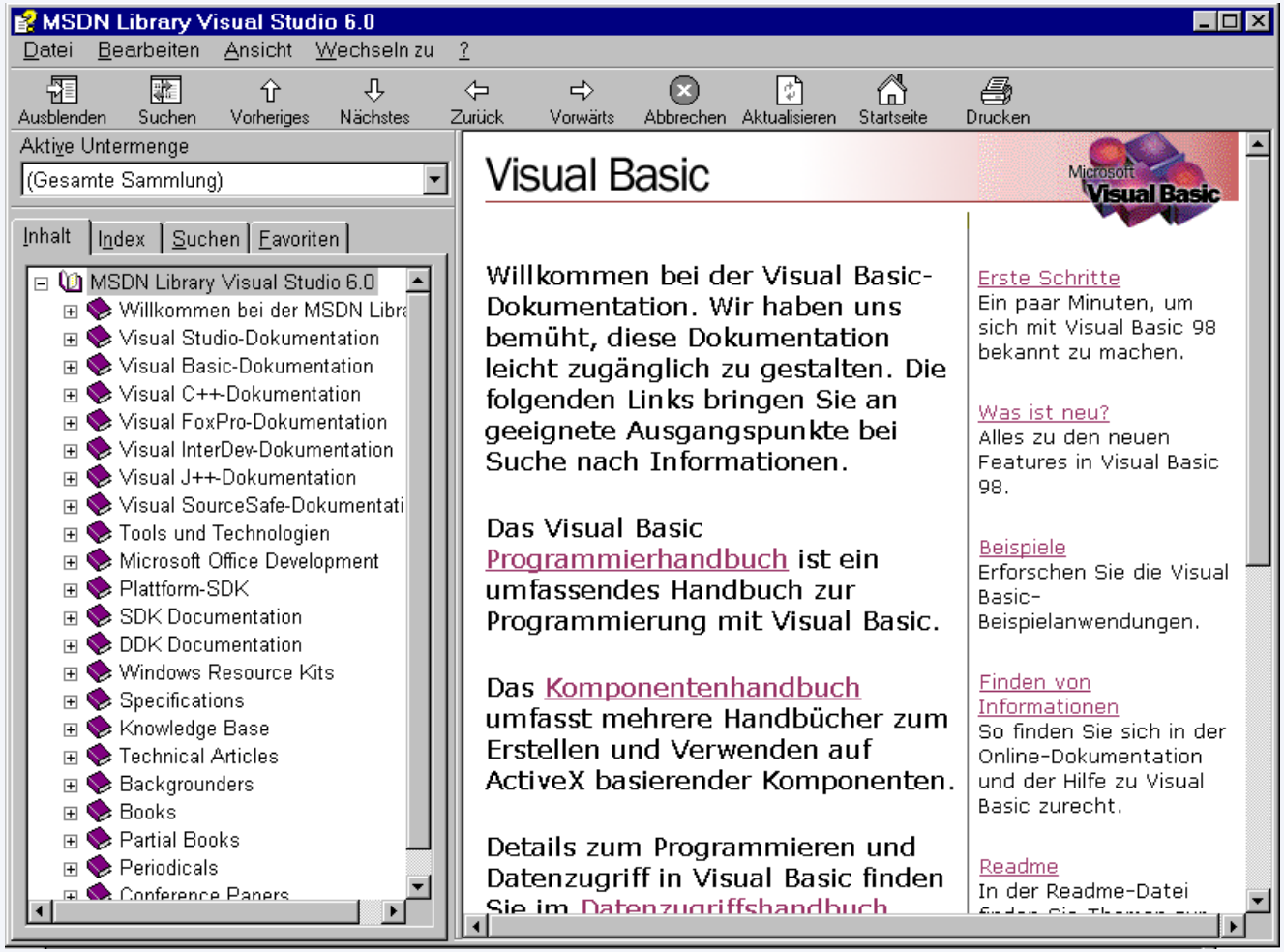
3. Register Aktuell

Liste der letzten geöffneten Projekte

Der Arbeitsbildschirm von Microsoft Visual Studio 6.0 (neues, leeres Formular)



Visual Studio Hilfe



1. Index

Im Register Index sind die Stichworte in alphabetischer Reihenfolge aufgeführt

2. Suchen

Mit der Suchfunktion kann bestimmte Begriffen innerhalb von Hilfetexten gesucht werden.

Schnelle Hilfe: Setzen des Cursors im Codefenster in die Anweisung zu der Hilfe benötigt wird und drücken der Taste F1. => Die Hilfe wird mit der Erklärung zu dem gewählten Wort geöffnet.

Visual Studio Projekt-Explorer

Im Projekt-Explorer sind alle zum Projekt gehörenden Dateien (Formen und Module) aufgelistet, die zur Entwurfszeit bearbeitet werden können und zum aktuellen Projekt gehören.

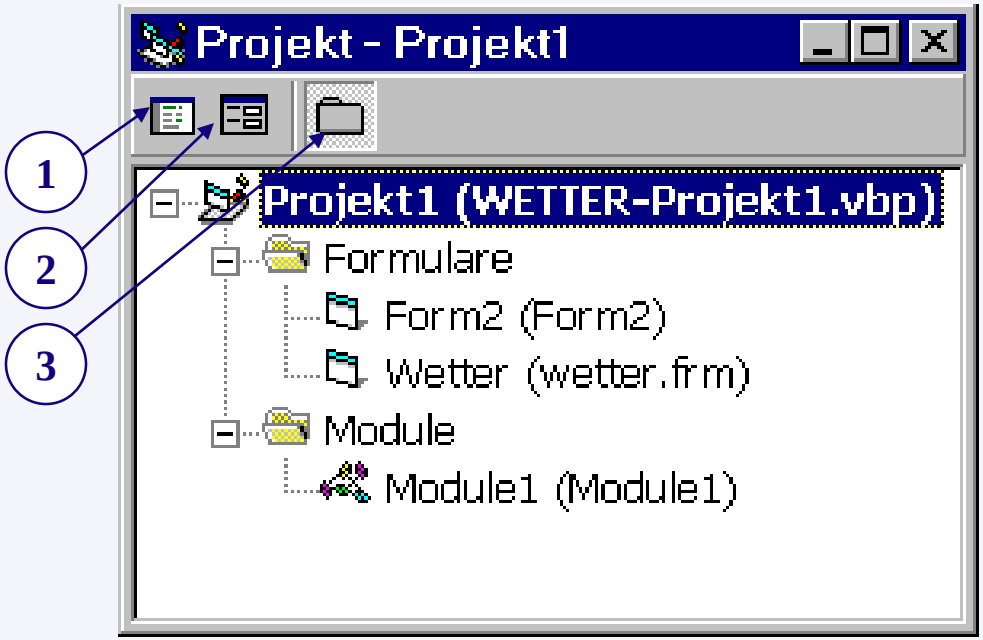
Die Liste wird in Baumform dargestellt. Rechts neben dem Namen, über den das Objekt später angesprochen werden kann, steht in Klammern der Name, unter dem das Element gespeichert wird. Die Ordnerstruktur ist unabhängig vom Dateisystem des Windows-Explorers.

Über die Symbolleiste kann zwischen den verschiedenen Ansichten umgeschaltet werden:

(1) Code anzeigen

(2) Objekt anzeigen

(3) Ordner wechseln



Visual Studio Form-Fenster

Im Form-Fenster werden alle Dialog- sowie Eingabe- und Ausgabefenster erzeugt, die für das Programm benötigt werden. Es werden z.B. Eingabefelder und Grafiken verwendet, die so angeordnet werden, wie sie bei der Programmausführung erscheinen sollen.

Visual Studio: Werkzeugsammlung

Zeiger

Label

Frame

CheckBox

ComboBox

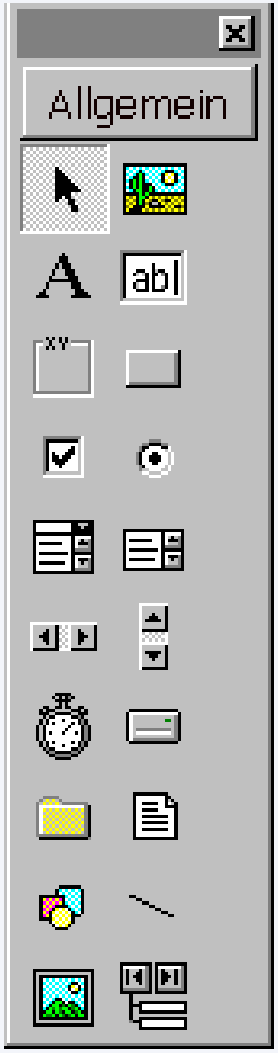
HScrollBar

Timer

DirListBox

Shape

Image



Picturebox

Textbox

CommandButton

OptionBox

ListBox

VScrollBar

DriveListBox

FileListBox

Line

Data

OLE

SSTab(Register)

CommonDialog

Toolbar

ProgressBar

...

Slider

...

MSFlexGrid

TabStrip(Register)

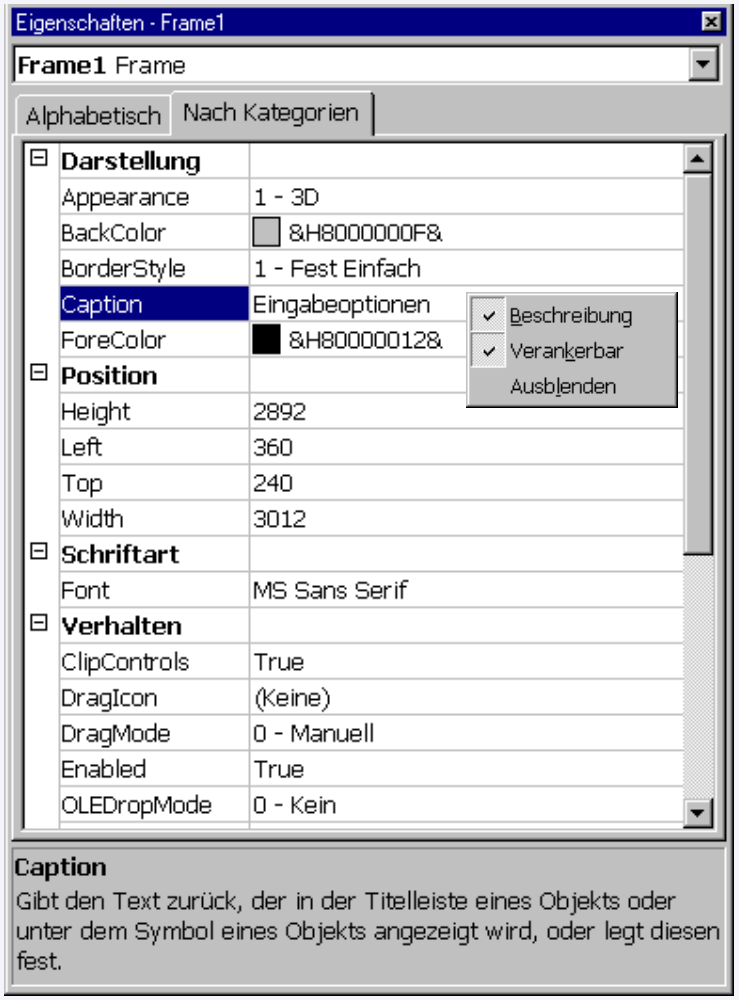
StatusBar

TreeView

...

Visual Studio: Eigenschaftenfenster

Im Eigenschaftenfenster werden alle zur Entwurfzeit einstellbaren Eigenschaften des gerade aktiven VB-Steuerelementes bzw. der aktiven Form aufgezeigt.
In der linken Spalte sind die Eigenschaften aufgelistet, in der rechten Spalte können die zugehörigen Werte angegeben bzw. verändert werden.



Über das Register können Eigenschaften entweder nach Kategorien oder alphabetisch sortiert angezeigt werden

Alle Steuerelemente, die einer Form hinzugefügt werden, besitzen verschiedene Eigenschaften, z.B. Farbe, Höhe oder Breite.

Im Eigenschaftenfenster wird im unteren Bereich ein Hinweistext eingeblendet, sofern dieser über ein Kontextmenü der rechten Maustaste eingeschaltet wird: Beschreibung

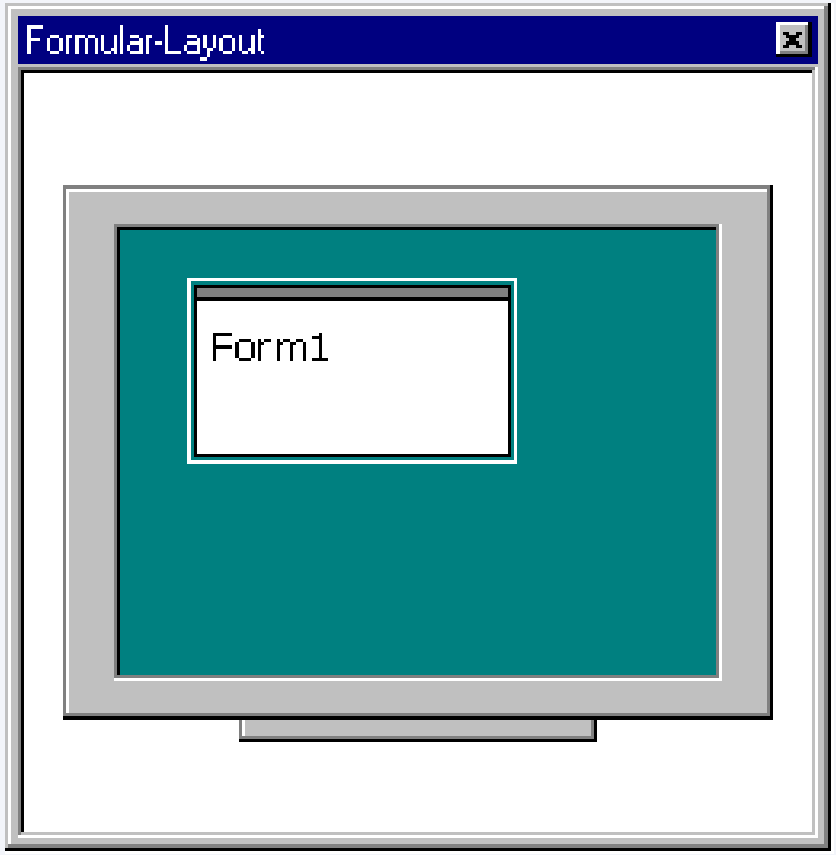
Überblick über wichtige Eigenschaften von Formen und Steuerelementen

Alignment:	**Text kann rechtsbündig, linksbündig oder zentriert ausgegeben werden**
BackColor:	**Die Hintergrundfarbe des Objektes kann verändert werden**
BorderStyle:	**Die Art der Objekt-Umrahmung kann verändert werden**
Cancel:	**Legt fest, ob es sich um die Schaltfläche abbrechen handelt**
Caption:	**Stellt die Beschriftung bzw. Überschrift eines O. dar**
DataField:	**Daten können mit einem Daten-Steuerelement verknüpft werden**
Default:	**Eine Schaltfläche ist ausgewählt und kann mit Return bestätigt werden**
DragIcon:	**Festlegen des Mauszeigers beim Ziehen eines Objektes**
DragMode:	**Wechseln zwischen automatischem und manuellen Ziehen eines Objektes**
Enabled:	**Aktiviert das Objekt, um z.B. auf Ereignisse zu reagieren**
Font:	**Textformatierung**
ForeColor:	**Schriftfarbe des Textes**

Interval:	**Der Abstand, wann ein Zeitgeber aktiviert wird**
Height:	**Höhe des Objektes**
HelpContextID:	**Erstellen des Hilfetext-nummer, die mit dem Objekt verknüpft ist**
Left:	**Position (X) der linken, oberen Ecke**
MousePointer:	**Aussehen des Maus-zeigers**
Name:	**Name, bzw. interne Bez.**
Picture:	**ein Bildfeld kann mit Inhalt versehen werden**
TabStop:	**Ein Steuerelement kann mit Tab angesprungen werden**
Text:	**Text des Steuerelem.**
Top:	**Position (Y) der linken, oberen Ecke**
Visible:	**Sichtbarkeit/ Unsicht-barkeit des Objektes**
Width:	**Breite eines Objektes**

Das Formular-Layout-Fenster

Beim Formular-Layout-Fenster handelt sich um eine stark verkleinerte Form der Bildschirm-darstellung. Die Positionen der Formulare auf dem Bildschirm können so überprüft werden. Ein Formular kann in diesem Fenster mit der Maus an die gewünschte Position gezogen werden. Genaue Positionierung der Fenster ist über die Angabe von Werten im Eigenschaftsfenster für die Eigenschaften Top und Left möglich.

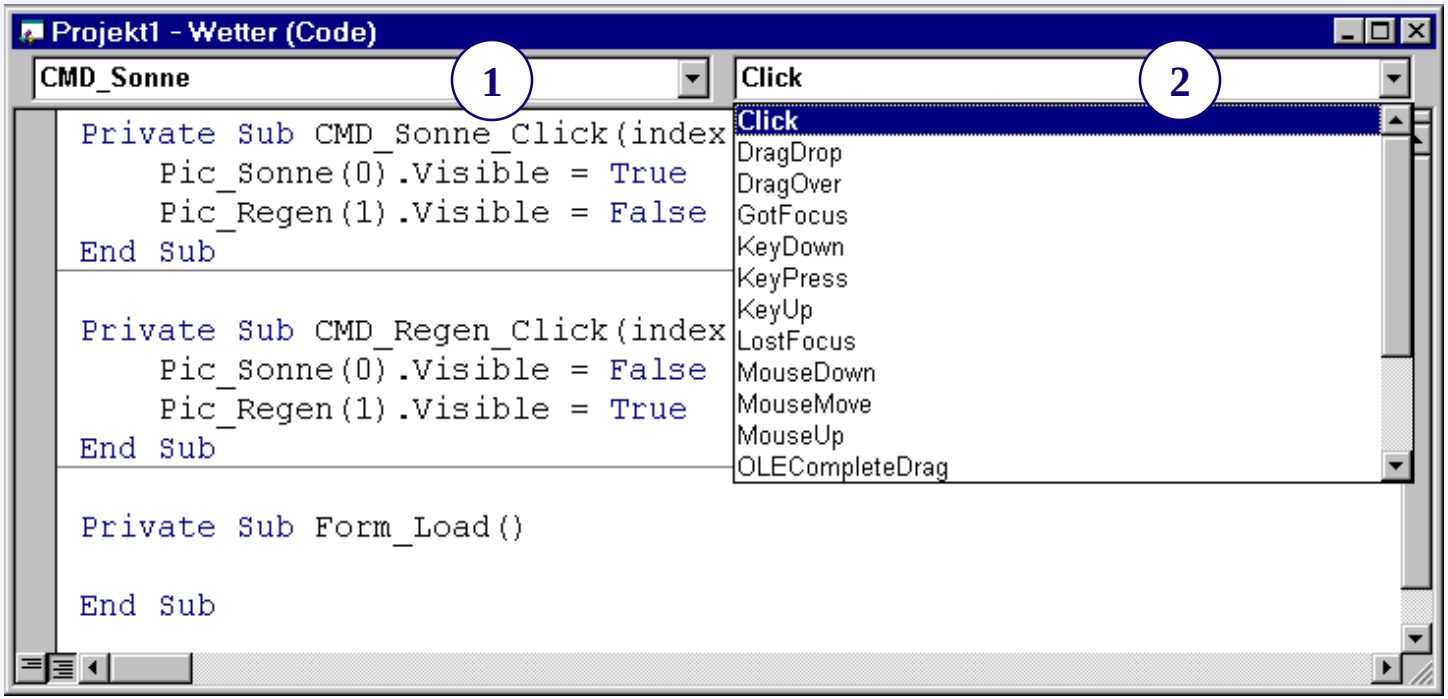


Das Code-Fenster für Ereignisprozeduren

Durch Doppelklick auf das Objekt (Steuerelement) im Formfenster wird das Codefenster automatisch geöffnet. Es wird standardmäßig eine Vorlage erstellt, die aus dem Anfang (*Sub*) und dem Ende (*End Sub*) einer Ereignisprozedur besteht. Dazwischen werden die Anweisungen eingegeben, die das Programm beim Eintreten dieses Ereignises ausführen soll.

Der Name der Prozedur wird automatisch hinter dem Wort *Sub* vergeben und besteht aus folgenden Teilen: *Objektname_Ereignis*

- Der Objektname ist im Eigenschaftsfenster unter *Name* vergeben bzw. voreingestellt
- Das Ereignis nach dem Unterstrich ist im Listenfeld 2 vorher ausgewählt, z.B. *Click*



/ Programmierung

Darstellung des VB-Codes

Im Codefenster wird der Visual Basic-Code standardmäßig wie folgt dargestellt:

Schlüsselwörter: VB-Schlüsselwörter, wie z.B. Sub und End Sub, werden blau angezeigt

Anweisungen: Befehle, Variablen, Konstanten usw. werden als schwarzer Text formatiert.

Fehlerhafter Code: Syntaktisch fehlerhafter Code, z.B. Prant statt Print wird rot dargestellt

Haltepunkte: Codezeilen, an denen Haltepunkte gesetzt wurden, sind braun unterlegt und zusätzlich durch einen Punkt am Rand gekennzeichnet

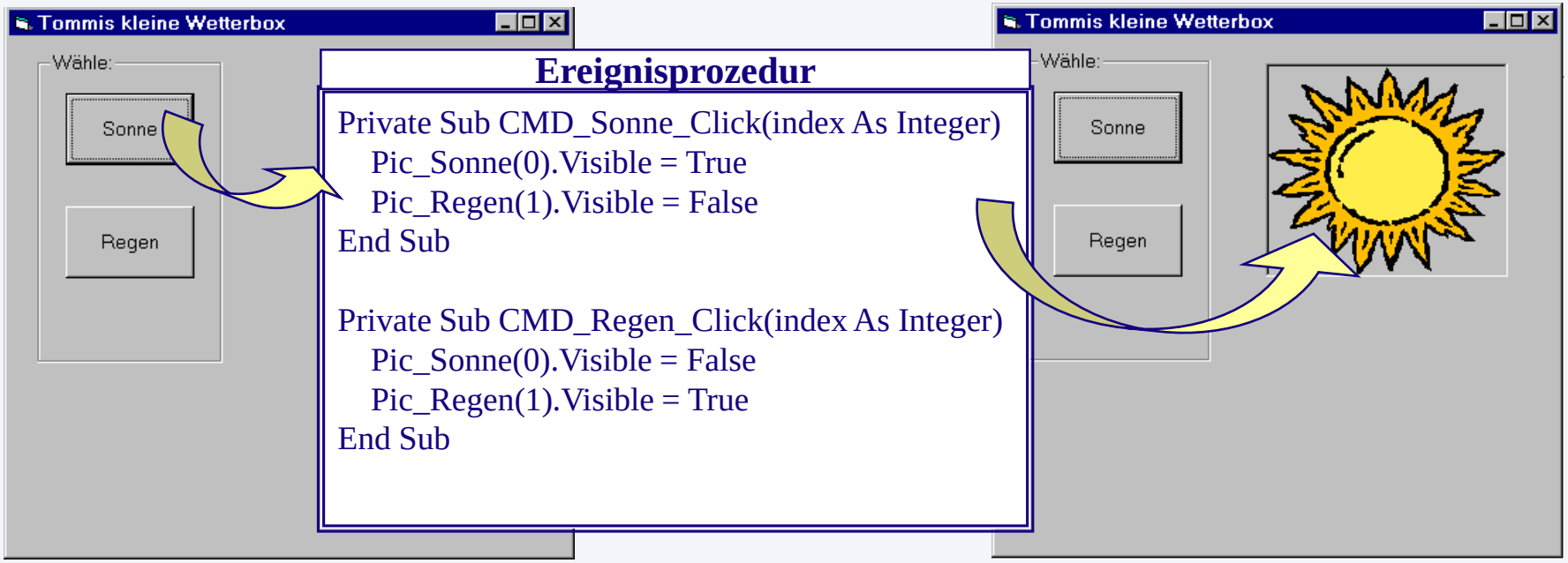
Kommentare: Kommentare werden durch ein vorangestelltes ‘ -Zeichen oder durch das Wort REM am Zeilenanfang gekennzeichnet, der nachfolgende Text wird nicht als Code kompiliert. Kommentare werden automatisch grün angezeigt. Beispiel:

```
Printer.Print  “Hallo Welt“   ‘ Ausgabe Hallo Welt auf Drucker
```

Ereignisgesteuerte Programmierung

VB-Programmierung wird als ereignisgesteuerte Programmierung bezeichnet. Das Ergebnis ist eine Aktion, die den Programmablauf beeinflußt. Alle Aktionen (Klick; Doppelklick; Verschieben, Öffnen und Schließen eines Fensters mit der Maus; Positionieren des Cursors in ein Eingabefeld mit der Tab-Taste) sind Ereignisse.

Über Ereignisse können interne Programmabläufe, z.B. Berechnungen, Öffnen und Schließen eines Fensters oder Abfragen von Uhrzeit oder Datum ausgelöst werden.



Vorgehensweise bei der Erstellung von Anwendungen

1. Problemstellung analysieren

- Informationen sammeln, Objekte festlegen und Programmablauf strukturieren

2. Benutzeroberfläche erstellen

- Anordnen der gewünschten Steuerelemente auf der Form

3. Eigenschaften festlegen

- Eigenschaften der Steuerelemente, z.B. Farben, Bezeichnungen oder Eingabefelder

4. Ereignisprozeduren codieren

- Aktion über Ereignis auf Steuerelement durch lineare Programmierung festlegen

5. Ausführen und Testen der Anwendung

6. Ausführbare Programmdatei erstellen

Ist die Anwendung ohne Fehler, kann eine Programmdatei erstellt werde, die unabhängig von Visual Basic ausgeführt und z.B: über den Programm-Manager aufgerufen werden kann.

Werte, Datentypen und Ausdrücke in Visual Basic

- Werte:
  - Numerische (Datentypen: Byte: 0..255; Boolean: 0, 1 bzw. TRUE, FALSE; Integer: -32768-32767);
    Long: -2.147.483.648 bis 2.147.483.647 ;
    Single: -3,37E+38 bis 3,37E+38 mit 7 Ziffern Genauigkeit;
    Double: -1,7E+308 bis 1,7E+308 mit 15 Ziffern Genauigkeit)
  - Alphanumerische und alphabetische Werte (Currency: -9,22E+14 bis 9,22E+14 mit 14 Stellen Genauigkeit, String: 0 bis 65635 Zeichen, Variant: beliebige Werte z.B. für Eingabefelder)
  - Datumswerte (Date: 1.1.100 (-657434) bis 31.12.9999 (+2958465)

- Ausdrücke:

Ein Ausdruck kann aus einem einzelnen Wert (z.B. Zeichenfolge oder Zahl) bestehen oder aus mehreren Werten gebildet werden, die durch Operatoren miteinander verknüpft sind. Ein Wert kann auch durch eine Funktion oder Variable geliefert werden.

Beispiel:

Ausdruck	Ergebnis
x = (10.34*2)	20,68
x = (“Hal“ & “lo“)	Hallo
x = (#21/01/00# + 9)	30.01.00

Variablen und Konstanten

Deklaration von Variablen: **1. Implizit** **(Deklaration durch direktes Einsetzen, ist zu vermeiden)**

2. Explizit **(empfohlen)**

Explizite Deklaration:

gebräuchlichste Form: **Dim *Variable1* As *Datentyp*, *Variable2* As *Datentyp*.....**

Merkmale:

- **Namen von Variablen dürfen max. 256 Zeichen lang sein und keine Schlüsselwörter wie z.B. *Sub*, *As*, *etc.* enthalten**
- **Wird hinter *As* kein Typ angegeben, so ist der Typ automatisch *Variant***
- **In einer Zeile können mehrere Variablen durch Komma getrennt deklariert werden. Für jede Variable muß ein Datentyp angegeben werden.**

Beispiele:

```
Dim Zahl_y as Double
Dim Text_x as String
Dim Var_v as Variant      oder:      Dim Var_v
Dim Zahl_Y, Zahl_Z as Double
```

Deklaration von Konstanten:

Syntax: **Const *Konstante1 = Ausdruck***

Merkmale: **Die Konstante bekommt den Datentyp des Ausdrucks**
Ein Ausdruck kann eine Zahl oder ein String, aber auch eine Berechnung sein

Wertzuweisungen

Eine Zuweisung übergibt den rechts von einem Gleichheitszeichen stehenden Ausdruck an die links stehende Variable. Die Variable übernimmt das Ergebnis dieses Ausdrucks als neuen Inhalt.

Syntax einer Zuweisung: **Variable = Ausdruck**
Zielobjekt.Eigenschaft = Ausdruck

Beispiele:

Zuweisung	Inhalt der Variablen nach der Zuweisung
Text_X = “Hallo Welt“	Hallo Welt
Text_X = Left(TextX, 4)	Hall
Zahl_Y = 12.5	12,5
Zahl_Y = 5*ZahlY/2	31,25
Var_V = #02-02-00#	02.02.02
Var_V = VarV + 12	14.02.02
Var_V = Date	*aktuelles Systemdatum*
Txt_Eingabe.Visible = True	Die Eigenschaft *Sichtbar* wird auf *Wahr* gesetzt

Operatoren

Zeichenverkettungsoperator: Text1 & Text2

Beispiel: Txt1 = “Hal“ & “lo“ & “ “ & “Welt“ => Txt1 = “Hallo Welt“

Mathematische Operatoren: +, -, *, /, \, MOD, ^

Beispiele:	**Division**	**x=5/3**	**=> x = 1,666**
	Ganzzahldivision:	**x=10\3**	**=> x = 3**
	Rest zu einem Vielfachen	**x= 10 MOD 3**	**=> x = 1**
	Potenzschreibweise	**x= 10^3**	**=> x = 1000**

Merkmale: Punkt vor Strich, Multiplikation vor Division vor Addition vor Subtraktion
Klammersetzung wird von innen nach außen ausgewertet

Vergleichsoperatoren: < , <= , > , >= , = , <> , Like

Beispiele:	**ZahlY < 5**	**Prüft, ob der Inhalt von ZahlY kleiner als 5 ist.**
	TextX = TextY	**Prüft, ob der Inhalt von TextX gleich dem von TextY ist.**
	TextX Like “?ü*	**Prüft, ob u.a. der 2. Buchstabe von TextX ein “ü“ ist.**

Logische Operatoren: And, Or, Xor, Eqv, Not

Beispiele:	**Bed1 Xor Bed2**	**Genau eine Bed. muß Wahr sein, damit das Erg. wahr ist.**
	Bed1 Eqv Bed2	**Das Ergebnis ist nur dann wahr, wenn die Bed. gleich sind**
	Not Bed1	**Ergebnis von Bed1 wird umgekehrt**

Prüfung und Umwandlung von Daten

Funktionen zur Datenprüfung (Auswahl):

VarType ( )	Liefert den Datentyp in Form eines Wertes: 2 = Integer, 3 = Long, 4 = Single, 5 = Double, 6 = Currency, 7 = Date, 8 = String,
IsDate ( )	Liefert Wahr, wenn das Argument in ein Datum umgewandelt werden kann.
IsEmpty ( )	Liefert Wahr, wenn die Variable noch nicht initialisiert wurde.
IsNumeric( )	Liefert Wahr, wenn das Argument in einen num. Datentyp umgewandelt werden kann

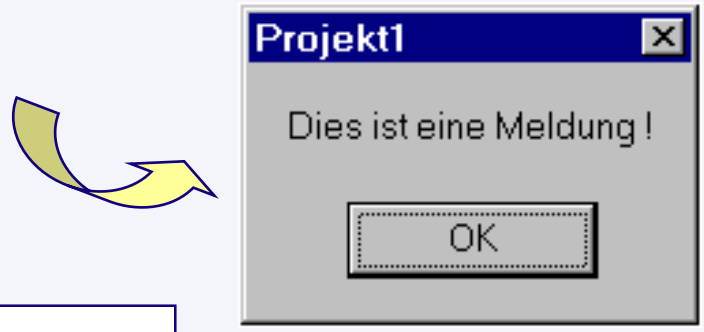
Funktionen zur Datenumwandlung (Auswahl):

CBool ( )	Wandelt in den Datentyp Boolean um.
CByte ( )	Wandelt in den Datentyp Byte um.
CCur ( )	Wandelt in den Datentyp Currency um.
CInt ( )	Wandelt in den Datentyp Integer um.
CDbl ( )	Wandelt in den Datentyp Double um.
Fix ( )	Wandelt durch Abschneiden nach dem Komma in Int. um: 8,4=> 8, -8.4=> -8
Int ( )	Wandelt durch Abschneiden nach dem Komma in Int. um: 8,4=> 8, -8.4=> -9

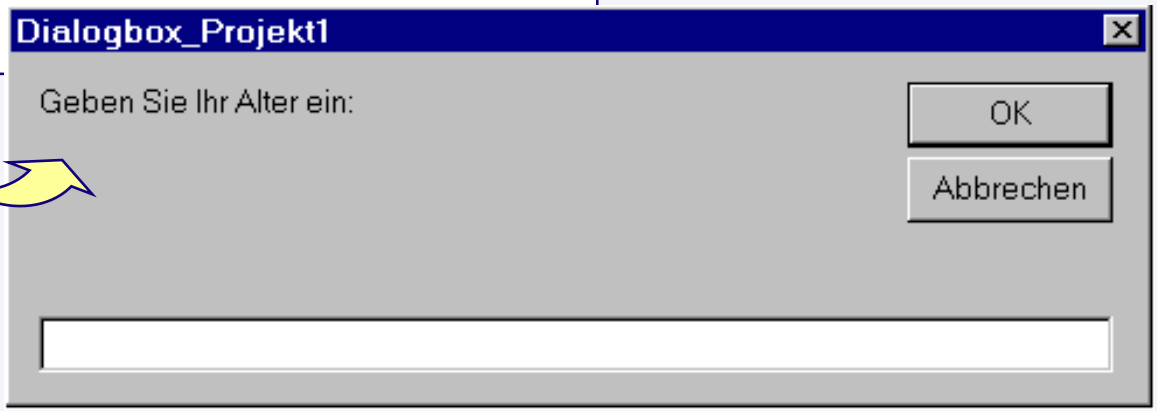
Messagebox und Inputbox

Aufruf in einer Ereignisprozedur (einfachste Form) und erzeugtes Fenster :

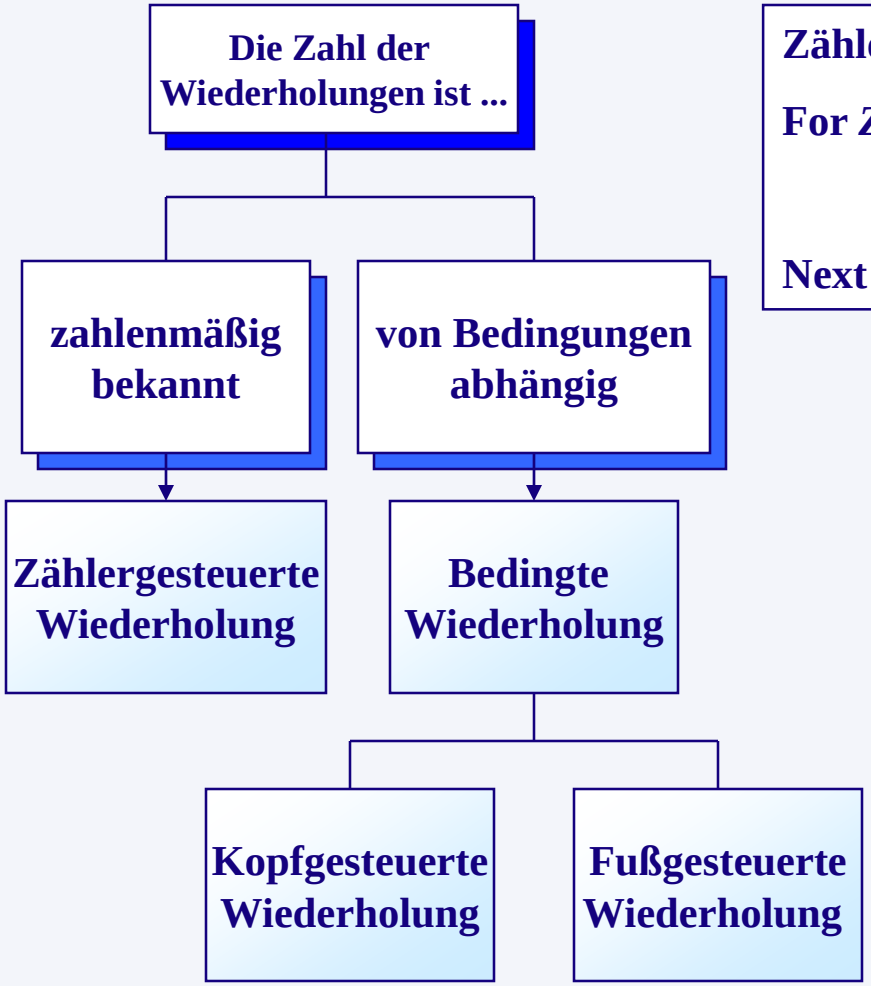
```
Private Sub Command1_Click()
MsgBox "Dies ist eine Meldung !"
End Sub
```




```
Private Sub Command2_Click()
Dim eingabe As Variant
eingabe = InputBox("Geben Sie Ihr Alter ein:")
MsgBox ("Ihre Eingabe: " & eingabe)

End Sub
```



Wiederholungskonstrukte



Zählergesteuerte Wiederholung:

For *Zaehlvar* = *Startwert* To *Endwert* Step *Schrittweite*

.....

.....

Next

Kopfgesteuerte Wiederholung:

Do While/Until *Bedingung*

.....

.....

Loop

Fußgesteuerte Wiederholung:

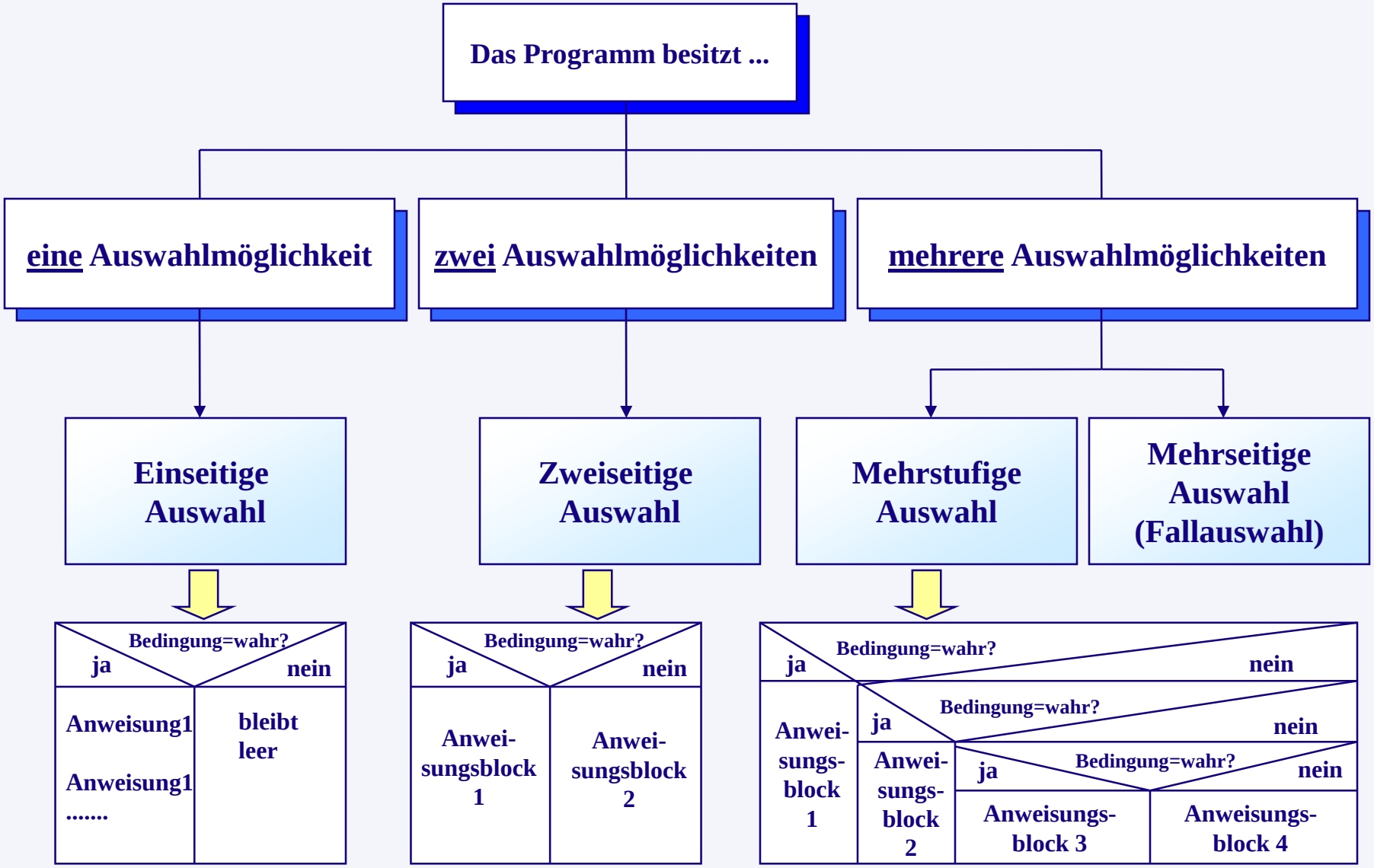
Do

.....

.....

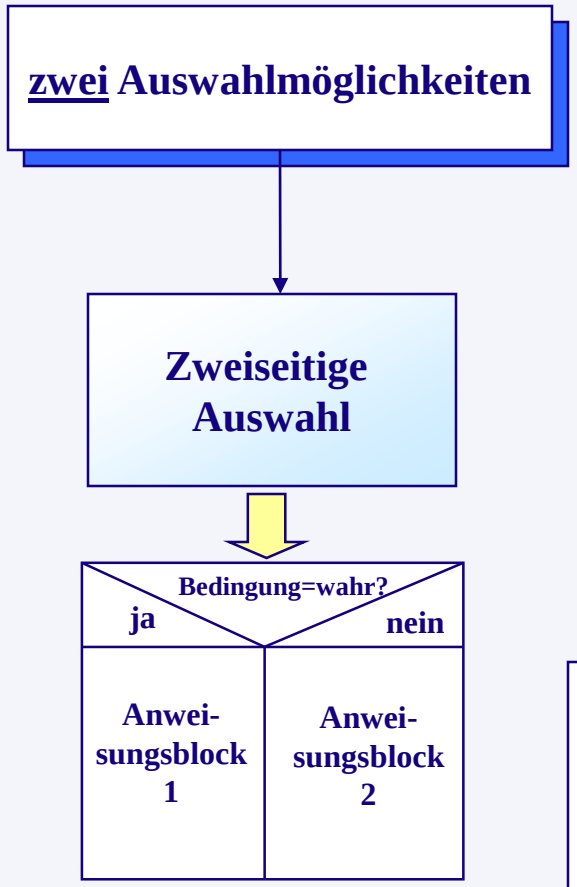
Loop While/Until *Bedingung*

Auswahlkonstrukte



/ Programmierung

Auswahlkonstrukte: Zweiseitige Auswahl



Syntax:

```
If Bedingung Then
        Anweisung1
        Anweisung2
        .....
Else
        Anweisung1
        Anweisung2
        .....
End If
```

Beispiel:

```
If Betrag <=0 Then
        Msgbox "Der Wert muß größer Null sein !"
Else
        Betrag = Sqr(Betrag)
        Msgbox ("Die Wurzel ist: ", Betrag)
End If
```

Beispiele: Prüfung über einseitige Auswahl

```
.....
Dim swApp As Object
Dim Part As Object
.......
Set swApp = CreateObject("SldWorks.Application")          ' SolidWorks-Aufruf
Set Part = swApp.ActiveDoc
swApp.Visible = True
If Part Is Nothing Then                                   ' kein Teil in SW geöffnet
        swApp.SendMsgToUser (“Kein Teil geöffnet")
        Exit Sub                                          ‘Verlassen der Ereignisprozedur
End If
If (Part.GetType <> swDocPART) Then
        swApp.SendMsgToUser (“Das ist kein Teil")
        Exit Sub                                          ‘Verlassen der Ereignisprozedur

End If
.......
```

Benutzerdefinierte Strukturen

Benutzerdefinierte Strukturen fassen (unterschiedliche) Datentypen übersichtlicher zusammen

Deklaration: **Type *Strukturname***

```
    Element1 As Datentyp
    Element2 As Datentyp
    Element3 As Datentyp
    Element4 As Datentyp
End Type
```

Beispiel: **Type Passfederstruk**

```
    Breite As Integer
    Hoehe As Integer
    Laenge As Integer
    Gehaertet As Boolean
    Material As String
End Type
```

Mit dieser Anweisung wird die Struktur nur als eigener Datentyp definiert. Vor der Anwendung muß erst eine Variable dieses Datentyps deklariert werden:

Dim Varstruktur As Strukturname **Beispiel:** **Dim Passfeder As Passfederstruk**

Merkmale:

- **Eine Struktur kann nur im allgemeinen Deklarationsteil einer Form bzw. eines Moduls deklariert werden.**
- **In einer Struktur können keine Objekttypen verwendet werden**
- **Die Deklaration einer Variablen dieses Strukturtyps kann überall erfolgen**

Verwendung der Struktur:

- **Übergeben eines Ausdrucks:** **Varstruktur.Element1 = 10**
- **Übergabe der kompletten Strukturdaten** **Neustruktur = Varstruktur**

(Neustruktur ist vom Typ Strukturname !)

Methoden im Bereich Skizzieren (Auswahl)

Einfügen einer Skizze: void ModelDoc.InsertSketch (), Beispiel: Part.InsertSketch

Erstellen einer Linie: retval = ModelDoc.CreateLineVB (P1, P2)
Beispiel: z_erfolg = Part.CreateLineVB (x1, y1, 0#, x2, y2, 0#)

Erstellen eines Bogens über 3 Punkte:
retval = ModelDoc.Create3PointArc (p1x, p1y, p1z, p2x, p2y, p2z, p3x, p3y, p3z)
Beispiel: Part.Create3PointArc 0.1, 0.0, 0, -0.1, 0, 0, 0, 0.1, 0

Erstellen eines Punktes: void ModelDoc.SketchPoint (x, y, z),
Beispiel: Part.SketchPoint 0.13, 0.01, 0

Auswahl von Elementen: retval = ModelDoc.SelectByID (selID, selParams, x, y, z)
Beispiel:
Part.SelectByID "Line1", "SKETCHSEGMENT", 0.02518833333333, 0.02, 0
Part.AndSelectByID "Line2", "SKETCHSEGMENT", 0.030226, -0.02, 0

Löschen der Auswahl von Elementen: void ModelDoc.ClearSelection ()
Beipiel: Part.ClearSelection

Bemaßen von Elementen: retval = ModelDoc.AddDimension (x, y, z)

Beispiel:
Part.ClearSelection
Part.SelectByID.....
Part.AndSelectByID.....
Part.AddDimension 0.0528955, 0.0762847, 0

Methoden im Bereich Dokument und Datei

Aufrufen von bzw. wechseln zu SolidWorks:

Set *Objektname* = CreateObject("SldWorks.Application") (*Objektname* z.B. swApp)

Setzt das bearbeitbare Teil auf das aktive Dokument in SolidWorks:

Set *Part* = *swApp*.ActiveDoc

Erzeugen eines neuen Teils in SolidWorks:

Set *Part* = *swApp*.NewPart

Laden eines Teils: ***retval* = *swApp*.OpenDoc (*Name*, Type)**

Aktivieren eines geladenen Teils: ***retval* = *swApp*.ActivateDoc (*Name*)**

Neuaufbau des aktiven Teils: **void *Part*.EditRebuild ()**

Speichern unter aktuellem Namen: **retval = *Part*.Save (*Name*)**

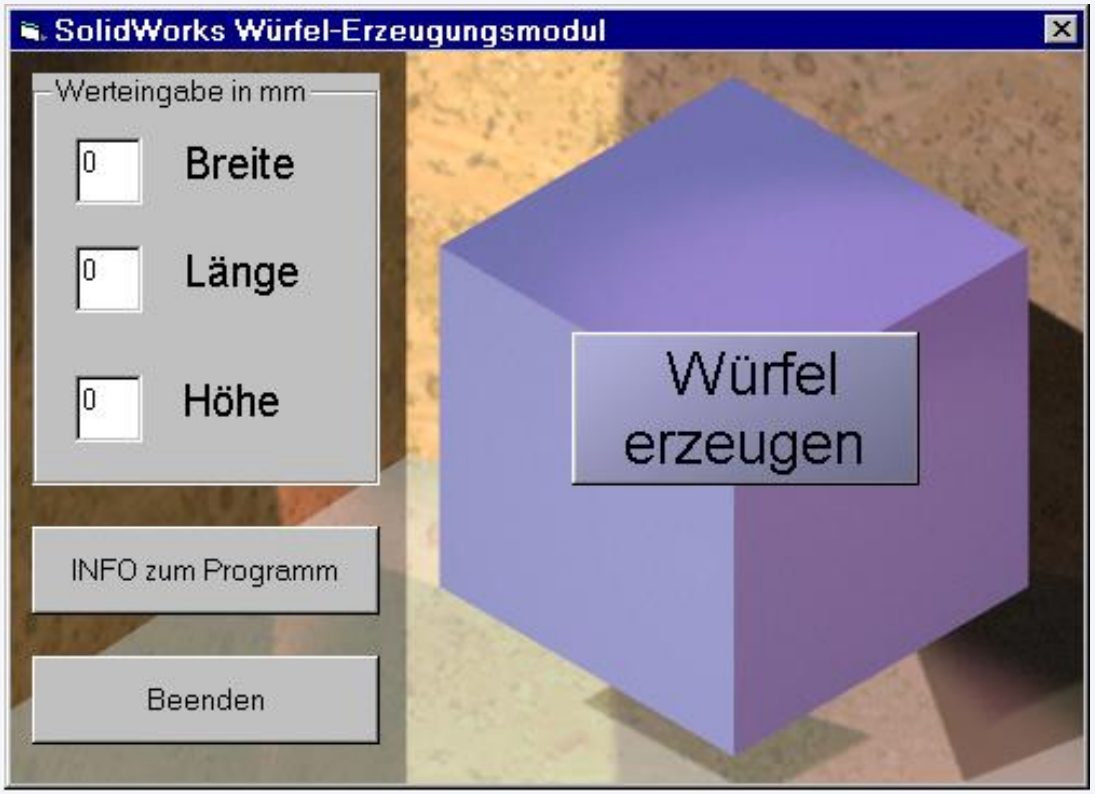
Speichern unter neuem Namen (neue Teile, z.B. “Part2“):

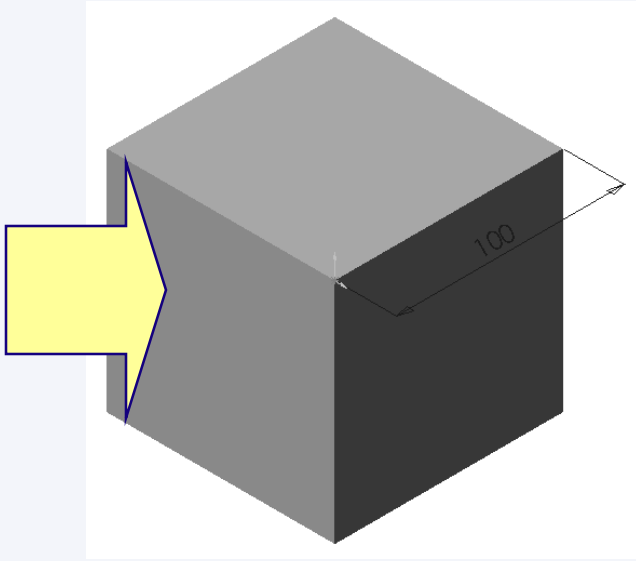
retval = *Part*.SaveAs (*Neuname*)

Hinweis: *kursive Namen* sind für Objekte frei wählbar

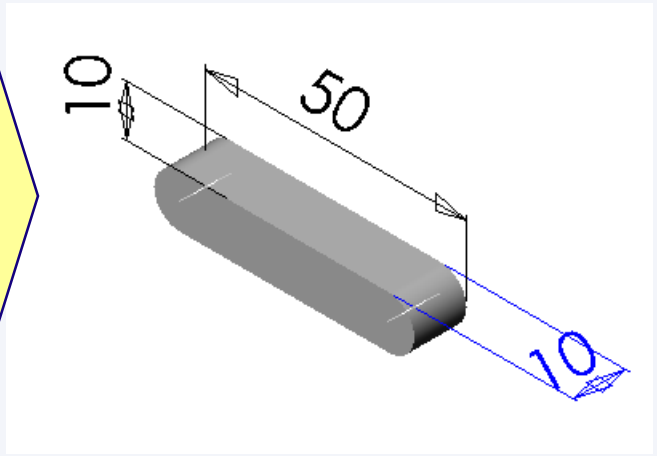
/ Programmierung

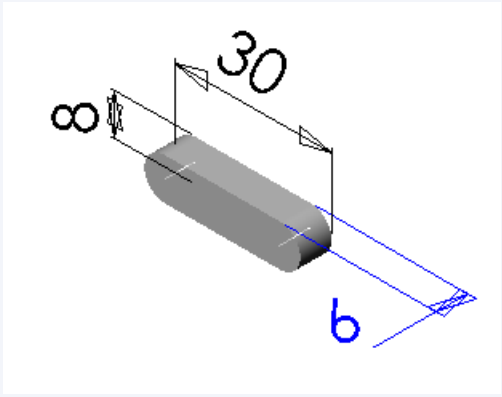
Projekt1: Erzeugen eines Würfels mit definierten Abmaßen





Projekt2: Erzeugen einer Passfeder durch Maßüberschreibung eines Vorgabeteils





Datenbanken

Grundlagen

Datenbanken

Datenbank: Eine geordnete Menge von Daten.

Speicherung erfolgt unabhängig von speziellen Anwenderprogrammen. Ebenso sollte die Hardwareunabhängigkeit gesichert werden.

Zu einem **Datenbankmanagementsystem** (DBMS) gehören neben den Daten u.a.:

- Abstraktionsmechanismen zur logischen Datenstrukturierung
- Komponenten zur Datenverwaltung *(Zugriff, Benutzer, Änderung, ...)*
- Kommunikationsschnittstellen (für Anwender und Anwendungsprogramme)

(*Transaktionen, Abfragen, Hilfsmittel für Eingabe und Präsentation, ..)*

Anforderungen an Daten und Funktionen*:*

- Redundanzfreiheit (keine Mehrfacheingaben,..)
- Integrität (Plausibilität, d.h. formal korrekt und nicht widersprüchlich)
- Steuerung des Datenzugriffs (Zugriffsrechte, Synchronisation des Zugriffs, ..)
- Absicherung der Daten (bei Transaktionen, System- und Anwendungsfehlern,...)

Grundbegriffe

Datenbanktechnologie:

die auf ein Datenbanksystem gestützte Verfahrensweise bei der Speicherung, Verwaltung und Nutzung von Informationen eines Anwendungsbereiches

Datenbanksystem (DBS):
Zusammenfassung von Datenbank und Datenbankmanagementsystem

Datenbank (DB):
Sammlung von Datenbeständen

Datenbankmanagementsystem (DBMS):
Softwaresystem zur Verwaltung und Bereitstellung von Datenbeständen

Datenmodell (DM):
Konzepte für die Abbildung von Diskursbereichen in Datenbanken
(Strukturen, Operationen, Integritätsbedingungen)

Datenbanksprache (DL):
Sprache zur Definition und Manipulation von Datenbeständen

Relationenmodell - Darstellungskonzepte

Begriffe:

- Relation
- Attribut
- Tupel
- Primärschlüssel
- Fremdschlüssel

Relation

- Menge (mathematisch) - hier nicht Beziehung
- Menge von Entitäten
- Gleiche Merkmale
- Kontextabhängige Mengenbildung
- Abbildung als **Tabelle**
- Beispiel: Student, Vorlesung, Professor

Attribut

- Ein Merkmal einer Entität (Komponente eines Tupels)
- Wert aus einem bestimmten Wertebereich
- Spalte einer Relation
- Keine Ordnung (Reihenfolge)
- Beispiel: Relation Student: { Name, Vorname, Matrikelnummer...}

Tupel

- Eine bestimmte Entität (Instanz)
- Zeile einer Relation
- Keine Ordnung (Reihenfolge)
- Beispiel: Relation Student: Name, Vorname, Matrikelnummer, Fachsemester, Studienrichtung

Primärschlüssel

- Ein oder mehrere Attribute
- Eindeutige Identifizierung eines Tupels
- Oftmals wird als Primäschlüssel ein zusätzliches Attribut verwendet (automatisch generierter Zähler)
- Beispiel: Relation Student: Matrikelnummer

Fremdschlüssel

- Beziehung zwischen Relationen
- Beziehung innerhalb einer Relation
- Fremdschlüssel ist Primärschlüssel in der referenzierten Relation

Relationen, Attribute, Tupel, Schlüssel

Attribut

Relation 1

Name	Vorname	Alter	Straßenschlüssel Fremdschlüssel	Haus-nummer
Müller	Ida	22 Attributwert	102	3
Müller	Maria	71	104	7
Schmitz	Anton	43	102	5

Tupel

Primärschlüssel, zusammengesetzt

Relation 2

Straßenschlüssel	Straßenname
101	Nordstraße
102	Hauptplatz
103	Eichenweg
104	Inselstraße

Primärschlüssel, einfach

Beispielrelationen 1

Primärschlüssel

Fremdschlüssel

Assistent:

Persnr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2134

Professor:

Persnr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Student:

Matrnr	Name	Sem
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

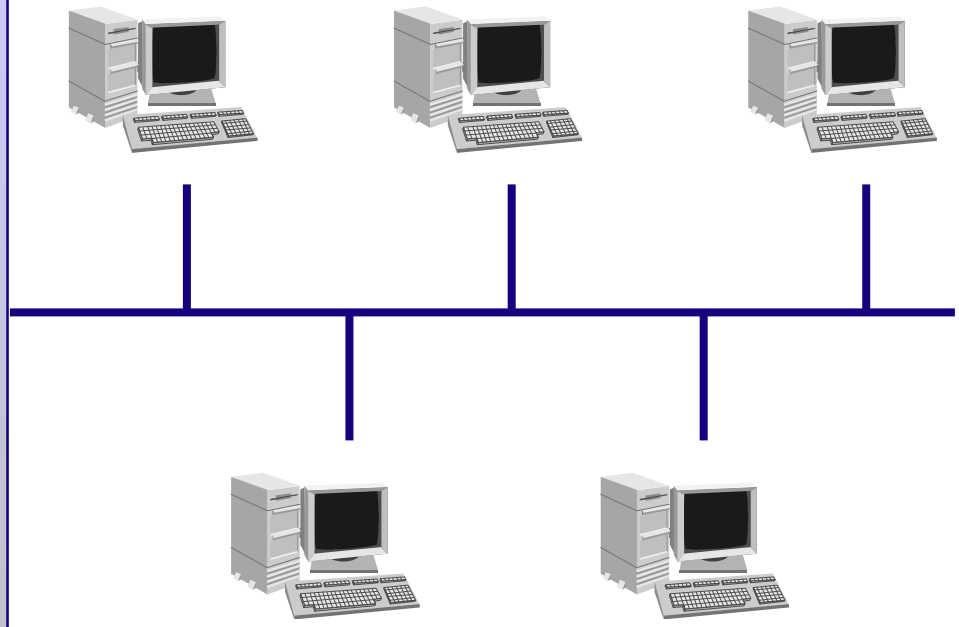
Inhalt

- Einführung
- Grundbegriffe der Objektorientierten Programmierung
- Grundlagen der Programmierung in Visual Basic
- Makro-Technik
- Einbinden von Makros in Visual Studio
- Anwendungsprogrammierung in SolidWorks
- Programmierung in C++ / C#
- Web-Programmierung

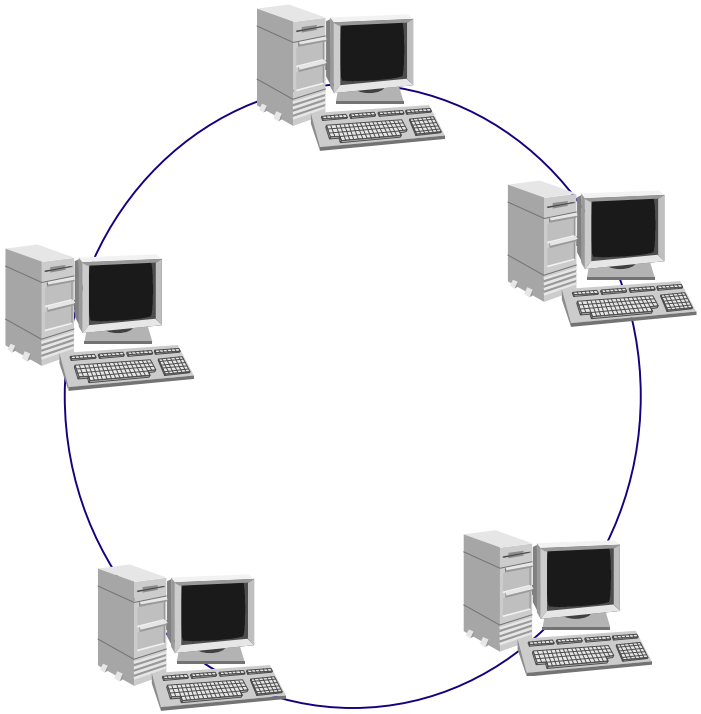
Netzwerktechnik

Grundlagen

Netztopologien (1)

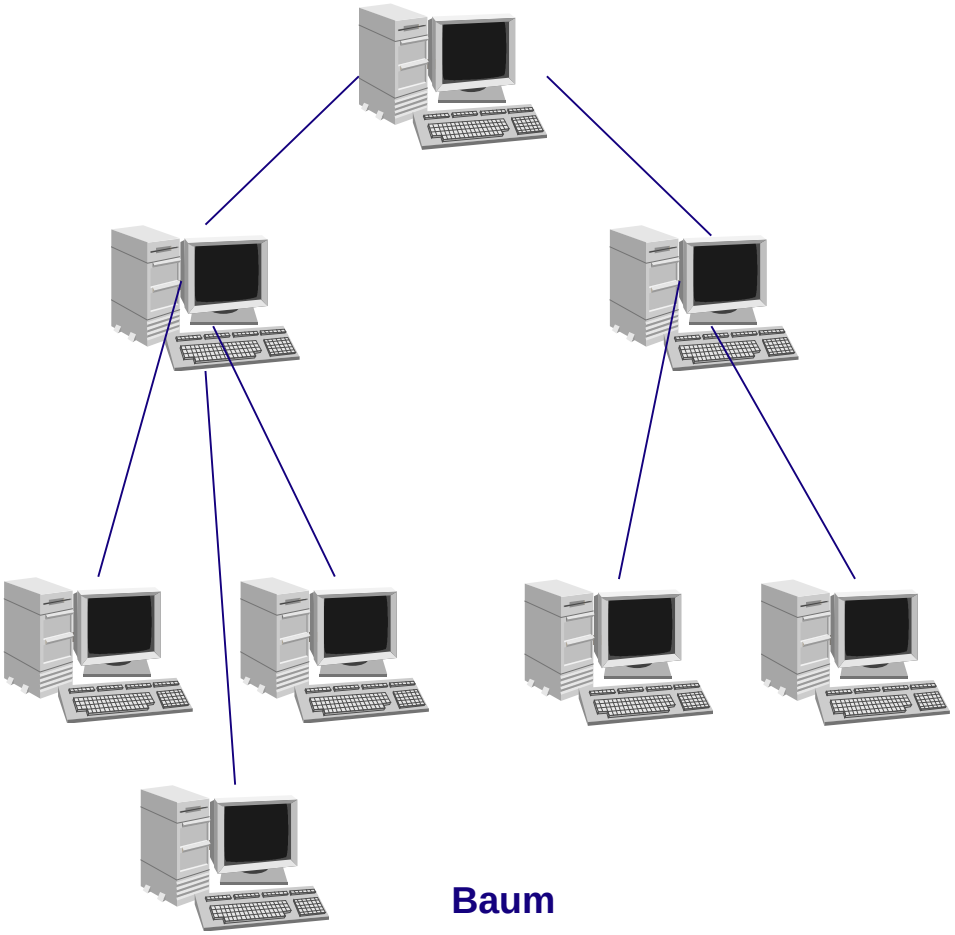


Bus

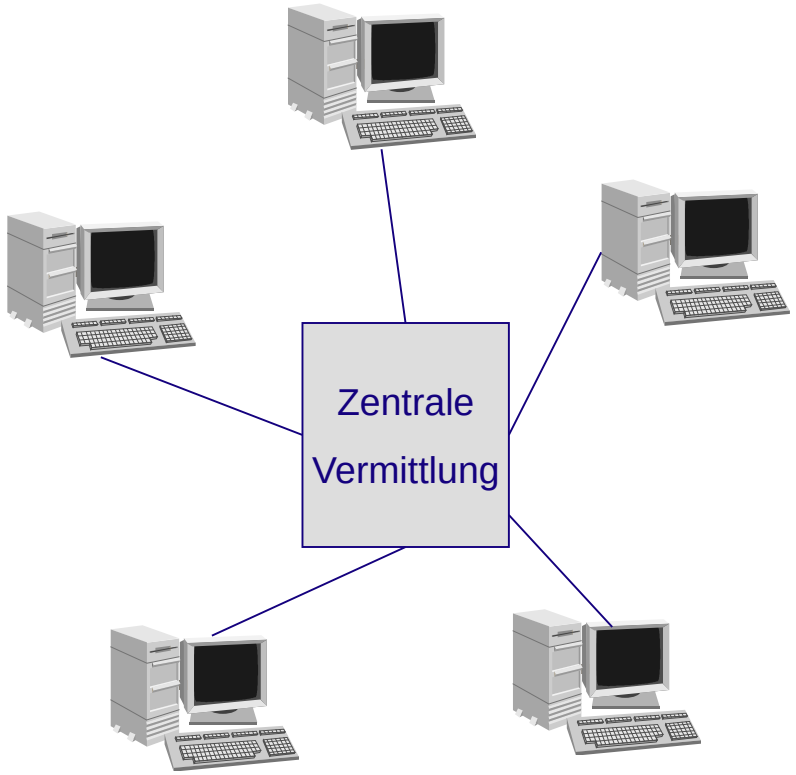


Ring

Netztopologien (2)

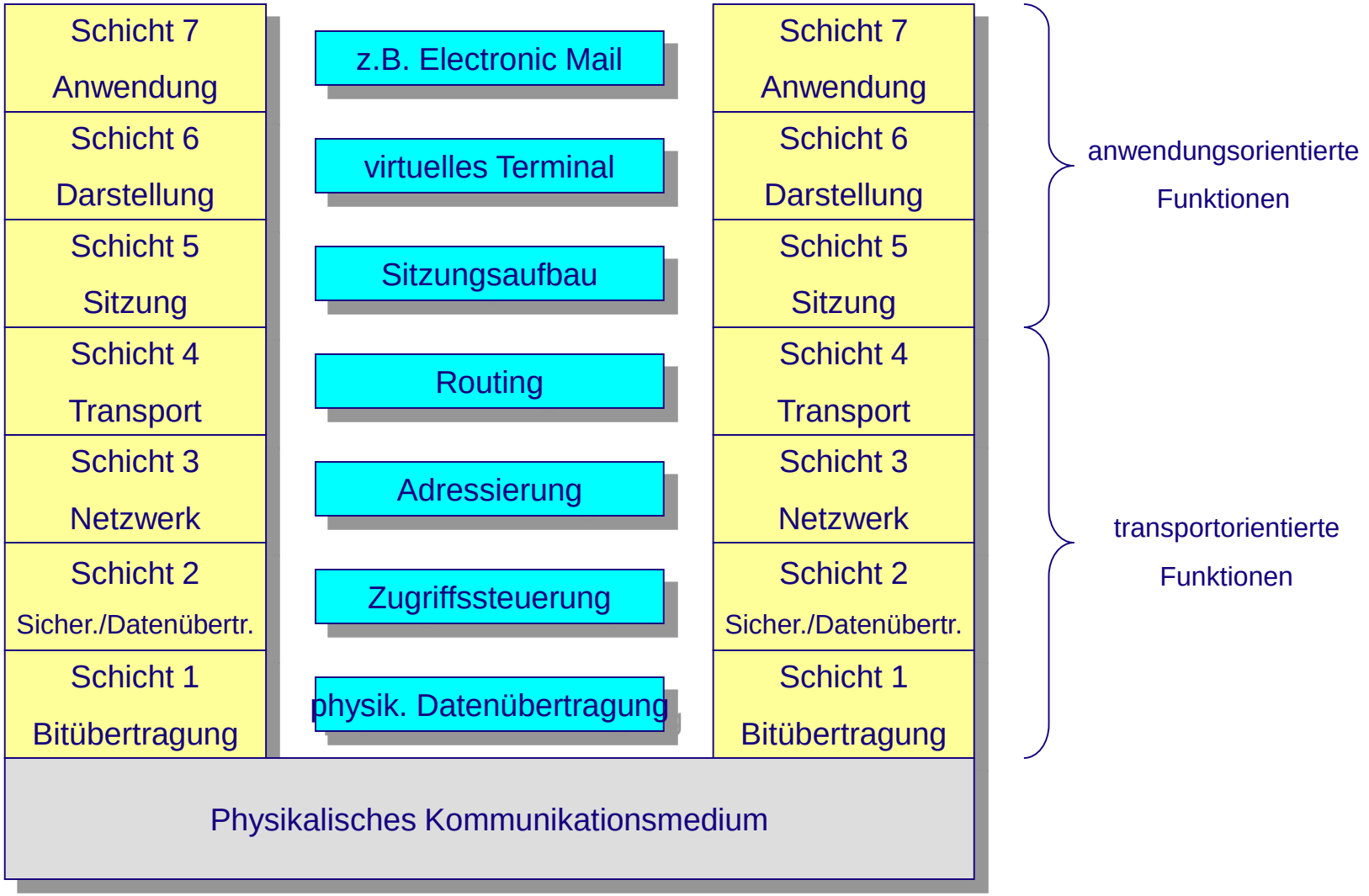


Baum



Stern

ISO/OSI-Referenzmodell



TCP/IP - OSI Referenzmodell

OSI-Schicht	TCP/IP-Schicht	Beispiel
Anwendungen (7)	Anwendungen	HTTP, UDS, FTP, SMTP, POP, Telnet, OPC UA
Darstellung (6)		
Sitzung (5)		
		SOCKS
Transport (4)	Transport	TCP, UDP, SCTP
Vermittlung (3)	Internet	IP (IPv4, IPv6), ICMP (über IP)
Sicherung (2)	Netzzugang	Ethernet, Token Bus, Token Ring, FDDI, IPoAC
Bitübertragung (1)		

OSI: Open Systems Interconnection

Quelle: Wikipedia

OSI Referenzmodell

Anwendungsschicht

Die Anwendungsschicht (engl.: *Application Layer*) umfasst alle Protokolle, die mit Anwendungsprogrammen zusammenarbeiten und die Netzwerkinfrastruktur für den Austausch anwendungsspezifischer Daten nutzen.

Transportschicht

Die Transportschicht (engl.: *Transport Layer*) ermöglicht eine Ende-zu-Ende-Kommunikation. Das wichtigste Protokoll dieser Schicht ist das Transmission Control Protocol (TCP), das Verbindungen zwischen jeweils zwei Netzwerkteilnehmern zum zuverlässigen Versenden von Datenströmen herstellt. Es gehören aber auch unzuverlässige Protokolle – zum Beispiel das User Datagram Protocol (UDP) – in diese Schicht.

Internetschicht

Die Internetschicht (engl.: *Internet Layer*) ist für die Weitervermittlung von Paketen und die Wegewahl (Routing) zuständig. Auf dieser Schicht und den darunterliegenden Schichten werden Direktverbindungen betrachtet. Die Aufgabe dieser Schicht ist es, zu einem empfangenen Paket das nächste Zwischenziel zu ermitteln und das Paket dorthin weiterzuleiten. Kern dieser Schicht ist das Internet Protocol (IP) in der Version 4 oder 6, das einen Paketauslieferungsdienst bereitstellt. Sogenannte Dual-Stacks können dabei automatisch erkennen, ob sie einen Kommunikationspartner über IPv6 oder IPv4 erreichen können und nutzen vorzugsweise IPv6. Dies ist für entsprechend programmierte Anwendungen transparent. Die Internetschicht entspricht der Vermittlungsschicht des ISO/OSI-Referenzmodells.

Netzzugangsschicht

Die Netzzugangsschicht (engl.: *Link Layer*) ist im TCP/IP-Referenzmodell spezifiziert, enthält jedoch keine Protokolle der TCP/IP-Familie. Sie ist vielmehr als Platzhalter für verschiedene Techniken zur Datenübertragung von Punkt zu Punkt zu verstehen. Die Internet-Protokolle wurden mit dem Ziel entwickelt, verschiedene Subnetze zusammenzuschließen. Daher kann die Host-an-Netz-Schicht durch Protokolle wie *Ethernet*, *FDDI*, *PPP* (Punkt-zu-Punkt-Verbindung) oder *802.11* (*WLAN*) ausgefüllt werden. Die Netzzugangsschicht entspricht der Sicherungs- und Bitübertragungsschicht des ISO/OSI-Referenzmodells.

Hypertext Transfer Protocol (HTTP)

- Protokoll zur Übertragung von Daten über ein Netzwerk.
- Anwendungsschicht
- Hauptsächliche Verwendung: Webseiten im World Wide Web (WWW) mit Hilfe eines Browsers anzeigen.
- HTTP ist vom Konzept her ein zustandsloses Prpotokoll. (Nach der Datenübertragung wird keine Verbindung zwischen den Kommunikationspartnern aufrecht erhalten).
- In den meisten Fällen verwendet HTTP als Transportprotokoll TCP.
- Durch ständige Erweiterung der Anfragemöglichkeiten wird der Anwendungsbereich stetig erweitert. (Nicht nur Hypertext)

HTTPS

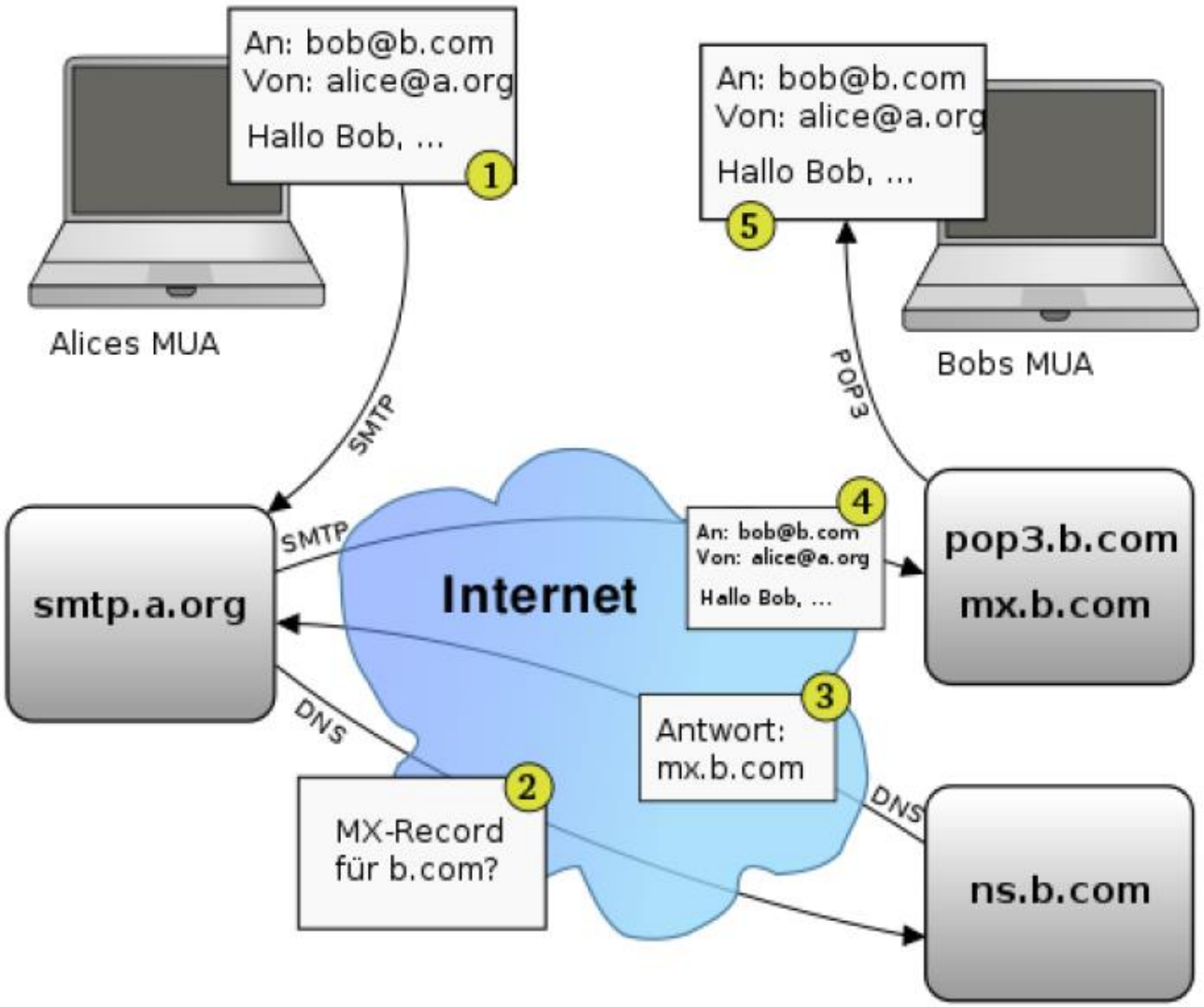
Hypertext Transfer Protocol Secure (HTTPS)

- Protokoll zur abhörsicheren Übertragung von Daten über ein Netzwerk
- Aufbauend auf HTTP (Zusätzliche Schicht zwischen TCP und HTTP)
- Erweiterung um Authentifizierung und Verschlüsselung
- Keine separate Sicherheitssoftware erforderlich
- Verschlüsselung durch SSL (Secure Sockets Layer)

Email

- In einem Computernetzwerk übertragene briefartige Nachricht.
- Meistgenutzter Dienst des Internets
- Format einer Email: RFC 2822 – Textzeichen (7 Bit ASCII-Zeichen)
- Inhalt: Header, Body durch eine Leerzeile getrennt.
- Ein Dateianhang (Attachment) ist eine Datei, die im Body der Email verschickt wird. Dazu muss das MIME Protokoll (Multipurpose Internet Mail Extensions) implementiert sein.
- Protokolle zum Versenden von Email:
  - SMTP – Simple Mail Transfer Protocol, Transport und Versand von Emails
  - POP3 – Point of Presence, Abrufen von Emails von einem Mailserver
  - IMAP – Internet Message Access Protocol, Zugriff auf Daten, die auf einem Mailserver liegen. (Im Gegensatz zu POP werden die Daten auf dem Server belassen.)

Email


An: bob@b.com
Von: alice@a.org
Hallo Bob, ...
1
Alices MUA
SMTP
SMTP
smtp.a.org
Internet
DNS
2
MX-Record
für b.com?
3
Antwort:
mx.b.com
4
An: bob@b.com
Von: alice@a.org
Hallo Bob, ...
pop3.b.com
mx.b.com
POP3
5
An: bob@b.com
Von: alice@a.org
Hallo Bob, ...
Bobs MUA
DNS
ns.b.com

File Transfer Protocol (FTP)

- Protokoll zur Übertragung von Dateien (RFC 959).
- Client / Server Prinzip
- Steuerung jeweils über Verbindungen.
- Ablauf: Client eröffnet eine TCP-Verbindung und sendet Befehle bzw. Anfragen, die jeweils vom Server beantwortet werden.
- Zugang entweder durch Anmeldung mit Benutzername und Kennwort oder
- Anonymes FTP: Öffentlich zugängliche Bereiche eines FTP-Servers. (Anmeldung als User „anonymous“, Passwort ist die Email-Adresse)

Teilnetzwerke

- **Internet:**

  Internet bezeichnet ein öffentliches und offenes Netz.

- **Intranet**

  Ein Intranet ist ein Netz, welches nur für eine geschlossene Benutzergruppe zugänglich ist. (Beispielsweise ein unternehmensinternes Netz). Ein Intranet verwendet die gleiche Technik (Protokolle) wie das Internet.

- **Extranet**

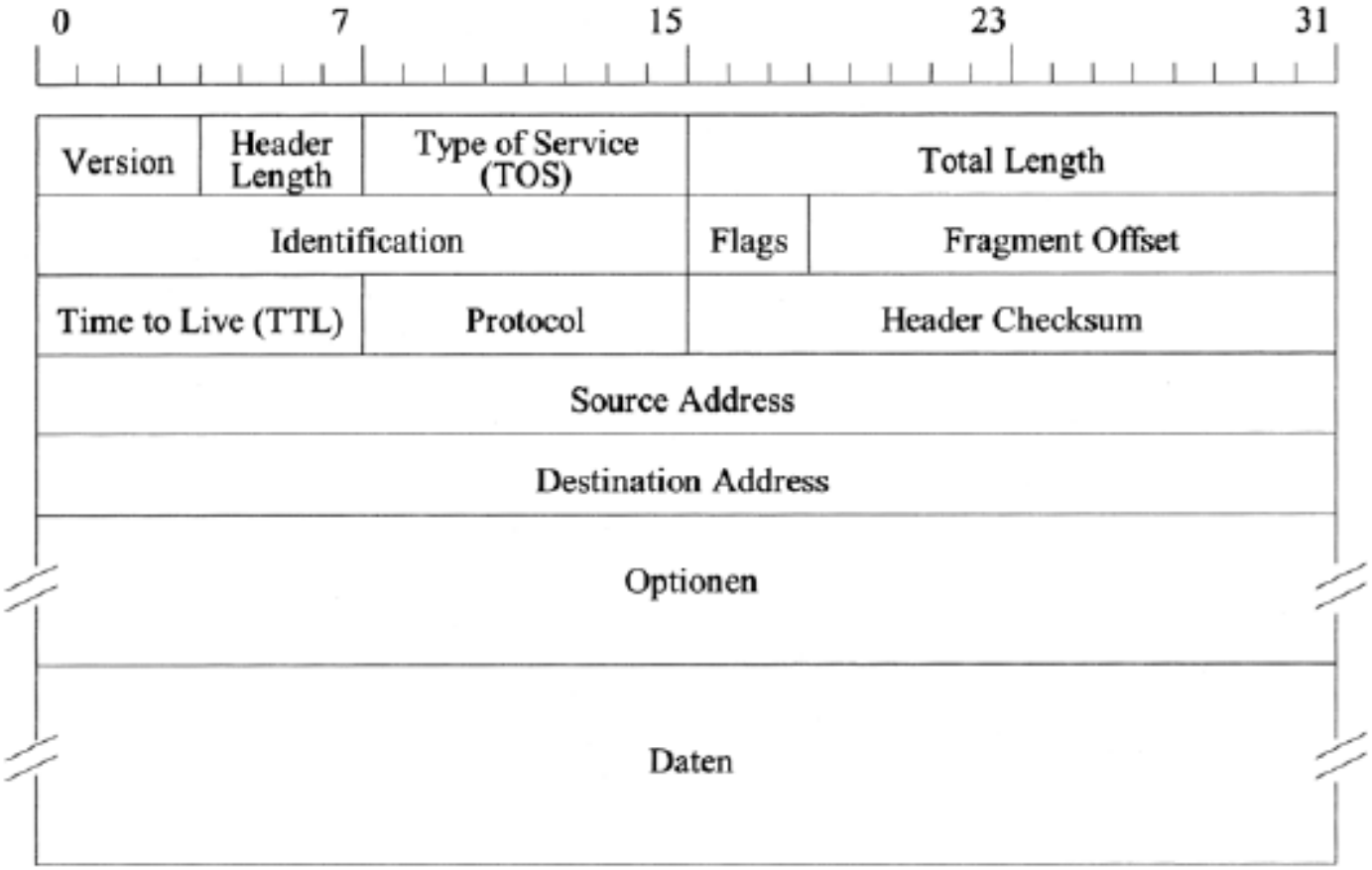
  Extranet bezeichnet ein Intranet, welches zusätzlich ausgewählten Partnern (Kunden, ext. Mitarbeiter etc.) einen Zugang ermöglicht.

TCP/IP

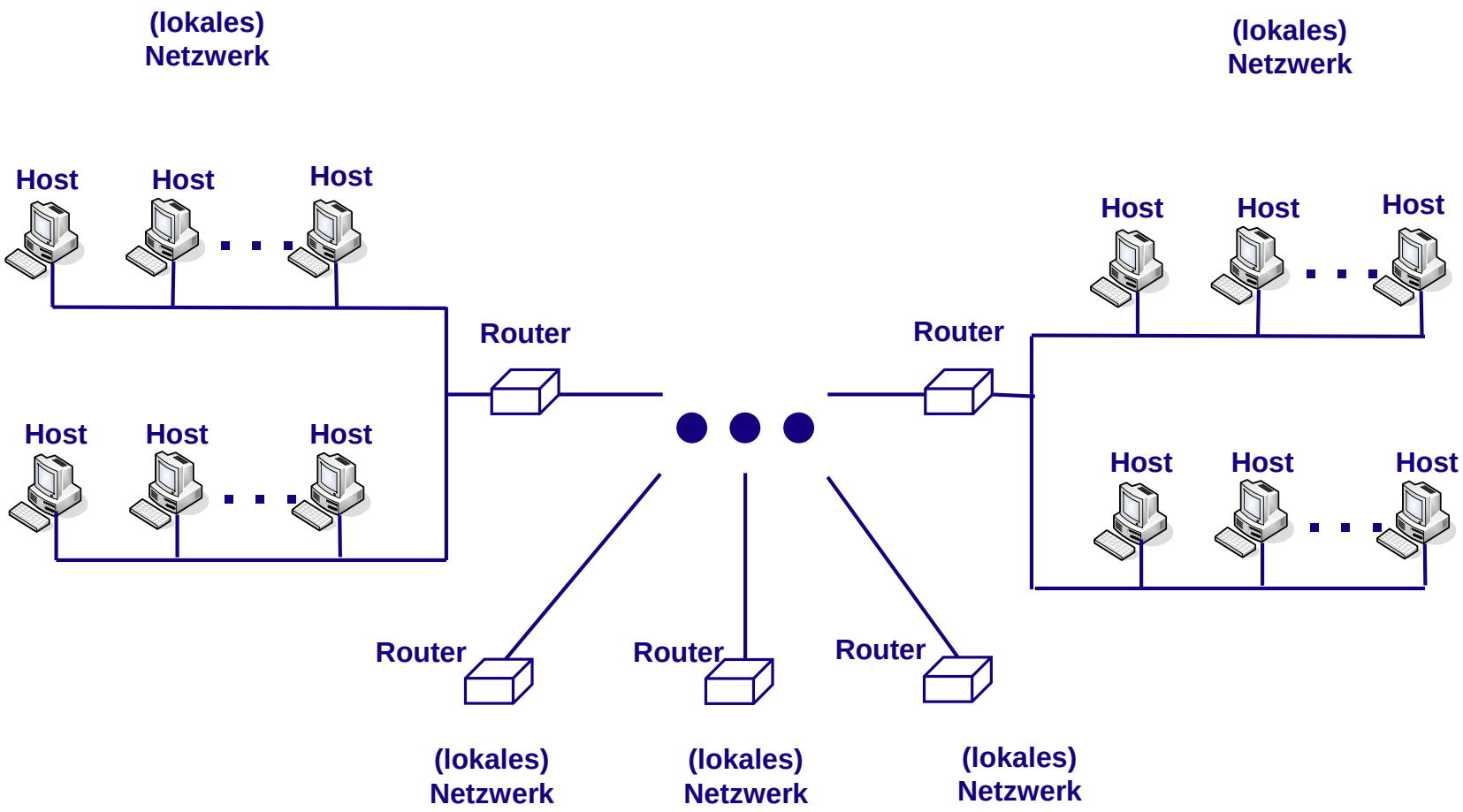
- Die Grundlage für die Kommunikation im Internet stellt das Internet Protokoll IP (Internet Protocol) dar.
- Basierend auf der Vermittlung von (Daten-)Paketen mit variabler Länge.
- Jedes Paket enthält eine eindeutige Adresse (Absender und Empfänger)
- Verbindungsloses Protokoll. Pakete werden von **Routern** bis zum Empfänger weitergeleitet.
- Prinzip: Ein Endgerät (Host) in einem lokalen Netz ist über einen Router mit anderen Endgeräten verbunden.

.

TCP/IP Datenpakete



TCP/IP Adressierung



TCP/IP Adressierung

- Die Identifikation von Geräten in einem Computernetzwerk (wie auch im Internet) erfolgt über eindeutige IP- Adressen.
- Eine IP-Adresse kann einen einzelnen Empfänger oder eine Gruppe beschreiben.
- Zur Zeit werden überwiegend IP-Adressen basierend auf dem Internetprotokoll Version 4 (IPv4) verwendet. Die zukünftige Version 6 (IPv6) ist bereits definiert.

IPv4:

- Jede IP-Adresse besteht aus 32 Bits, d.h. 2^{32} = 4.294.967.296 Adressen.
- Darstellung üblicherweise: 4 Gruppen ganzer dezimaler Zahlen im Intervall 0 – 255.
  - 192.161.1.0,
  - 127.0.0.1
  - 134.252.123.2

IPv6:

- Jede IP-Adresse besteht aus 128 Bits, d.h. 2^{128} = ca. $3{,}4*10^{38}$ Adressen.
- Darstellung üblicherweise: hexadezimal, wobei je 2 Oktetts zusammengefasst werden.
  - 13ac:8513:dc20::4ad2:125:3022

TCP/IP Adressierung

- Eine (IPv4) Adresse besteht aus einem Netzwerk- und einem Host-Teil.
- Die Netzmaske gibt an, welcher Teil der IP-Adresse das Netzwerk bzw. den Host identifiziert.

Beispiel:

Netzwerkadresse: 132.252.121.0

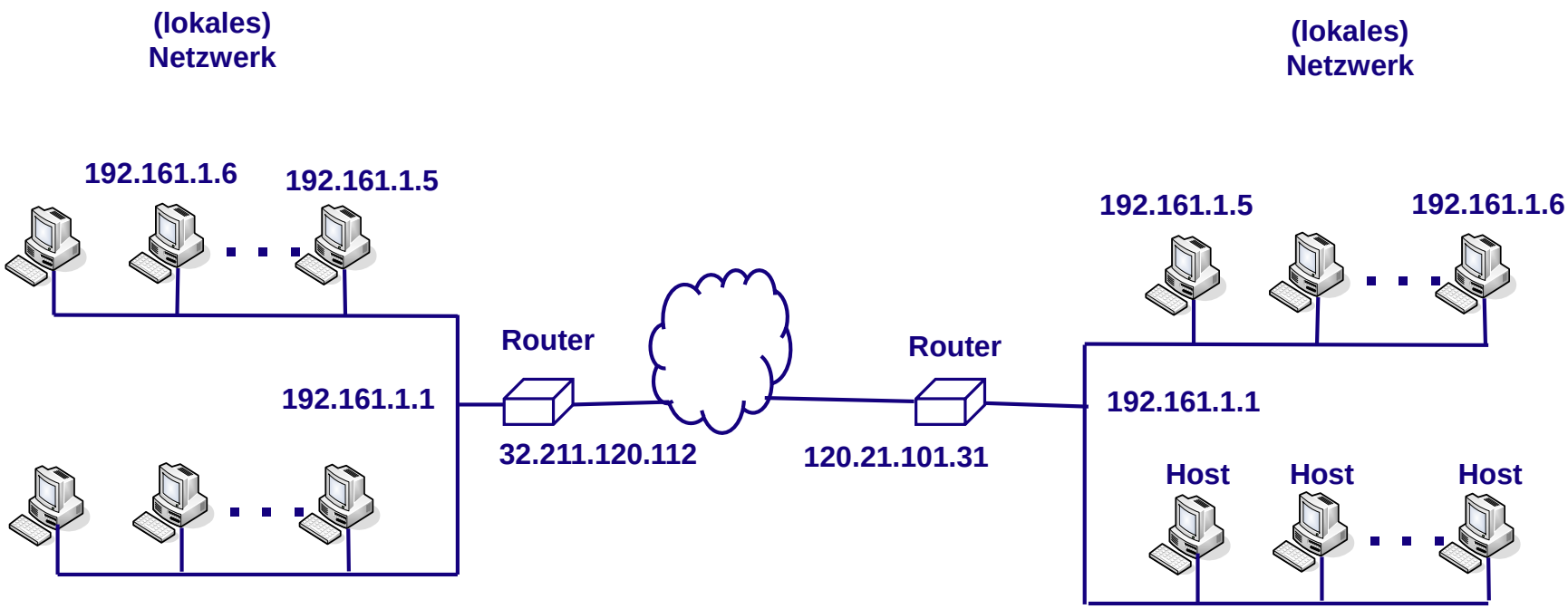
Netzmaske: 255.255.255.0

Alle Hosts, deren Adresse mit 132.252.121 beginnt, gehören zum selben Subnetz.

Es sind max. 254 Adressen (Hosts) in diesem Netz möglich.

- Besondere IP-Adressen:
- Loopback: 127.0.0.1 (beschreibt immer den lokalen Host)
- Broadcast: 255.255.255.255 (Nachrichten werden an alle Hosts gesendet)

TCP/IP Adressierung



- IP-Adressen:können statisch oder dynamisch vergeben werden.
- Eine dynamische Vergabe erfolgt über DHCP (Dynamic Host Configuration Protocol)
- Ein DHCP-Server verwaltet einen Adressbereich.
- Client Rechner erhalten vom DHCP Server eine komplette Netzwerkkonfiguration. (IP-Adresse, Netzmaske, Gateway, DNS Domain….)

DNS

DNS – Domain Name System:

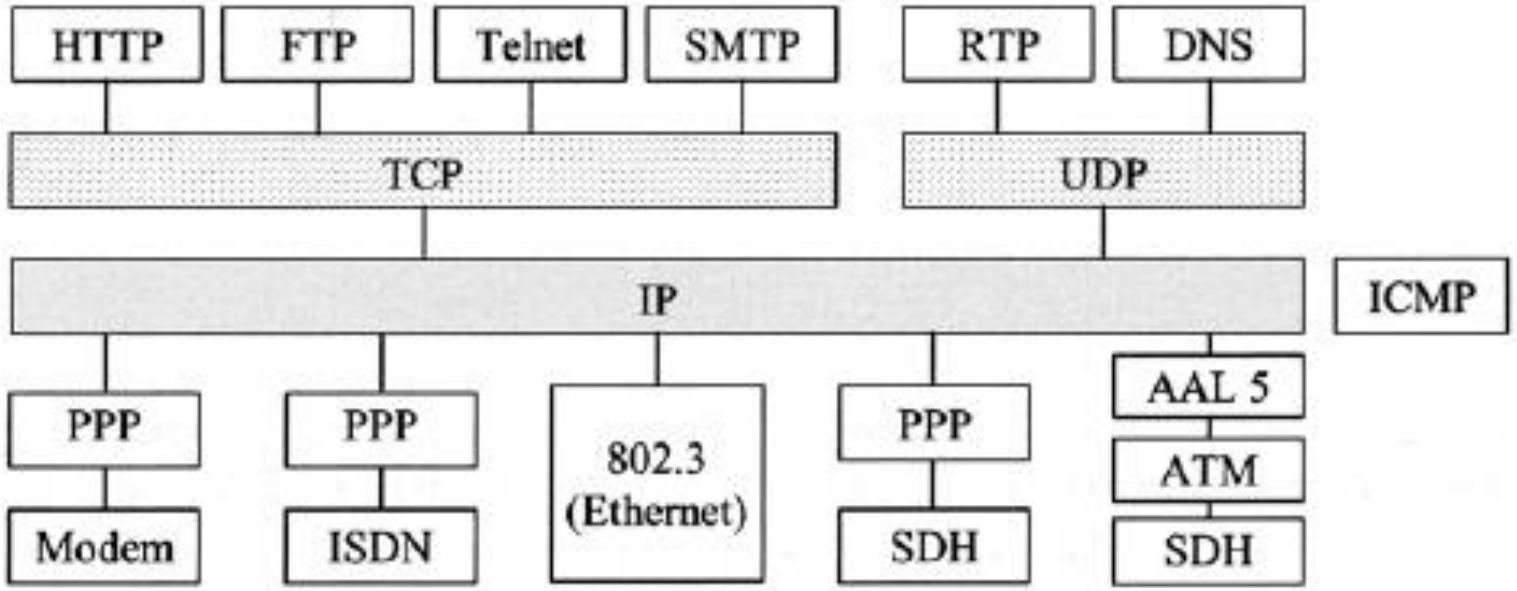
Da IP-Adressen nur schwer lesbar sind, können für Hosts (oder Teilnetze) Namen vergeben werden.

Bsp: Die Adresse 132.252.181.87 hat den Namen www.uni-due.de

- Der Namensraum des Internet wird in Form eines Verzeichnisdienstes durch eine Vielzahl sog. DNS-Server verwaltet.
- Aufteilung in DNS-Zonen, die von einem DNS-Server verwaltet werden.
- Ein DNS-Server verfügt über lookup Tabellen (Auflösen eines Namens zu einer Adresse) und reverse-lookup Tabellen (Finden eines Namens für eine Adresse)

```
@       IN      SOA     sparc1.ikb.uni-essen.de. root.sparc1.ikb.uni-essen.de. (
                                        99060400; Serial
                                        10800   ; Refresh every 3 hours
                                        3600    ; Retry every hour
                                        604800  ; Expire after a week
                                        86400 ) ; Minimum ttl of 1 day
        IN   NS      sparc1.ikb.uni-essen.de.
        IN   NS      ns1.netz.uni-essen.de.
        IN    NS      ns2.netz.uni-essen.de.
localhost                       IN      A       127.0.0.1
; lokale Rechner
sparc1.ikb.uni-essen.de.   IN          A           132.252.121.2
sparc2.ikb.uni-essen.de.   IN          A           132.252.121.3
krikkit.ikb.uni-essen.de.  IN          A           132.252.121.5
```

TCP/IP


HTTP
FTP
Telnet
SMTP
RTP
DNS
TCP
UDP
IP
ICMP
PPP
PPP
802.3
(Ethernet)
PPP
AAL 5
ATM
Modem
ISDN
SDH
SDH

ping [<ip-adr>], [<name>]

- Diagnose Programm zur Überprüfung der Erreichbarkeit eines Rechners im Netzwerk.

```
E:\admin>ping www.uni-due.de
Ping wird ausgeführt für www.uni-duisburg-essen.de [132.252.181.87] mit 32 Bytes Daten:
Antwort von 132.252.181.87: Bytes=32 Zeit=2ms TTL=59
Antwort von 132.252.181.87: Bytes=32 Zeit=2ms TTL=59
Antwort von 132.252.181.87: Bytes=32 Zeit=2ms TTL=59
Antwort von 132.252.181.87: Bytes=32 Zeit=3ms TTL=59
Ping-Statistik für 132.252.181.87:
    Pakete: Gesendet = 4, Empfangen = 4, Verloren = 0
    (0% Verlust),
Ca. Zeitangaben in Millisek.:
    Minimum = 2ms, Maximum = 3ms, Mittelwert = 2ms
E:\admin>
```

ping 127.0.0.1 oder ping loopback

- Ping an den lokalen Rechner. Test des TCP/IP Protokolls

nslookup [<ip-adr>], [<name>]

- Abfrage an den DNS-Dienst nach einem Hostnamen oder einer Adresse.

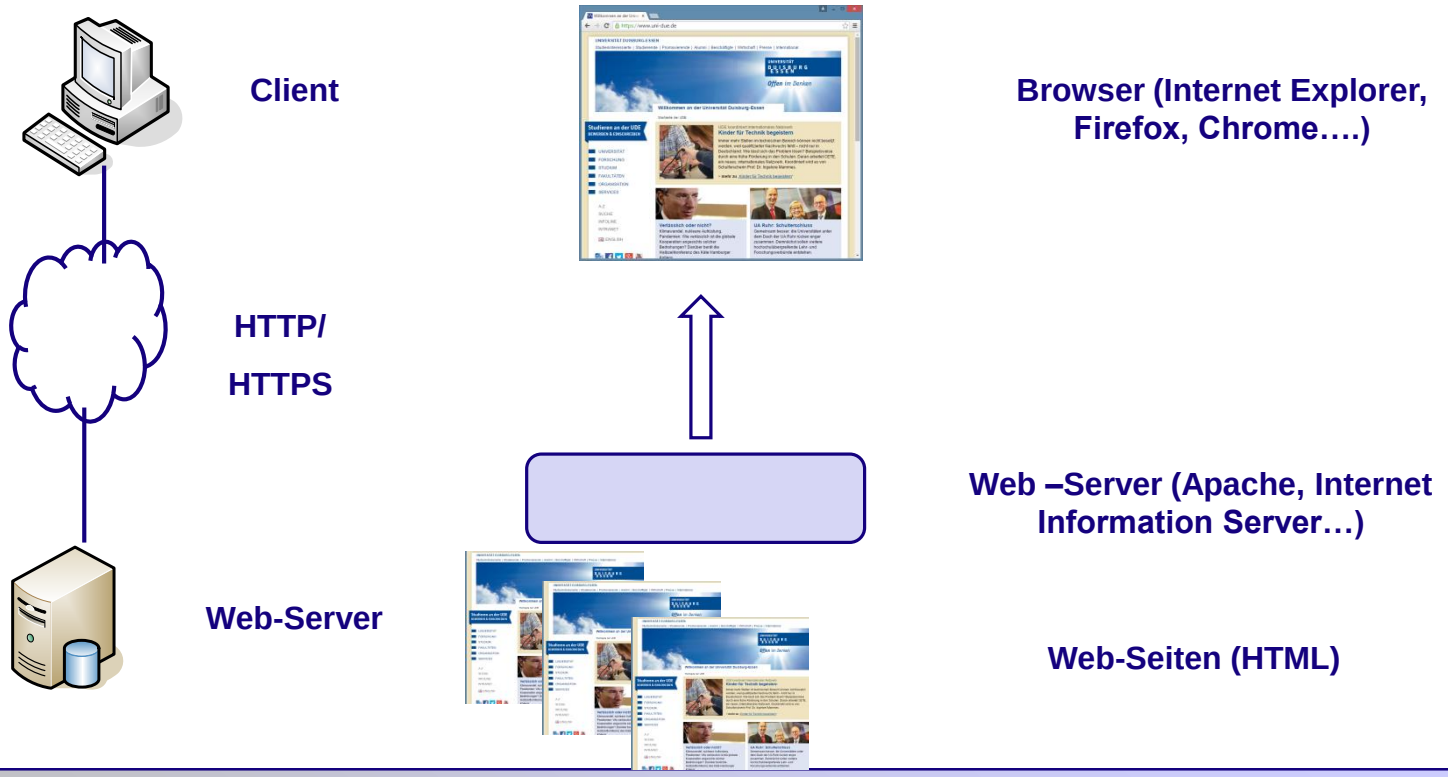
ipconfig [/all]

- .Ausgabe der aktuellen Netzwerk Konfiguration

```
E:\admin>ipconfig

Windows-IP-Konfiguration

Ethernet-Adapter Ethernet:

   Verbindungsspezifisches DNS-Suffix:

   Verbindungslokale IPv6-Adresse  . :
fe80::8103:70e4:cb2d:b3c5%14

   IPv4-Adresse  . . . . . . . . . . : 134.91.174.39

   Subnetzmaske  . . . . . . . . . . : 255.255.255.192

   Standardgateway . . . . . . . . . : 134.91.174.1

Drahtlos-LAN-Adapter LAN-Verbindung* 3:

   Medienstatus. . . . . . . . . . . : Medium getrennt

   Verbindungsspezifisches DNS-Suffix:
```

WWW (World Wide Web)

- System von abrufbaren Dokumenten / Webseiten
- basiert auf Protokollen HTTP, HTTPS
- Bekanntester Dienst im Internet
- Grundprinzip: Aufruf einer Webseite mit einem „Web Browser“ -> Server schickt die angeforderte Seite, die von dem Browser dargestellt wird.
- Webseiten sind durch Hyperlinks verbunden.



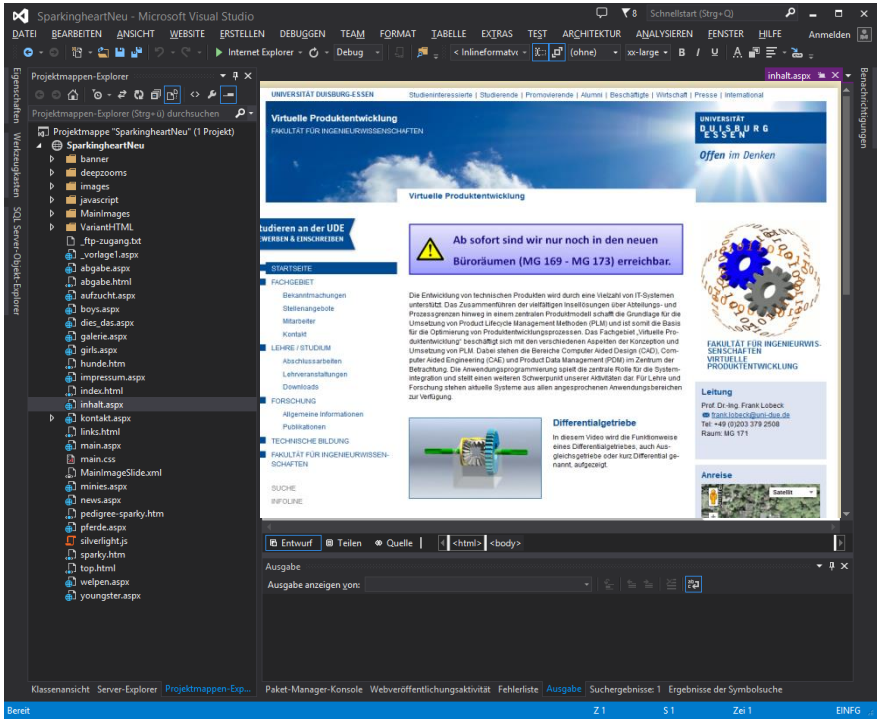
HTML

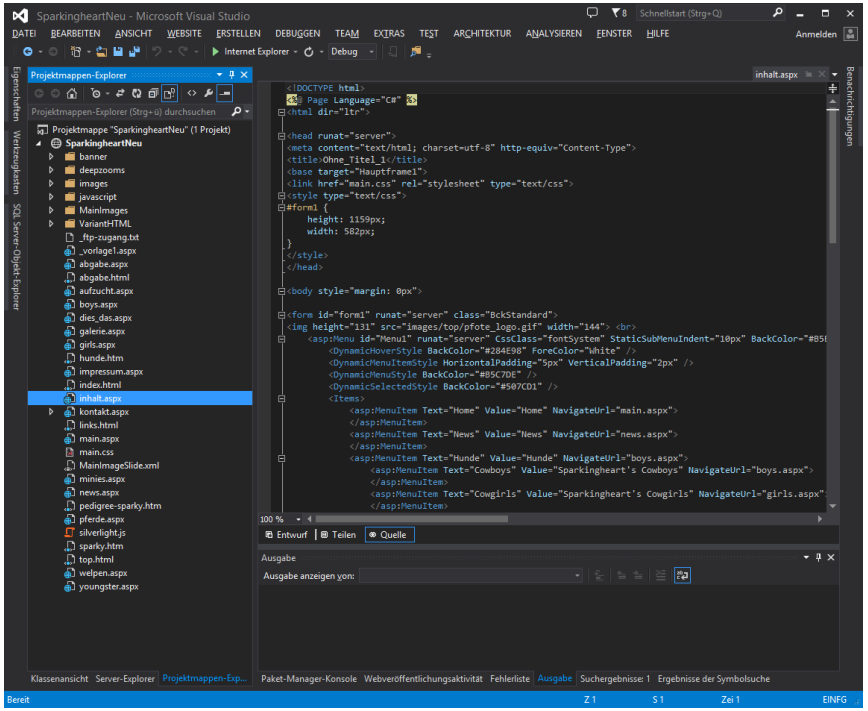
HTML(Hyper Text Markup Language)

- Auszeichnungssprache (Strukturierung, keine Formatierung)
- Strukturierung von Dokumenten
- Texte, Bilder, Links, weitere Medien
- Entwickelt durch das W3C (World Wide Web Consortium)
- Aktuelle Version HTML 5
- HTML wird oft in Verbindung mit CSS (Cascading Style Sheets für die Präsentation der Inhalte) verwendet.


```
<!DOCTYPE html>

<!--[if IEMobile 7 ]><html class="no-js iem7" manifest="default.appcache?v=1"><![endif]-->
<!--[if lt IE 7 ]><html class="no-js ie6" lang="de"><![endif]-->
<!--[if IE 7 ]><html class="no-js ie7" lang="de"><![endif]-->
<!--[if IE 8 ]><html class="no-js ie8" lang="de"><![endif]-->
<!--[if (gte IE 9)|(gt IEMobile 7)|!(IEMobile)|!(IE)]><!--><html class="no-js" lang="de"><!--<![endif]-->

<head>
<meta name="GENERATOR" content="IMPERIA 8.6.33" />

<meta charset="utf-8">
<title>Virtuelle Produktentwicklung</title>
<meta name="keywords" content="fachgebiet, virtuelle, produktentwicklung, universität, duisnurg, essen, lobeck">
<meta name="description" content="Startseite des Fachgebiets für Virtuelle Produktentwicklung an der Universität Duisbur
<meta name="robots" content="index,follow">
<meta name="template" content="_portal2007_basic" />
<!-- mit mobiler Anpassung -->
<meta name="rubrik"    content="Virtuelle Produktentwicklung (/vip) ... Virtuelle Produktentwicklung (/vip)" />
<meta name="X-Imperia-Live-Info" content="6f31e55b-40ae-2401-5560-5694e68b13a1/118827/119305" />

<!-- http://t.co/dKP3o1e -->
<meta name="HandheldFriendly" content="True" />
<meta name="MobileOptimized" content="320" />
<meta name="viewport" content="width=device-width, target-densitydpi=160dpi, initial-scale=1" />

<!-- For less capable mobile browsers
<link rel="stylesheet" media="handheld" href="/portal/css/2011/css/handheld.css?v=1">  -->

<!-- For all browsers -->
<link rel="stylesheet" href="/portal/css/2011/style_2013.css" />
<link rel="stylesheet" href="/portal/css/2011/legacy.css" />
<link rel="stylesheet" href="/portal/css/2011/jqueryui/uni-due.css" />
<!-- JavaScript at bottom except for Modernizr and Cookies-->
<!-- general functions for reading/writing cookies and mobile menu -->
<script src="/portal/tools/js/2011/cookies.js"></script>
<script src="/portal/tools/js/2011/mobilemenu.js"></script>
<!-- / -->
<script src="/portal/tools/js/2011/libs/modernizr-1.7.min.js"></script>

<!-- For iPhone 4 -->
<link rel="apple-touch-icon-precomposed" sizes="114x114" href="/imperia/md/images/cms/h/apple-touch-icon.png">
<!-- For iPad 1-->
<link rel="apple-touch-icon-precomposed" sizes="72x72" href="/imperia/md/images/cms/m/apple-touch-icon.png">
<!-- For iPhone 3G, iPod Touch and Android -->
<link rel="apple-touch-icon-precomposed" href="/imperia/md/images/cms/l/apple-touch-icon-precomposed.png">
<!-- For Nokia -->
<link rel="shortcut icon" href="/imperia/md/images/cms/l/apple-touch-icon.png">
<!-- For everything else -->
<link rel="shortcut icon" href="/favicon.ico">
```

HTML (Hyper Text Markup Language)





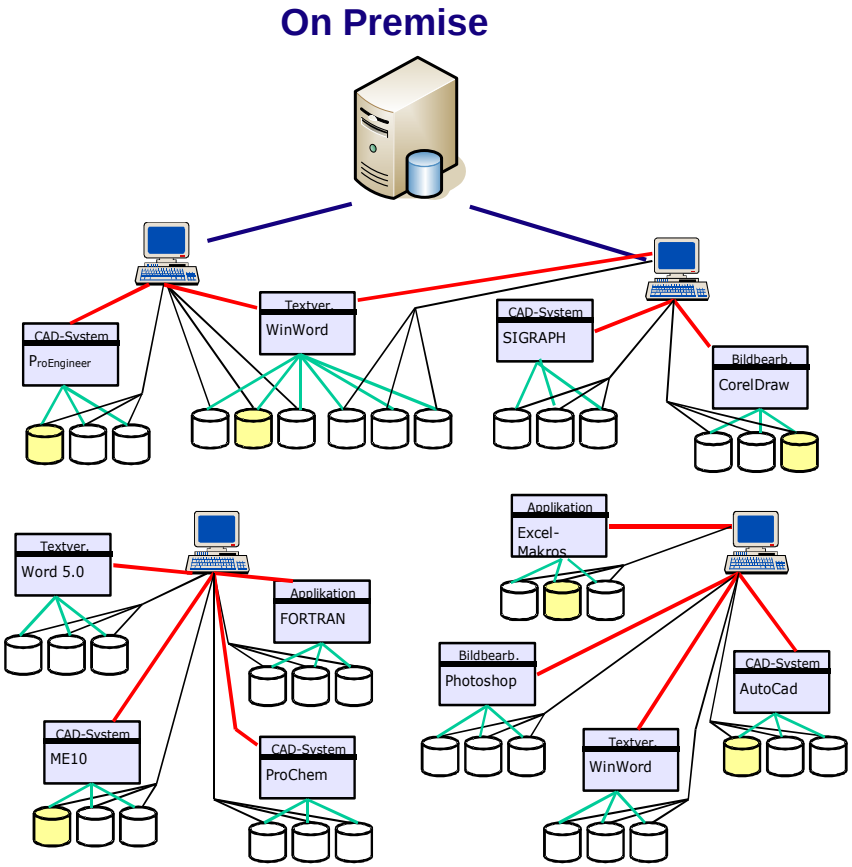
Web 2.0

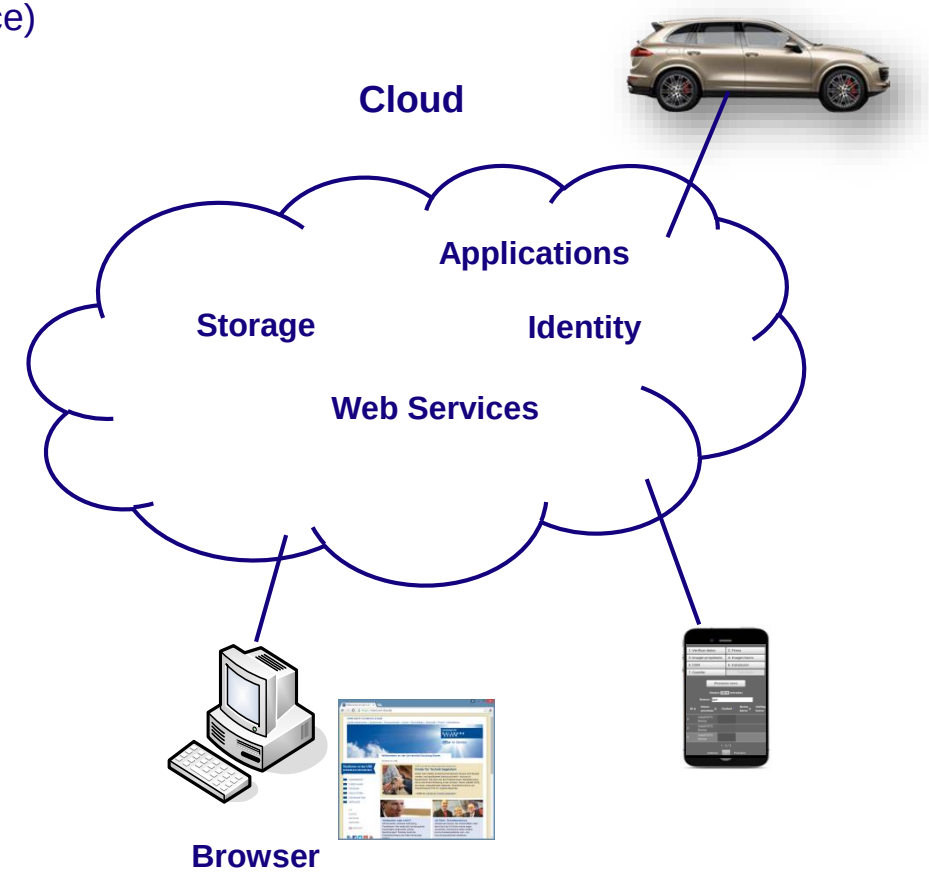
- Web 2.0 bezeichnet eine Entwicklung, die dem Anwender eine aktive Gestaltung / Interaktivität erlaubt.
- Sammelbegriff für verschiedene Techniken / Anwendungen
  - Foren
  - Blogs
  - Wiki-Technik
  - …

Quelle: Wikipedia, https://de.wikipedia.org/wiki/Web_2.0

Cloud (Cloud Computing)

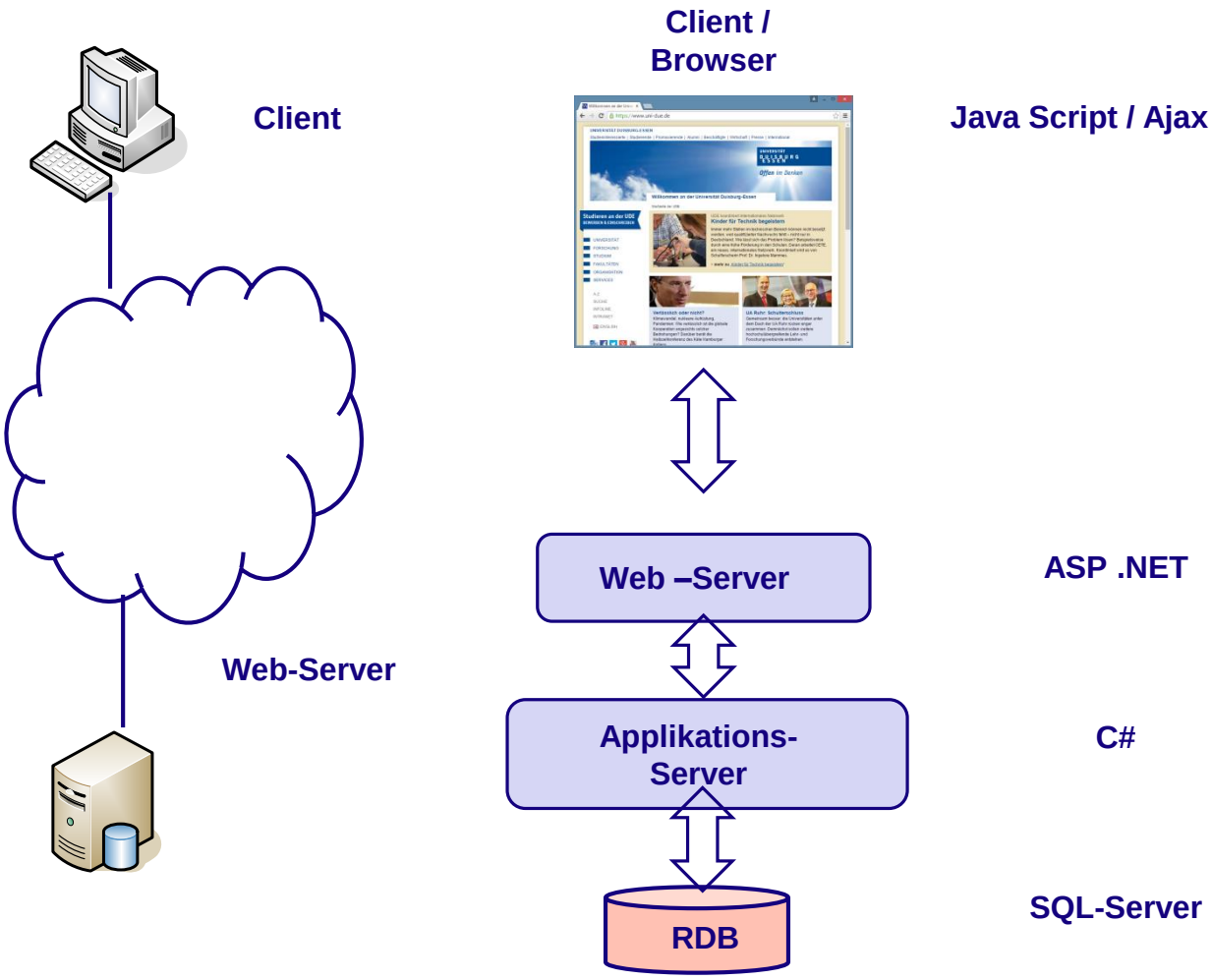
- Ursprünglich bezeichnet Cloud Computing das Speichern von Daten und die Ausführung von Programmen auf einem (oder mehreren) Servern.
- Alle IT-Anwendungen stehen in der Cloud skalierbar zur Verfügung.
- Geschäftsmodell: SaaS (Software as a Service)



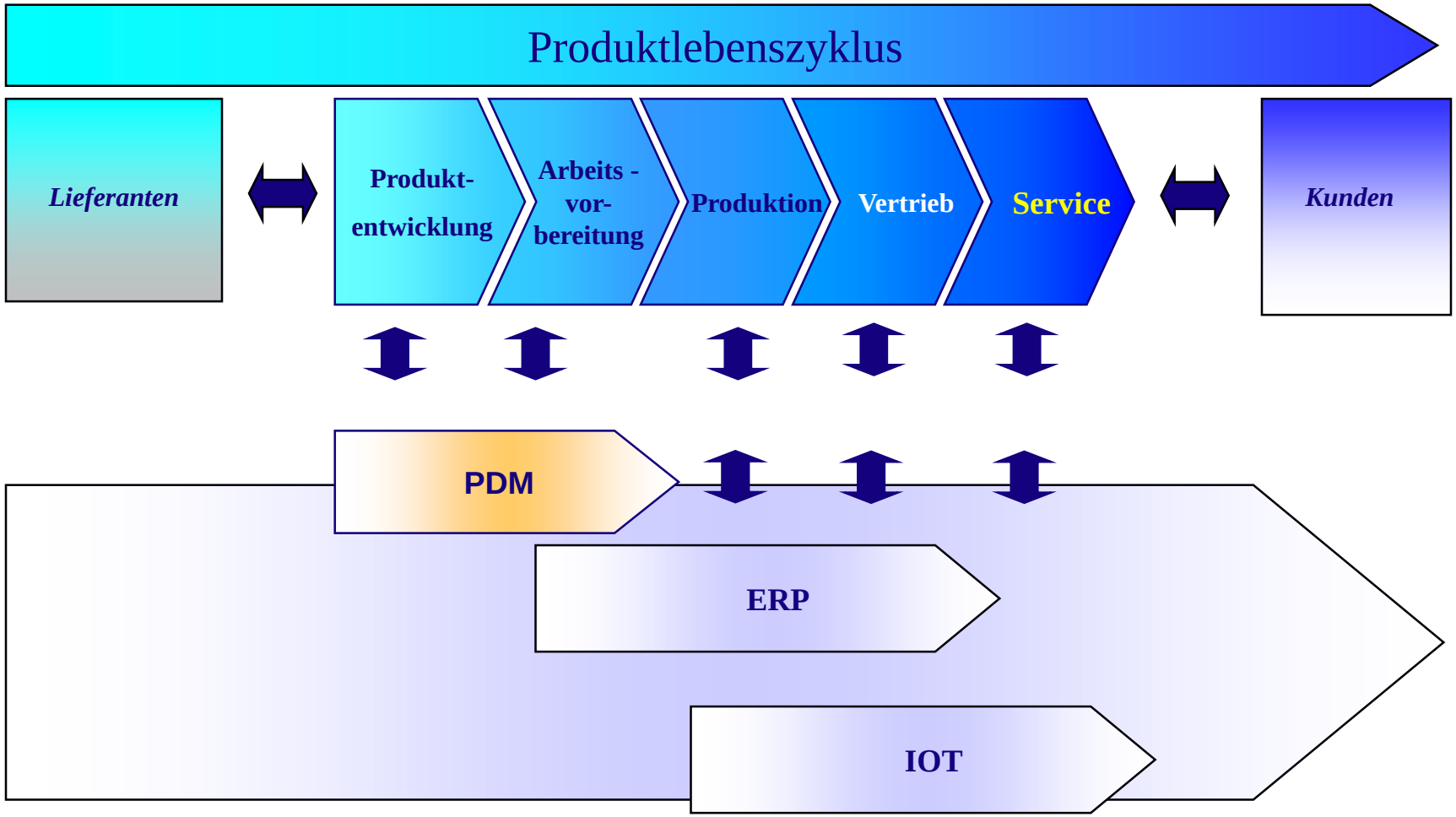


Cloud – Applikationen (Beispiel)

Beispiel: Web Anwendung



Bezug zu PDM / PLM



Web-Anwendungen

Grundlagen

Grundlagen

C# (c sharp): Von Microsoft entwickelte Programmiersprache

Bestandteil der .NET Strategie (des .NET Frameworks)

Objektorientierte Programmiersprache

Syntax ähnlich C++

- Bereinigt um „unsichere“ Elemente, wie Pointer
- Erweiterter Sprachumfang (vergleichbar Java)

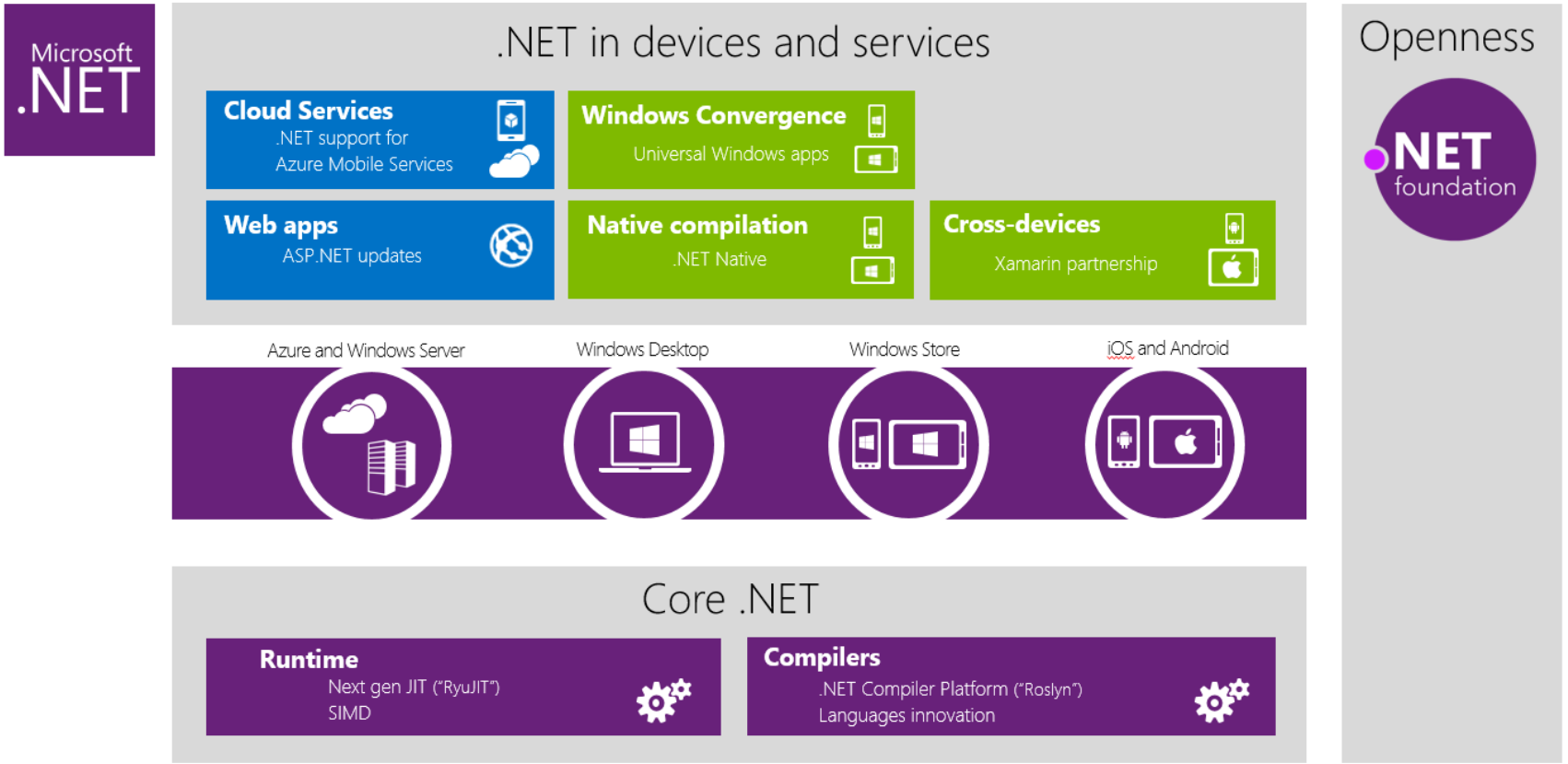
Literatur: http://www.guidetocsharp.de/

Skript ZIM: Die Programmiersprache C#

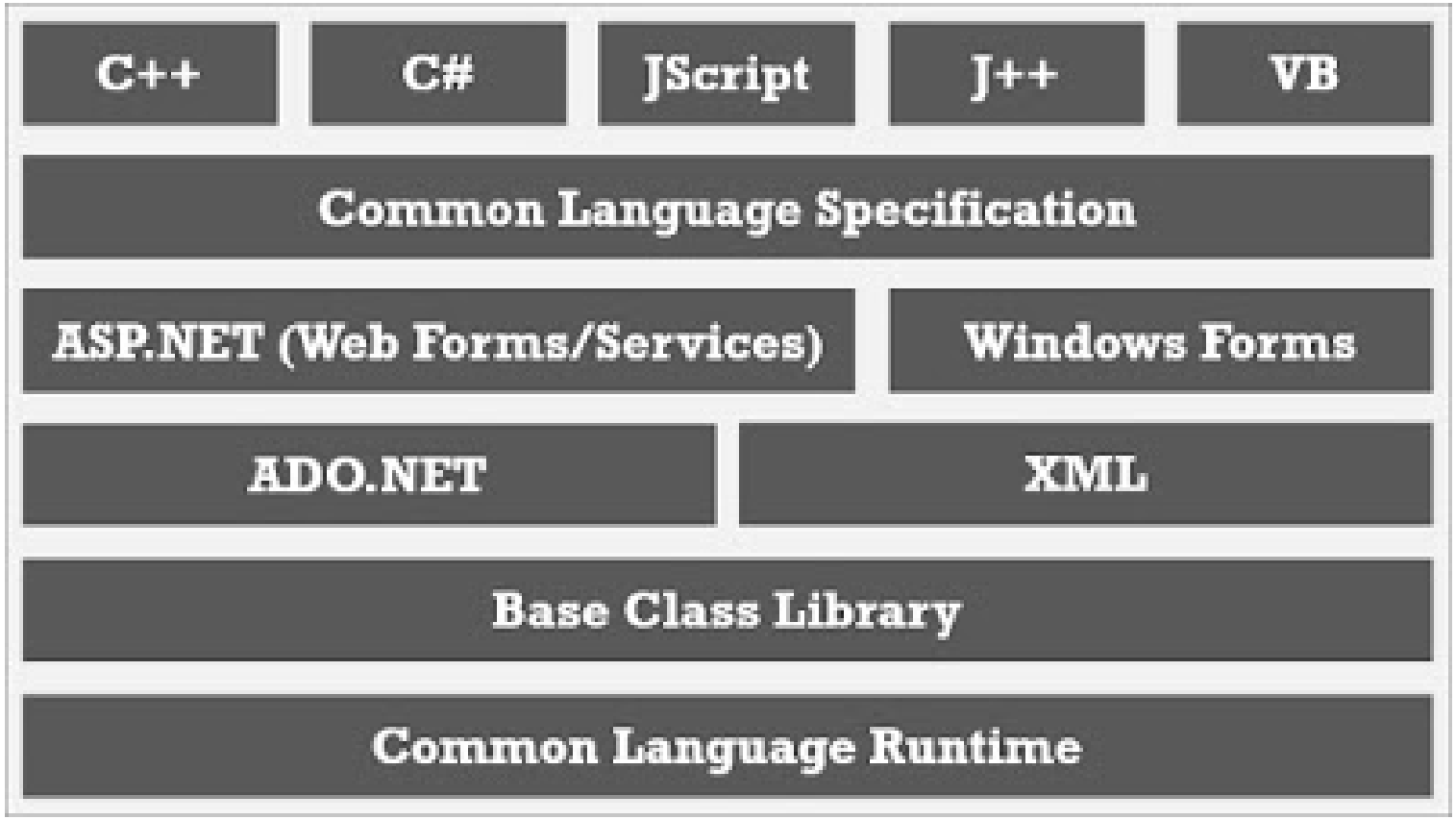
Foren:

www.stackoverflow.com

www.codeproject.com


Microsoft
.NET
.NET in devices and services
Cloud Services
.NET support for
Azure Mobile Services
Windows Convergence
Universal Windows apps
Web apps
ASP.NET updates
Native compilation
.NET Native
Cross-devices
Xamarin partnership
Azure and Windows Server
Windows Desktop
Windows Store
iOS and Android
Core .NET
Runtime
Next gen JIT ("RyuJIT")
SIMD
Compilers
.NET Compiler Platform ("Roslyn")
Languages innovation
Openness
.NET
foundation

.NET Architektur / Common Language Runtime



Namensräume

Die Namen von Klassen (und sonstigen Datentypen) müssen eindeutig sein. Zur Vereinfachung gibt es strukturierte Namensräume. Der vollständige Aufruf einer Methode sieht beispielsweise so aus:

```
System.Console.Write (“Hallo Welt.“);
```

<Namensraum>.<Klasse>.<Methode>(<Parameter>)

Zur Vereinfachung kann mit Hilfe der using-Direktive ein Namensraum (Namespace) importiert werden.

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

Namensräume

Namensraum	Inhalt
System	... enthält grundlegende Basisklassen sowie Klassen für Dienstleistungen wie Konsolenkommunikation oder mathematische Berechnungen. U.a. befindet sich hier die Klasse **Console**, die wir im Einführungsbeispiel für den Zugriff auf Bildschirm und Tastatur verwendet haben.
System.Collections	... enthält Container zum Verwalten von Listen, Warteschlangen, Bitarrays, Hashtabellen etc.
System.Data	... enthält zusammen mit diversen untergeordneten Namensräumen (z.B. **System.Data.SqlClient**) die Klassen zur Datenbankbearbeitung.
System.IO	... enthält Klassen für die Ein-/Ausgabebehandlung im Datenstrom-Paradigma.
System.Net	... enthält Klassen für die Netzwerk-Programmierung.
System.Reflection	... ermöglichst es u.a., zur Laufzeit Informationen über Klassen abzufragen oder neue Methoden zu erzeugen. Dabei werden die Metadaten in den .NET – Assemblies genutzt.
System.Seccurity	... enthält Klassen, die sich z.B. mit Verschlüsselungs-Techniken beschäftigen.
System.Threading	... unterstützt parallele Ausführungsfäden.
System.Web	... unterstützt die Entwicklung von Internet-Anwendungen (inkl. ASP.NET).
System.Windows.Controls	... enthält Klassen für die Steuerelemente einer Windows-Anwendung (z.B. Befehlschalter, Textfelder, Menüs).
System.XML	... enthält Klassen für den Umgang mit XML-Dokumenten.

Beispiel einer Klasse: Bruch

Klassenname, Klassendefinition innerhalb des Blocks aus { }

Attribute der Klasse (Membervariablen), Typ: int

```
public class Bruch
{
int _zaehler, _nenner;
public int Zaehler { get { return _zaehler; } set { _zaehler = value; } }
public int Nenner { get { return _nenner; } set { _nenner = value; } }
```

Öffentliche Eigenschaften.

```
Bruch ()
{
_zaehler = 0;
_nenner = 1;
}
```

Konstruktor

(Öffentliche) Methode

```
public double PrintDezimal()
{
return _zaehler / _nenner;
}
```

Anweisungen: Werden durch ein „;“ abgeschlossen.

Variablen / Datentypen

Variablen dienen zum Speichern von Daten.

Variablen haben die folgenden Eigenschaften:

Name: Ein beliebiger Bezeichner, um auf die Variable zuzugreifen.

Datentyp: Wertebereich und Art der Werte, die von der Variablen gespeichert werden.

Wert:

Variablen müssen vor der ersten Verwendung deklariert werden. Deklarationen sehen i.a. so aus:

<Datentyp> <Name> [= <Wert>];

Beispiele:

```
Int Anzahl;
double MwSt = 0.19;
Bruch B (1,0);
string Text;
```

Datentyp kann jede Klasse oder auch einer der elementare Datentypen sein.

Datentypen

Typ	Beschreibung	Werte	Bits
sbyte	Diese Variablentypen speichern ganze Zahlen *mit* Vorzeichen. Beispiel: `int zaehler = -7`	–128 … 127	8
short		–32768 … 32767	16
int		–2147483648 ... 2147483647	32
long		–9223372036854775808 … 9223372036854775807	64
byte	Diese Variablentypen speichern ganze Zahlen ≥ 0. Beispiel: `byte alter = 31;`	0 … 255	8
ushort		0 … 65535	16
uint		0 ... 4294967295	32
ulong		0 … 18446744073709551615	64
float	Variablen vom Typ **float** speichern Gleitkommazahlen nach der Norm IEEE 754 (32 Bit) mit einer Genauigkeit von mind. 7 signifikanten Dezimalstellen. Beispiel: `float p = 1252.61f;` **float**-Literale (siehe unten) benötigen den Suffix **f** (oder **F**).	Minimum: $-3.4028235 \cdot 10^{38}$ Maximum: $3.4028235 \cdot 10^{38}$ Kleinster Betrag > 0: $1.4 \cdot 10^{-45}$	32 1 für das Vorz., 8 für den Expon., 23 für die Mantisse

Datentypen

Typ	Beschreibung	Werte	Bits
char	Variablen vom Typ **char** speichern ein Unicode-Zeichen. Im Speicher landet aber nicht die Gestalt eines Zeichens, sondern seine Nummer im Zeichensatz. Daher zählt **char** zu den ganzzahligen (integralen) Datentypen. Beispiel: `char zeichen = 'j';` **char** – Literale (s.u.) sind mit *einfachen* Anführungszeichen einzurahmen.	Unicode-Zeichen Tabellen mit allen Unicode-Zeichen sind z.B. auf der folgenden Webseite http://www.unicode.org/charts/ des Unicode-Konsortiums zu finden.	16
bool	Variablen vom Typ **bool** speichern Wahrheitswerte. Beispiel: `bool cond = false;`	**true**, **false**	1
double	Variablen vom Typ **double** speichern Gleitkommazahlen nach der Norm IEEE 754 (64 Bit) mit einer Genauigkeit von 15-16 signifikanten Dezimalstellen. Beispiel: `double p=113445.626535891;`	Minimum: $-1{,}7976931348623157 \cdot 10^{308}$ Maximum: $1{,}7976931348623157 \cdot 10^{308}$ Kleinster Betrag > 0: $4{,}9 \cdot 10^{-324}$	64 1 für das Vorz., 11 für den Expon., 52 für die Mantisse
decimal	Variablen vom Typ **decimal** speichern Gleitkommazahlen mit 28 bis 29 Dezimalstellen exakt und eignen sich besonders für die **Finanzmathematik**, wo Rundungsfehler zu vermeiden sind. Beispiel: `decimal p = 2344.2554634m;` **decimal**-Literale (siehe unten) benötigen den Suffix **m** (oder **M**).	Minimum: $-(2^{96}-1) \approx -7{,}9 \cdot 10^{28}$ Maximum: $2^{96}-1 \approx 7{,}9 \cdot 10^{28}$ Kleinster Betrag > 0: 10^{-28}	128 1 für das Vorz., 5 für den Expon., 96 für die Mantisse, restl. Bits ungenutzt Im Exponenten sind nur die Werte 0 bis 28 erlaubt, die negativ interpret. werden.

Architektur

Client / Browser

Java Script / Ajax

Web –Server

ASP .NET

Applikations-Server

C#

RDB

SQL-Server

ASP.NET

ASP := Active Server Page

- Dynamische Webseiten

.NET := SW-Architektur/Technologie von Microsoft

- Sprachen: VisualBasic, C#, Java, C/C++
- Quellcode wird übersetzt, Binärcode wird innerhalb einer Laufzeitumgebung (Ähnlich Java VM) ausgeführt.
- „Managed Code“
- CLR: Common Language Runtime

Grundlagen

C# (c sharp): Von Microsoft entwickelte Programmiersprache

Bestandteil der .NET Strategie (des .NET Frameworks)

Objektorientierte Programmiersprache

Syntax ähnlich C++

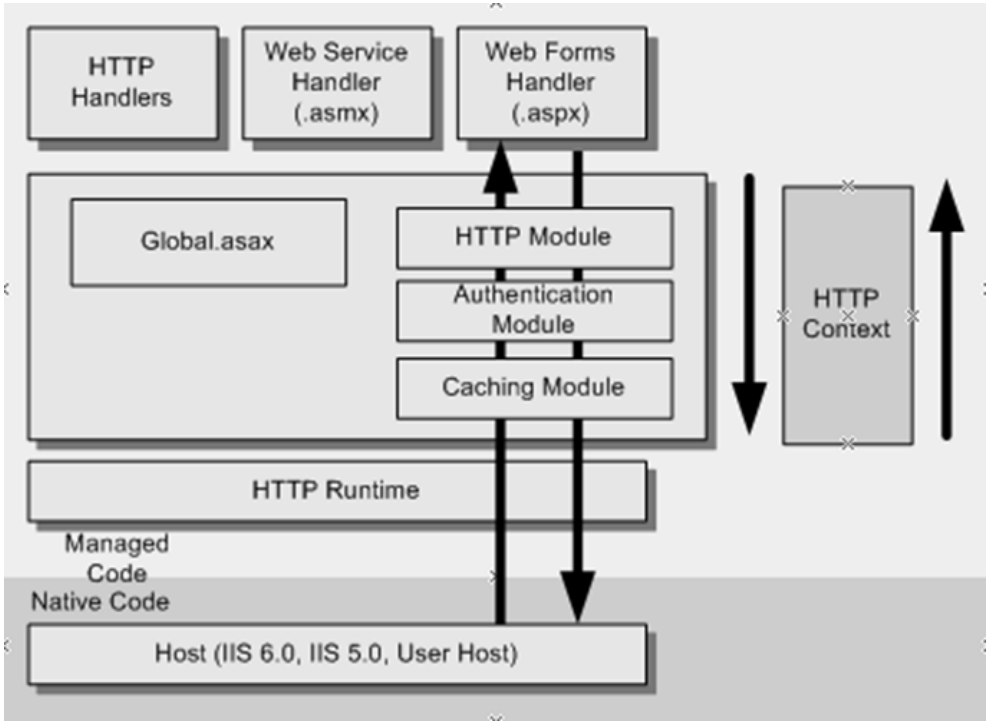
- Bereinigt um „unsichere“ Elemente, wie Pointer
- Erweiterter Sprachumfang (vergleichbar Java)

Literatur: http://www.guidetocsharp.de/

Skript ZIM: Die Programmiersprache C#

ASP.NET (Active Server Pages .NET): Von Microsoft entwickeltes Framework

- Erstellung von
  - dynamischen Webseiten
  - Webanwendungen
  - Web Services
- Nach PHP die zweithöchste Verbreitung innerhalb der serverseitigen Technologien (ca. 18%)
- Unterstützte Programmiersprachen: Alle .NET fähigen; C#, VB…
- IDE: (Integrated Development Environment) / Entwicklungs-umgebung: Visual Studio



ASP.NET

ASP.NET unterstützt mehrere Verfahren zur Anwendungserstellung:

ASP.NET WebForms, ASP.NET AJAX, ASP.NET MVC, ASP.NET Dynamic Data, ASP.NET WebAPI und ASP.NET SignalR

Diese sind jeweils als HTTP-Handler realisiert. In einer Anwendung können mehrere Verfahren gleichzeitig verwendet werden.

WebForms Anwendungen:

Dateiendung: .aspx

- Statisches XHTML Markup (analog HTML-Datei)
- Steuerelemente (Controls), wie Textbox, ListBox, Kalender….
- Startdatei einer Webseite: default.aspx

Code, der auf dem Server ausgeführt werden soll, kann in .aspx-Datei integriert werden:

```
<%-- Programm-Code --%>
```

ASP.NET – Code Behind

Die empfohlene Vorgehensweise bei der Erstellung von ASP.NET Anwendungen entspricht dem

von Microsoft entwickelten Ansatz „Code Behind“.

Dabei werden die (statischen, clientseitigen) Elemente in einer **.aspx-Datei**,

die dynamischen (auf dem Server ausgeführten) Inhalte in einer zweiten, sogenannten „Code-Behind Datei“ eingegeben. Der Dateiname der Code-Behind Datei ist (per Definition) identisch mit dem der .aspx-Datei, wobei eine Erweiterung für die verwendete Programmiersprache angehängt wird. Z.B.

default.aspx - default.aspx.cs für C#

default.aspx – default.aspx.ba für Visual Basic

Die Code-Behind Datei wird compiliert.

Der Ansatz unterstützt die Trennung von Code für die Darstellung und für die Inhalte. Die Geschäftslogik wird üblicherweise auf dem Server ausgeführt und in der Code-Behind Datei implementiert.

ASP.NET – Code Behind

default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

default.aspx.cs

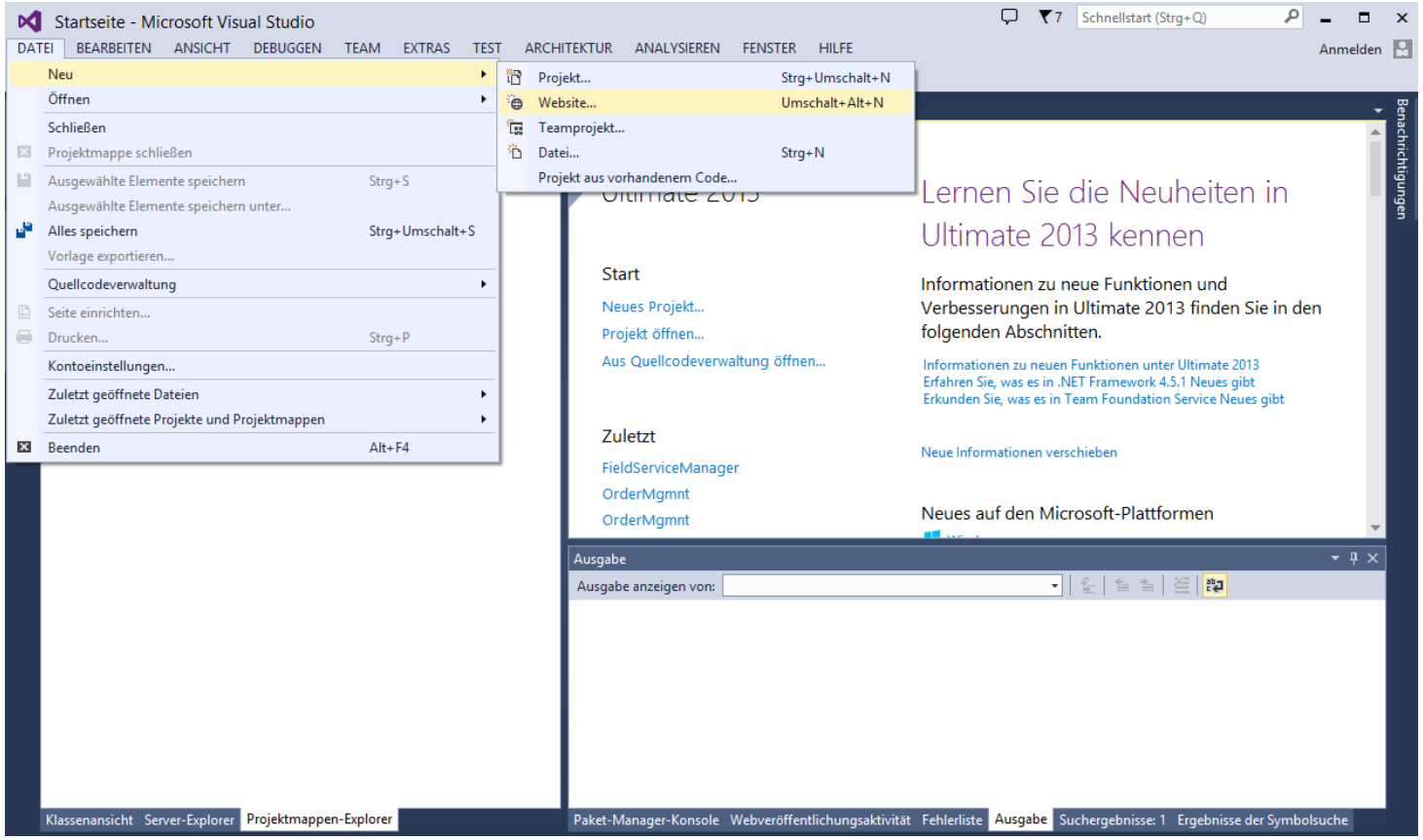
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
}
```

Erstellung einer Web-Anwendung

Visual Studio starten

- Datei -> Neu -> Webseite
- „Leere ASP.NET Webseite“ auswählen.
- Verzeichnis wählen (oder erstellen)

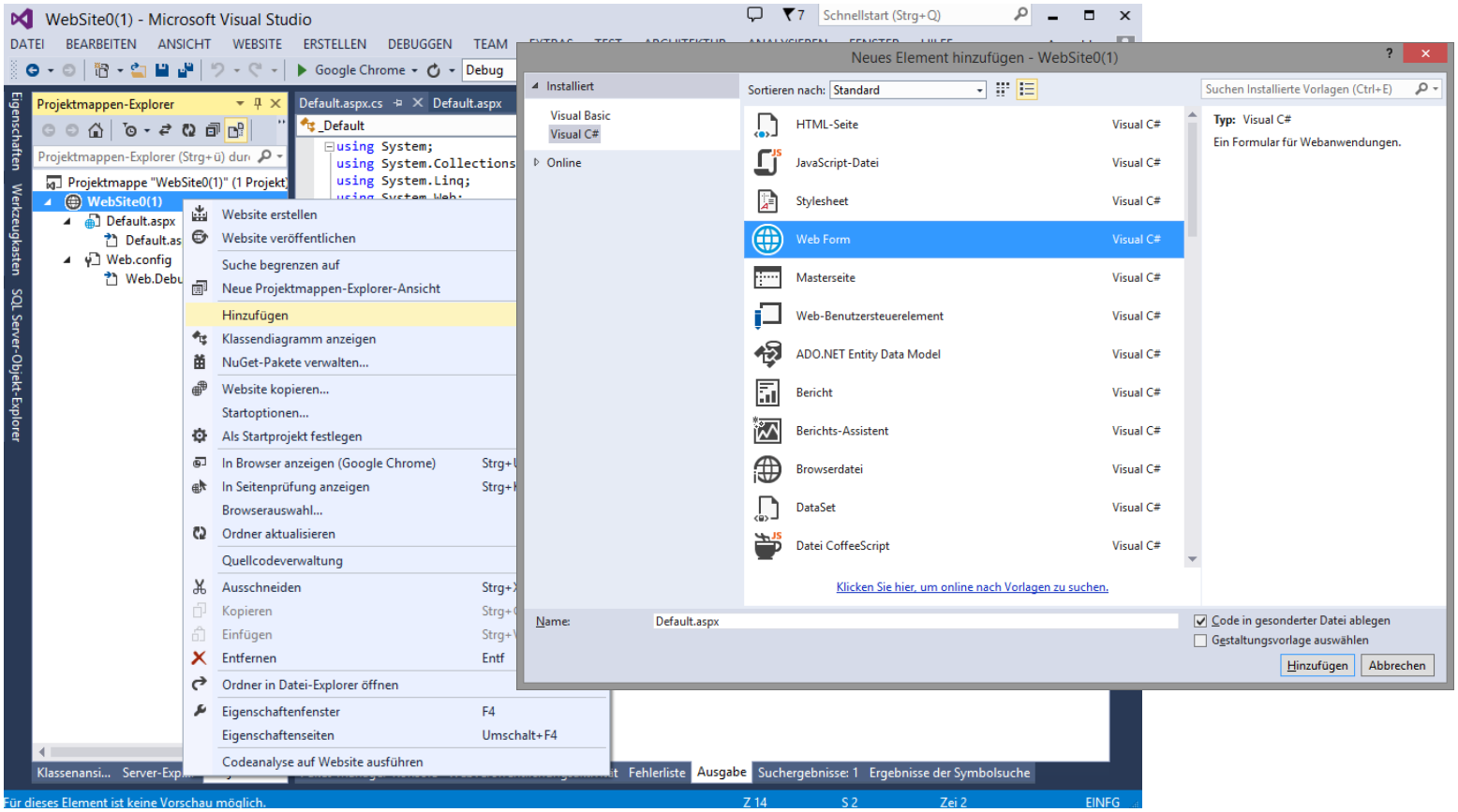
Visual Studio erstellt eine Projektmappe, Verzeichnisse und Dateien für das Web-Projekt.



Erstellung einer Web-Anwendung

- Dem Projekt eine neue Datei hinzufügen.
- Name: Default.aspx

Die IDE erstellt die .aspx- und die zugehörige .aspx.cs-Datei

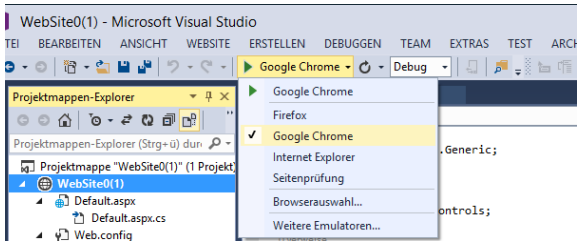


Erstellung einer Web-Anwendung

Zum Testen kann die Anwendung im Debug-Modus gestartet werden.

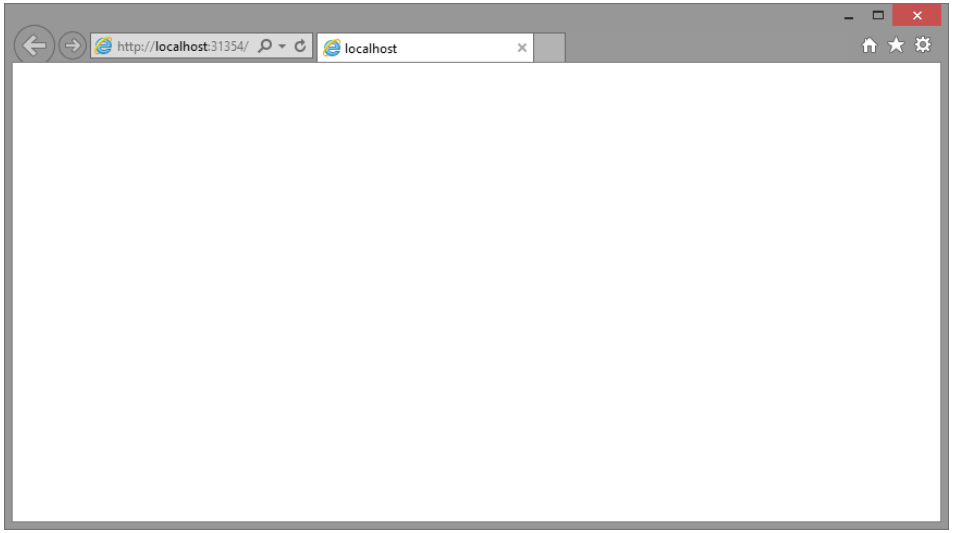
Visual Studio enthält dazu einen integrierten, lokalen Webserver (IIS). Die Anwendung kann so in verschiedenen Browsern getestet werden ohne, dass eine Veröffentlichung (Deployment) erforderlich ist.

- Anwendung kompilieren: Erstellen -> Projektmappe erstellen (Strg-Shift-B)
- Anwendung debuggen: Debuggen -> Debugging starten (F5) oder:



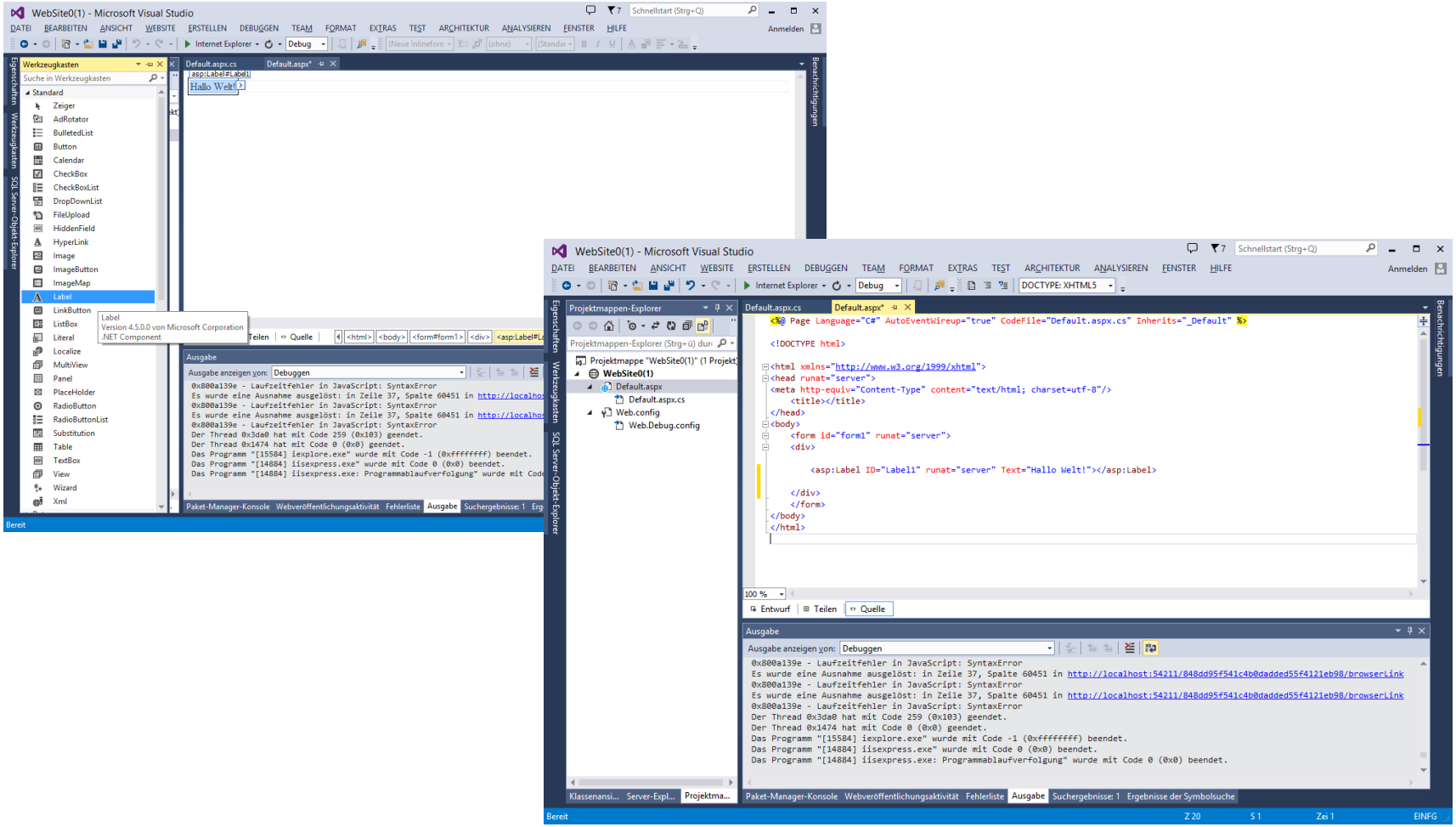
Die Anwendung wird gestartet und in einem neuen Tab des ausgewählten Browsers ausgeführt.

Breakpoints können in Code-Behind Dateien gesetzt werden.



Hinzufügen von Steuerelementen

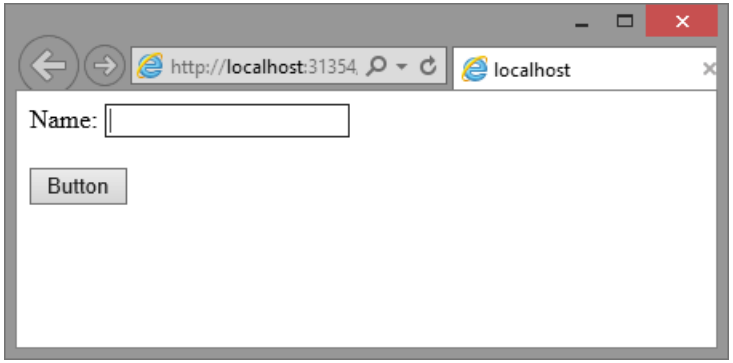
In einer .aspx-Datei können beliebige HTML Controls eingegeben werden. Darüber hinaus enthält die IDE Bibliotheken mit Steuerelementen, die auch mit Hilfe des Werkzeugkastens in die Seite eingefügt werden können.

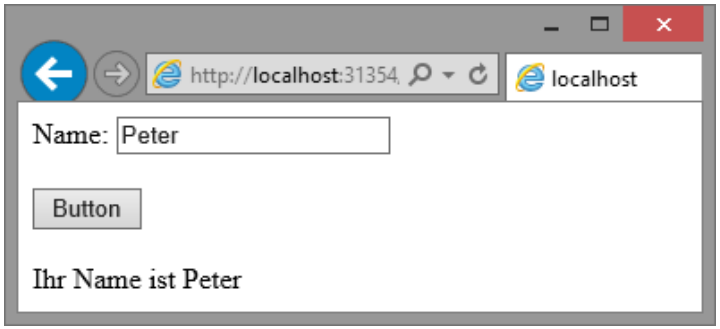


Hinzufügen von Steuerelementen

ASP.NET unterstützt die ereignisgesteuerte Programmierung.

Steuerelemente lösen Ereignisse aus, die serverseitig bearbeitet werden können.





Hinzufügen von Steuerelementen

ASP.NET unterstützt die ereignisgesteuerte Programmierung.

Steuerelemente lösen Ereignisse aus, die serverseitig bearbeitet werden können.

Default.aspx.cs | Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server" Text="Name: "></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label2" runat="server" Text="Label" Visible="false"></asp:Label>
        <br />

    </div>
    </form>
</body>
</html>
```

100 % | Entwurf | Teilen | Quelle | <html> <body> <form#form1> <asp:Label#Label2>

Default.aspx.cs | Default.aspx

_Default

```
0 Verweise
public partial class _Default : System.Web.UI.Page
{
    0 Verweise
    protected void Page_Load(object sender, EventArgs e)
    {

    } 2
    0 Verweise
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label2.Text = "Ihr Name ist " + TextBox1.Text;
        Label2.Visible = true;
    }
}
```

Hinzufügen von Steuerelementen

ASP.NET unterstützt die ereignisgesteuerte Programmierung.

Steuerelemente lösen Ereignisse aus, die serverseitig bearbeitet werden können.

Default.aspx.cs | Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server" Text="Name: "></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label2" runat="server" Text="Label" Visible="false"></asp:Label>
        <br />

    </div>
    </form>
</body>
</html>
```

100 % | Entwurf | Teilen | Quelle | <html> <body> <form#form1> <asp:Label#Label2>

Default.aspx.cs | Default.aspx

_Default

```
0 Verweise
public partial class _Default : System.Web.UI.Page
{
    0 Verweise
    protected void Page_Load(object sender, EventArgs e)
    {

    } 2
    0 Verweise
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label2.Text = "Ihr Name ist " + TextBox1.Text;
        Label2.Visible = true;
    }
}
```

Erstellung einer Web-Anwendung

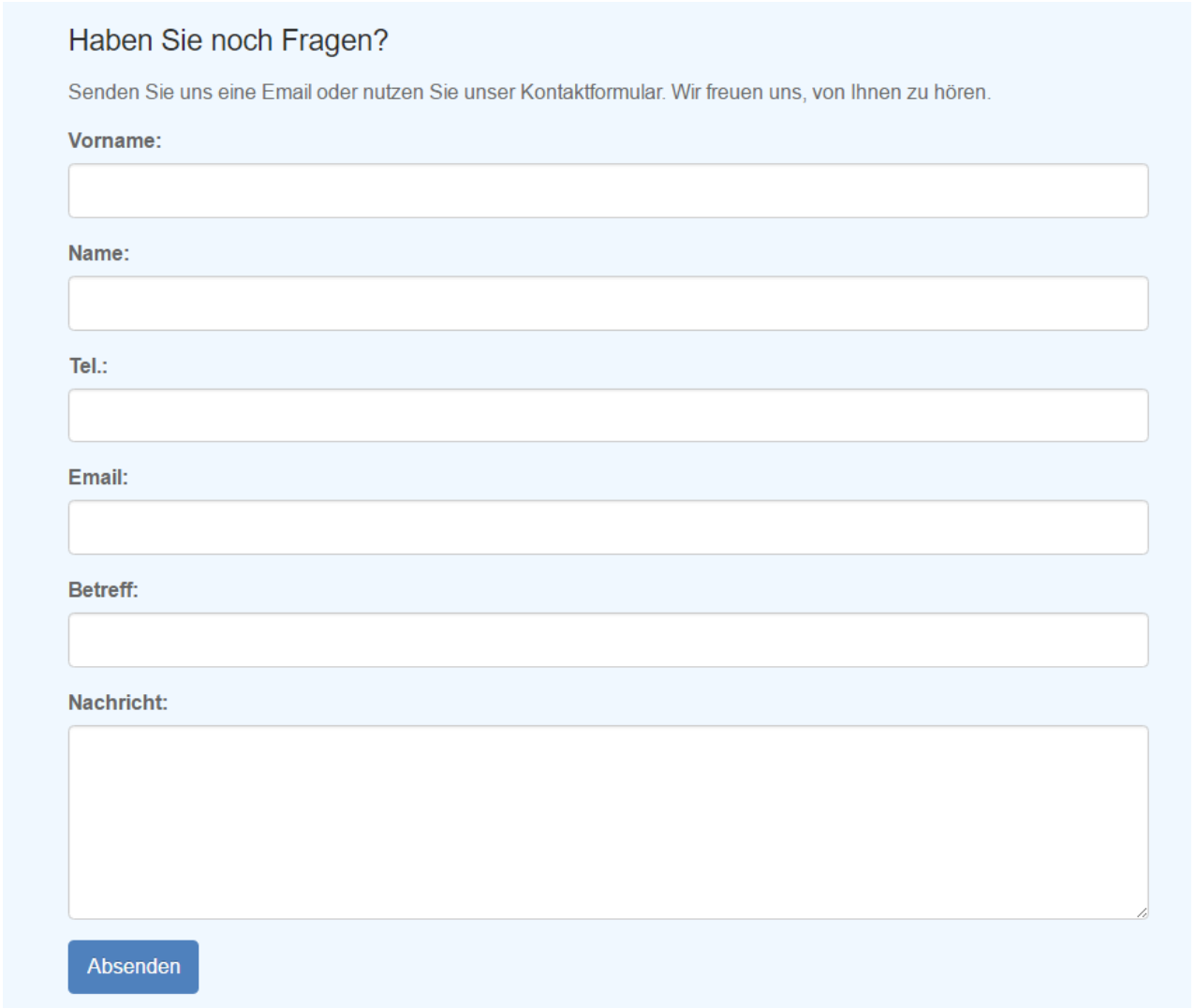
Aufgabe

Erstellen sie eine ASP.NET Anwendung, in der Personen-Daten eingegeben werden können. Die Anwendung soll Steuerelemente für folgende Daten enthalten.

Name, Vorname, Tel., Email-Adresse, Betreff, Nachricht

Es soll weiterhin eine Schaltfläche (Button) vorhanden sein, die eine Validierung der eingegebenen Daten ausführt.

Aufgabe



Erweitern Sie die Webseite aus aufgabe 1 um die Steuerelemente für eine Kontakt Seite.

Bei Betätigen des Absenden-Buttons soll eine Email verschickt werden.

Aufgabe

```
}
protected void cmdSubmit_Click1(object sender, EventArgs e)
{
    bool bValid = true;
    bool bValidEmail = false;
    // Validation
    if (TextName.Text.Length == 0)
    {
        bValid = false;
        LabelMsg.Text += "**Name** darf nicht leer sein. ";
    }
    if (TextEmail.Text.Length == 0)
    {
        bValid = false;
        LabelMsg.Text += "**Email** darf nicht leer sein. ";
    }
    else if (TextEmail.Text.IndexOf("@") > -1)
    {
        if (TextEmail.Text.IndexOf(".", TextEmail.Text.IndexOf("@")) > TextEmail.Text.IndexOf("@"))
            bValidEmail = true;
    }
    if (bValidEmail == false)
    {
        bValid = false;
        LabelMsg.Text += "Es wurde keine gültige Email Adresse eingegeben. ";
    }
    if (TextNachricht.Text.Length == 0)
    {
        bValid = false;
        LabelMsg.Text += "**Email** darf nicht leer sein. ";
    }
```

Aufgabe

```
MailMessage newMail = new MailMessage();
newMail.To.Add(new MailAddress("service@test.de"));
newMail.From = new MailAddress("service@test.de", "Test Kontaktformular");
newMail.ReplyTo = new MailAddress(TextEmail.Text);
newMail.Subject = "Feedback von " + TextName.Text;
newMail.IsBodyHtml = true;
newMail.Priority = MailPriority.Normal;
newMail.Body = "<html>Hallo, die folgende Nachricht wurde über Dein Kontaktformular verschickt:\r\n<br>";
newMail.Body += "Absender:\t" + TextVorname.Text + " " + TextName.Text + "\r\n<br>";
newMail.Body += "Email-Adr:\t" + TextEmail.Text + "\r\n<br>";
newMail.Body += "Tel.:\t\t\t" + TextTel.Text + "\r\n<br>";
newMail.Body += "Subject:\t\t" + TextSubject.Text + "\r\n<br>";
newMail.Body += "Nachricht:\t" + TextNachricht.Text + "</html>";

SmtpClient mailClnt = new SmtpClient("smtp.1und1.de");
mailClnt.UseDefaultCredentials = false;
mailClnt.Credentials = new System.Net.NetworkCredential("spwin@test.de", "Email$GeheimesPwd");
```

```
TextNachricht.Text = newMail.Body;
if (bValid)
{
    mailClnt.Send(newMail);

    TextName.Text = "";
    TextVorname.Text = "";
    TextEmail.Text = "";
    TextTel.Text = "";
    TextSubject.Text = "";
    TextNachricht.Text = "";

    LabelMsg.Text = "Die Email wurde versendet.";
}
```

Erstellung einer WebForms-Anwendung

Für die Erstellung einer WebForms- Anwendung stellt Visual Studio ein Template bereit, welches bereits über zahlreiche Funktionen und integrierte Bibliotheken verfügt.

Die Vorgehensweise bei der Entwicklung einer WebForms-Anwendung kann allgemein mit den folgenden Schritten beschrieben werden.

1. Erstellung einer Applikation / Projektmappe mit Hilfe des entsprechenden Templates
2. Ggf. Hinzufügen weiterer benötigter Bibliotheken.
    - Verfügbare Bibliotheken können als sogenannte NuGet-Pakete direkt über die IDE hinzugefügt werden.
3. Erstellung der Daten-Ebene (Data Acces-Layer). D.h. Erstellung einer Datenbank mit Tabellen und der dazugehörenden Klassen zur Verarbeitung der Daten.
4. Erstellung der Benutzeroberfläche und der Logik-Ebene. Programm-Navigation (Menüs), Webseiten mit Steuerelementen und den korrespondierenden serverseitigen Klassen.
5. Testen der Anwendung im lokalen Debug-Modus.
6. Veröffentlichen der Anwendung auf einem Webserver (Deployment) und Testen der Anwendung.

Die Entwicklung ist insgesamt relativ umfangreich und komplex. In der Regel wird eine Anwendung daher schrittweise erstellt. Die Schritte 2 bis 6 werden für abgegrenzte Funktionalitäten iterativ durchgeführt.

Dateien einer ASP.NET WebForms Applikation

Datei	Bedeutung
Default.aspx	Die Webseite, die beim Aufruf der Anwendung in einem Browser angezeigt wird. (Startseite)
Site.Master	Eine Seite, in der ein einheitliches Layout für alle Seiten der Anwendung und das Standardverhalten implementiert werden.
Global.asax	Optionale Datei mit Code für die Ereignisbehandlung von Application- und Session-Level Ereignissen, die durch ASP.NET oder HTTP Module ausgelöst werden.
Web.config	Konfigurationsdatei für die Applikation

ASP.NET WebForms Vorlage

Visual Studio erstellt eine Projektmappe mit vordefinierten Funktionalitäten.

Dazu gehören Seiten, die üblicherweise in Applikationen vorkommen, wie: Home.aspx, About.aspx und Contact.aspx.

Ebenfalls enthalten ist die Funktionalität der Benutzerverwaltung (Registrieren, An- und Abmelden) auf Basis von ASP.NET Identity..

Element	Beschreibung
Identity	Benutzer-Credentials werden in einer Datenbank verwaltet, die automatisch erstellt wird. Die Seiten mit der entsprechenden Funktionalität sind im Ordner „Account“ enthalten.
Datenbank	Eine WebForms Anwendung verfügt über eine lokale SQL-Server-Express Datenbank „localdb“. Diese ist in dem Ordner „AppData“ enthalten.
Master Page	Globale Funktionalitäten und einheitliches Layout (Menü, Logo,…) können in einer Master Page definiert werden.

ASP.NET WebForms Vorlage

Element	Beschreibung
HTML 5	Die ASP.NET App. unterstützt HTML 5.
Modernizr	Mechanismus, der es ermöglicht, mit Browsern, die kein HTML 5 unterstützen eine entsprechende Funktionalität anzubieten. (JavaScript Bibliothek, eingebunden als NuGet-Paket)
Bootstrap	Bootstrap ist ein von Twitter entwickeltes Framework, welches Layout und Design Themes verwendet, um „responsive“ Webseiten zu erstellen, die auf einer Vielzahl von Geräten dargestellt werden können.

Entity Framework

Die Verwaltung der Anwendungsdaten einer ASP.NET Applikation wird üblicherweise mit Hilfe einer SQL-Server Datenbank erfolgen.

Der Entwickler muss in diesem Fall eine relationale Datenbank entwerfen und erstellen, welche die Daten des objektorientierten Programms in geeigneter Weise abbilden kann. Ebenso müssen Funktionalitäten in den Klassen des Programms enthalten sein, um die erforderlichen Datenzugriffe bereit zu stellen. Mindestens sind hier die so genannten CRUD-Zugriffe zu implementieren (CRUD: Create, Read, Update, Delete).

Zur Vereinfachung dieser Tätigkeiten werden ORM-Tools (Object-Relational Mapping) eingesetzt, welche in der Lage sind, objektorientierte Datenstrukturen eines Computerprogramms und relationale Datenstrukturen einer Datenbank zu verknüpfen.

Entity Framework (EF) ist das von Microsoft entwickelte ORM-Tool.

Die zugehörigen Bibliotheken gehören nicht zum ASP.NET WebForms Template. Sie könenn als NuGet-Paket einem Projekt hinzugefügt werden. In Visual Studio können NuGet-Pakete hinzugefügt werden über:

Extras -> Bibliotheks-Paket-Manager -> NuGet Pakete für Projektmappe verwalten

Entity Framework

EF unterstützt zwei Ansätze zur Modellierung.

EF Code First	EF Model First
Die Definitionen von Klassen und Attributen werden im Programm-Code mit Annotationen versehen. Dadurch wird die Abbildung in der Datenbank gesteuert. Bei Ausführung des Programms werden die Datenbank und die entsprechenden Tabellen erzeugt.	Die Definition der Entity-Klassen erfolgt mit Hilfe eines Designers graphisch interaktiv. Ausgehend von diesem Modell (Diagramm) werden die Klassen automatisch erstellt. Die entsprechende Datenbank und die Tabellen werden ebenfalls automatisch mit Hilfe von generierten SQL-Skripten erzeugt.

Im Rahmen der hier behandelten Anwendungen wird der Modellierungsansatz Entity Framework Code First ausgewählt.

Entity Framework Code First

EF Annotations werden in die Klassendefinition eingefügt.

```
[Serializable]
   public class n2Person
   {
       public int ID { get; set; }
       public string UsrName { get; set; }
       public string UsrVorName { get; set; }
       [DataType(DataType.DateTime)]
       public DateTime? DOB { get; set; }
       public string Email { get; set; }
       public string TelNo { get; set; }
       public virtual n2Adress Adress { get; set; }
       [Display(Name = "Comment"),
        DataType(DataType.MultilineText)]
       public string Comment { get; set; }

       [ForeignKey ("Picture")]
       public int? PictureID { get; set; }
       public virtual n2Document Picture { get; set; }

       [NotMapped]
       public bool bselected { get; set; }
```

Klasse ist serialisierbar. Optional. (erforderlich z.B. für Konvertierung in JSON)

Hat ein Attribut den Namen ID, wird dieses automatisch zum Primärschlüssel in der erstellten Tabelle. (Jedes Attribut kann mit [PrimaryKey] als Primärschlüssel gesetzt werden.)

Bei Bedarf kann der Zieldatentyp (der SQL-Server DB) explizit angegeben werden.

Kennzeichnung eines Fremdschlüssels

Attribute, die nicht in der Datenbank erstellt werden sollen.

Entity Framework Code First

Die Verbindung zwischen Klassen und Datenbank wird mit Hilfe eines DbContext-Objekts hergestellt

Dieses Objekt stellt Methoden für Abfragen (Queries). Verfolgung und Speicherung zur Verfügung.

```
 public class ApplicationDbContext : DbContext    {
        public ApplicationDbContext()
            : base("DefaultConnection")
        {
        }
        public DbSet<UsrProfile> UsrProfiles { get; set; }
        public DbSet<n2Object> n2Objects { get; set; }
//--> FieldServiceContext
        public DbSet<n2Adress> n2Adresses { get; set; }
        public DbSet<n2Person> n2Persons { get; set; }
        public DbSet<n2PLZ> n2PLZ { get; set; }
}
```

Zur Abbildung der Klasse n2Person wird eine Tabelle „n2Persons“ verwendet.

Entity Framework Code First

Wenn das erste Objekt in der DB angelegt wird, erstellt Entity Framework eine neue Datenbank an der Default location. Die verwendete Datenbank kann auch explizit angegeben werden, wenn beispielsweise zwischen Test- und Produktiv-System unterschieden werden muss. Dazu wird die Verbindung zur DB mit Hilfe einer DB-Connection in der Datei Web.config eingetragen.

Der Eintrag erfolgt in der Sektion „connectionStrings“.

```
<connectionStrings>
   <!--Lokale DB (Development)-->
   <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0; AttachDbFilename=|DataDirectory|\aspnet-
WebApplication1-20140114055124.mdf; Initial Catalog=aspnet-WebApplication1-20140114055124;
         Integrated Security=True" providerName="System.Data.SqlClient" />
   <!--SQL Server (Production)-->
   <!--<add name="DefaultConnection"
      connectionString="Data Source=db51808765594.db.1and1.com;
      Initial Catalog=db51974458594;
      User Id=dbo51801234866594;
      Password=pass$wort;
      User Instance=True;
      multipleactiveresults=True&amp;quot;"
      providerName="System.Data.SqlClient" />-->
 </connectionStrings>
```

Entity Framework Code First

Die Verbindung zwischen Klassen und Datenbank wird mit Hilfe eines DbContext-Objekts hergestellt

Dieses Objekt stellt Methoden für Abfragen (Queries). Verfolgung und Speicherung zur Verfügung.

Es muss daher eine Kontext-Klasse erstellt werden, welche die Zuordnung von Entity-Klassen zu Tabellen enthält.

```
 public class ApplicationDbContext : DbContext    {
        public ApplicationDbContext()
            : base("DefaultConnection")
        {
        }
        public DbSet<UsrProfile> UsrProfiles { get; set; }
        public DbSet<n2Object> n2Objects { get; set; }
//--> FieldServiceContext
        public DbSet<n2Adress> n2Adresses { get; set; }
        public DbSet<n2Person> n2Persons { get; set; }
        public DbSet<n2PLZ> n2PLZ { get; set; }
}
```

Zur Abbildung der Klasse n2Person wird eine Tabelle „n2Persons“ verwendet.

Entity Framework Code First

Die Erstellung einer Datenbank erfolgt bei der ersten Verwendung d.h. wenn das erste Objekt einer Entity-Klasse gespeichert wird.

Eine Initialisierungsklasse wird benötigt, um die Datenbank zu erstellen. In dieser Klasse werden mit Hilfe sogenannter Seed-Methoden gegebenenfalls auch initiale Daten in die Datenbank eingegeben.

```
namespace WebApplication1.Models
{
    public class ProductDatabaseInitializer : DropCreateDatabaseAlways<ProductContext>
    {
        protected override void Seed(ProductContext context)
        {
            GetStatusColl().ForEach(c => context.n2Status.AddOrUpdate<n2Status>(c)); //    .Add(c));
            int iRet = context.SaveChanges();
        }
        private static List<n2Status> GetStatusColl()
        {
            var Status = new List<n2Status> {
                new n2Status { value = -2, n2Name="deleted„ },
                new n2Status { value = -1, n2Name="disabled"},
                new n2Status { value = 0, n2Name="new" },
                new n2Status { value = 1, n2Name="checkedIn" },
                new n2Status { value = 2, n2Name="checkedOut“ }, };
            return Status;
        }
```

Entity Framework Code First

Der Zugriff auf die Entity-Klassen innerhalb des C#-Programms erfolgt dann nach dem Schema:

1. DB Kontext-Objekt erstellen

```
public void DoSomething(int nID)
{
    WebApplication1.Models.ApplicationDbContext _db = new WebApplication1.Models.ApplicationDbContext();;
    n2Person objRet;
    IQueryable<n2Person> query = _db.n2Persons.Where(p => p.ID == nID).Include("Adress");
    if (query.Count() > 0)
    {
        objRet = query.First();
    }
    else
    {
        return ;
    }
    objRet.UsrName = "Meier";
    objRet.UsrVorName = "Hans";
    objRet.Adress.City = "Duisburg";
    _db.SaveChanges();
    return;
 }
```

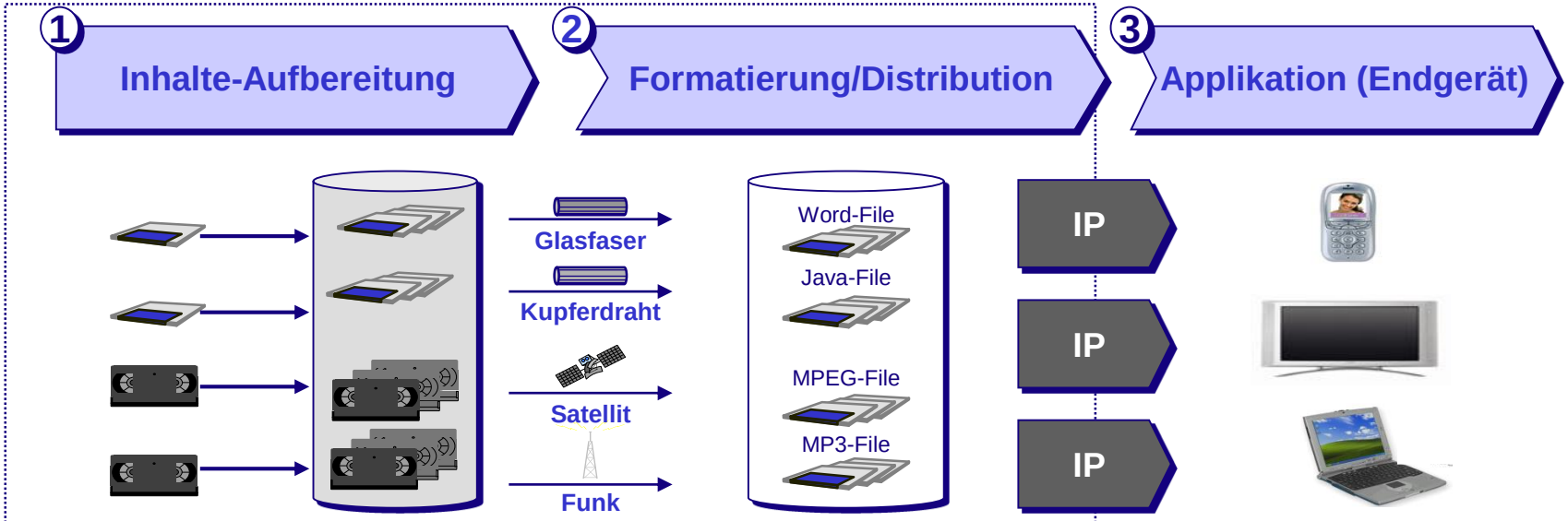
2. Mit Hilfe einer Query ein oder mehrere Objekte der DB abfragen

3. Objekt(e) bearbeiten

4. Mit Hilfe des Kontext-Objekts Änderungen in der DB speichern. (wirkt nur auf Objekte, die im selben Kontext erstellt wurden)

Mobility – Mobile Anwendungen

Technologie: IP-Netzkonvergenz



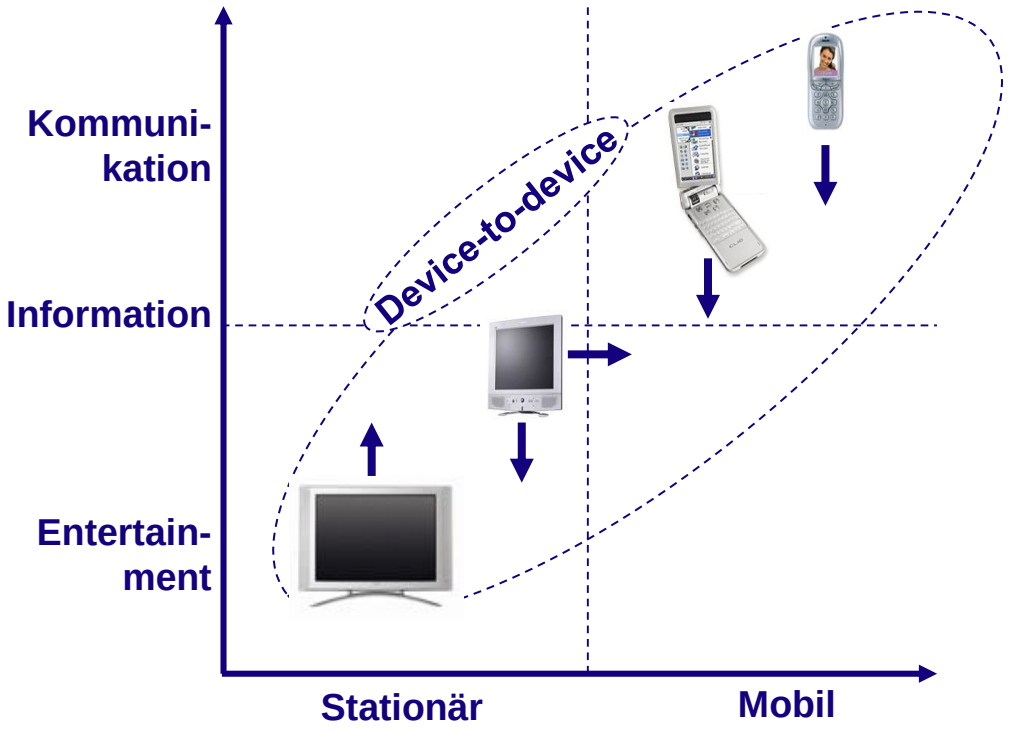
- Inhalte müssen für verschiedene Screen-Größen und Endgeräte bearbeitet werden
- Anbieter "Web-Service-Farm“: Inhalte-Anbieter, ISPs/Portale, Software-Anbieter

- Digitale Signale müssen in IP-fähiges Format umgewandelt Endgeräten zugeordnet werden
- Anbieter Serverparks und Netzinfrastruktur: Netzbetreiber, Mobilfunkbetreiber, IT-Integratoren

- Eine IP-fähige Applikation erlaubt Empfang und Darstellung der Inhalte
- Mögliche Anbieter: Software-Unternehmen, Endgeräte-Hersteller

** Bereits im digitalen Format, keine Signalumwandlung mehr notwendig*

Technologie

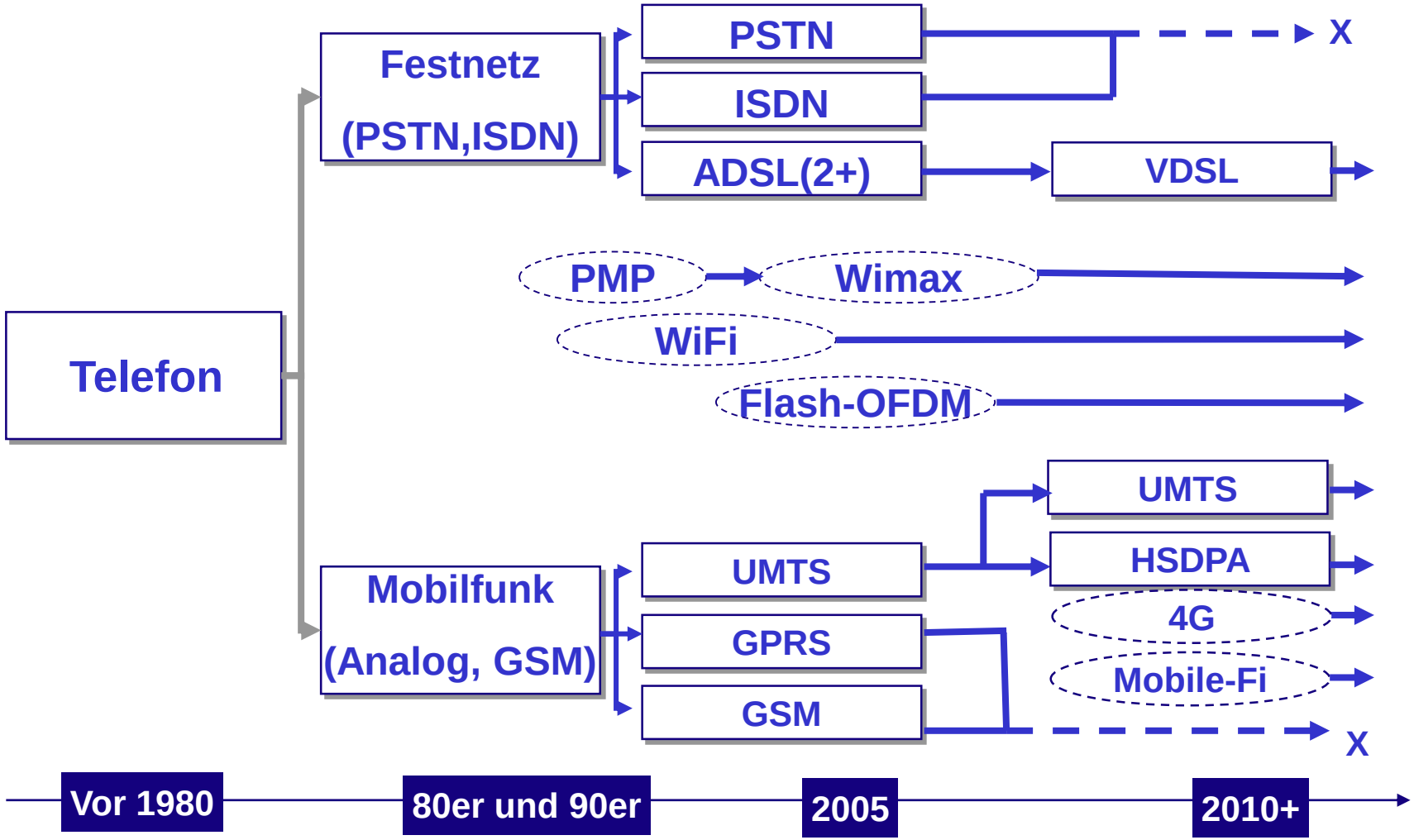


- Technische Entkopplung der Plattform-Inhalte-Endgeräte-Logik
- „Alle Geräte können alles“
- Begriffe der Endgeräte-bezogenen Nutzung wie „Telefonieren“ oder „Fernsehen“ verschwinden
- Device-to-device Kommunikation verbindet die Endgeräte
- Bildschirm-Größe und Mobilität werden zum Haupt-Kriterium

→ *Potentielle Aktionsbereiche*

Quelle: Experteninterviews, Delphi, BAH Analyse

Entwicklung der Übertragungsplattformen



Mobile Devices

10 Millionen Geräte bis 2020

61% der CIO‘s setzen „Mobility“ als Priorität

45% gesteigerte Produktivität durch mobile Apps

Mobility: Anwendungsbereiche

Business to Enterprise

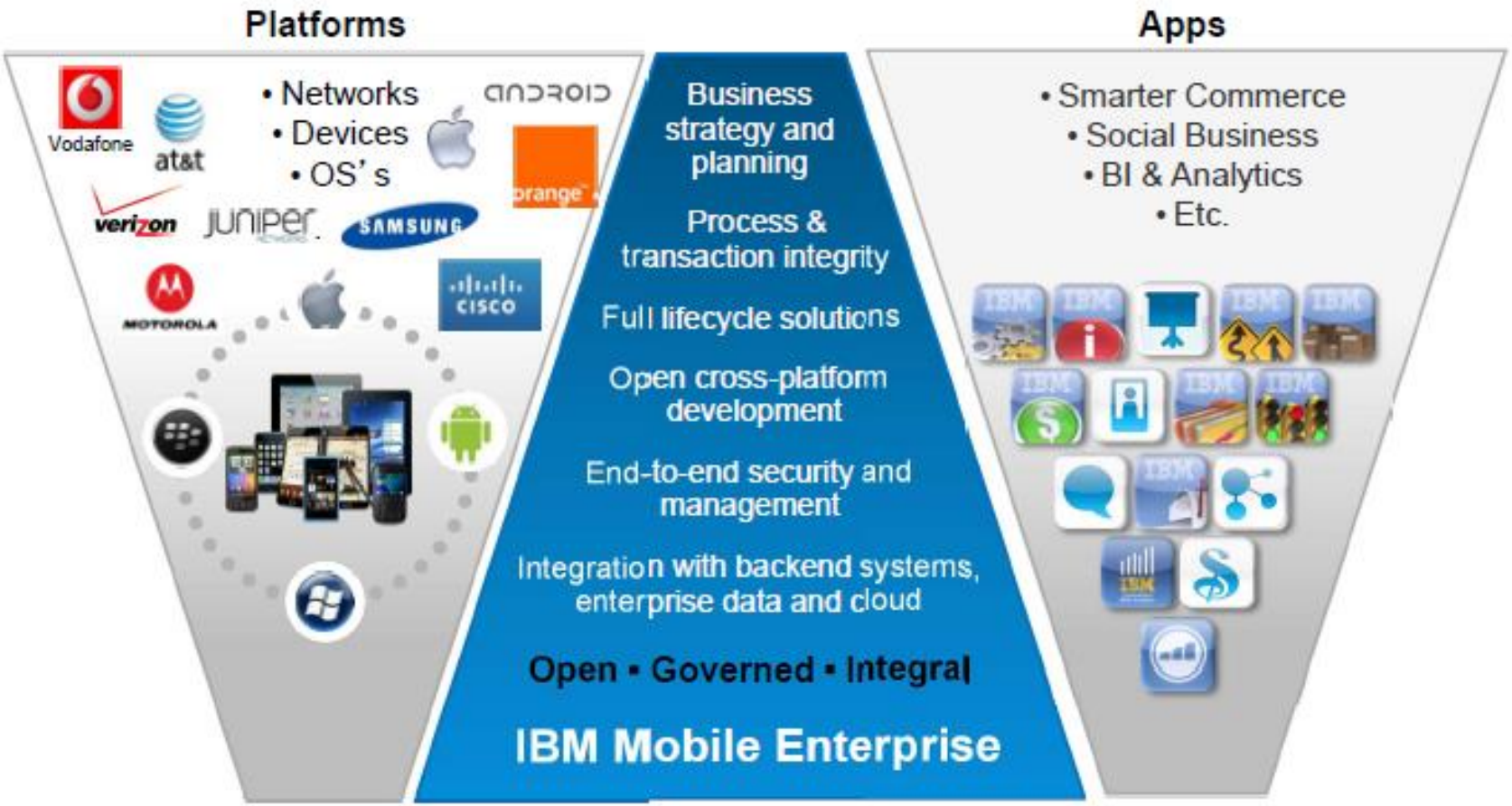
- Erhöhung der Produktivität
- Ausdehnung bestehender Anwendungen für Außendienst / Home Office
- Kostensenkung für Treibstoff, Flotten-unterhalt
- Verbesserte Erreichbarkeit und beschleunigte Entscheidungsprozesse
- Optimierte Geschäftsprozesse mit externen Partnern
- Kostensenkung (BOD)
- ….

Business to customer

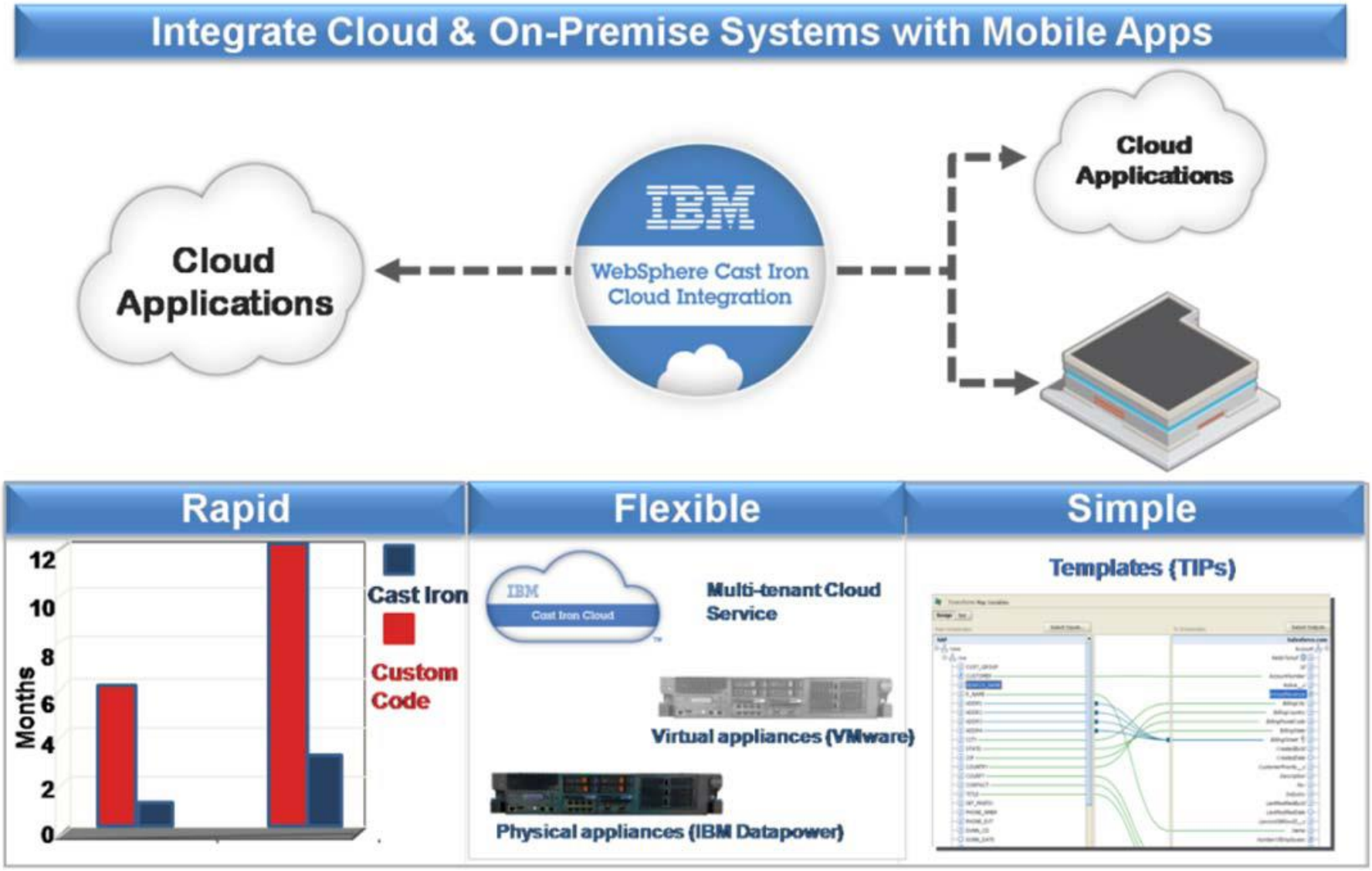
- Verbesserte Kundenzufriedenheit
- Kundenbindung
- Personalisierte Angebote
- Wettbewerbsmerkmal
- Verbesserung des Markenimages
- Analyse Kaufverhalten der Kunden

….

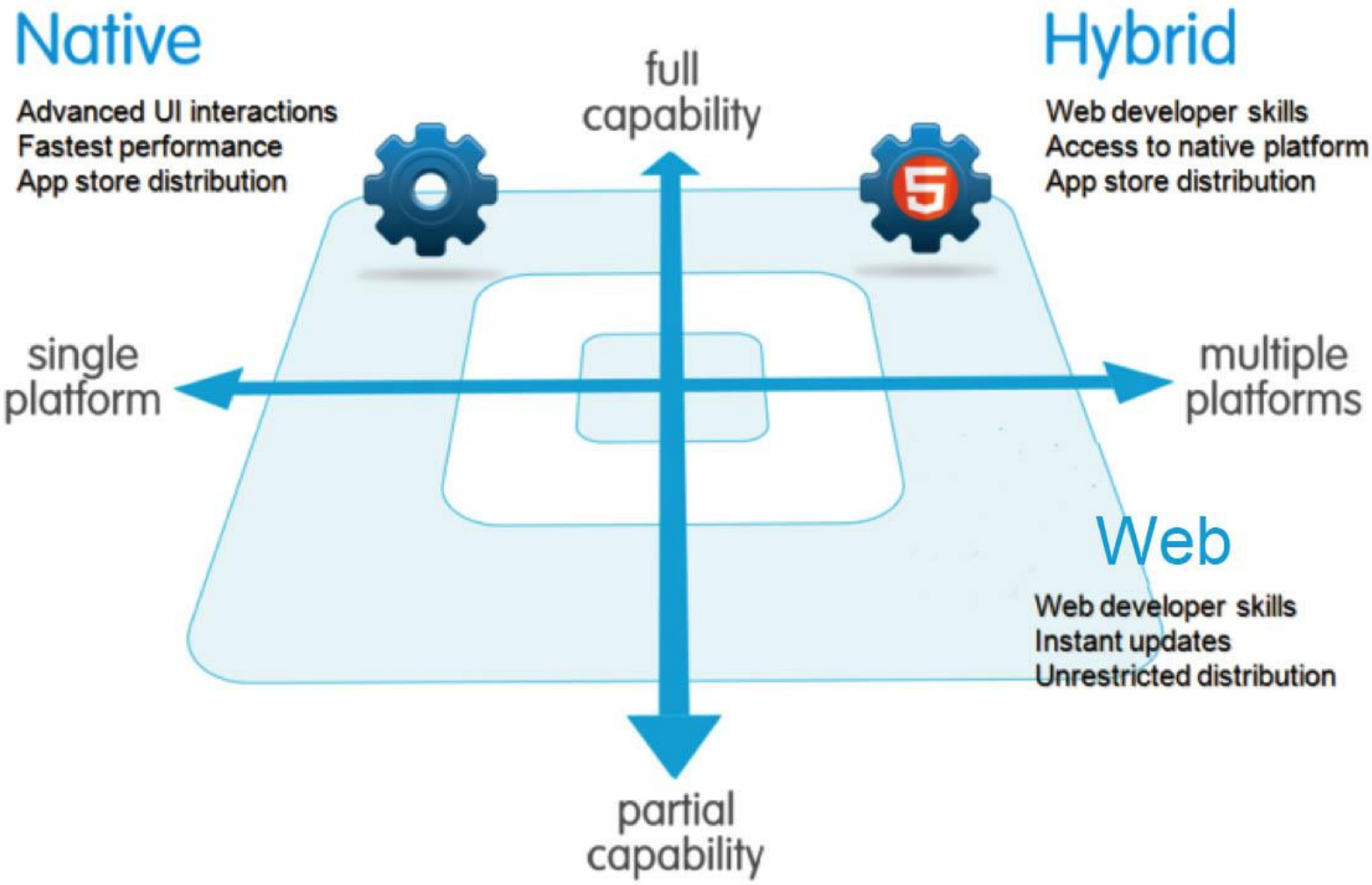
Mobility: Anwendungsbereiche



Mobility: Anwendungsbereiche



Mobile Anwendungen / Apps



Mobile Anwendungen / Apps

	Web App	Hybrid App	Native App
Development costs	Medium	Medium	High
Maintenance/Update	Easy	Hard	Hard
Experience of user	Fair	Good	Very Good
Certification of stores	No	Yes	Yes
Install	No	Need	Need
Cross-Platform	Very Good	Good	Bad
Languages	HTML 5 / CSS / Java Script	HTML 5 / CSS / Java Script	Objective C/C++ (IOS)

Native Apps

Weit verbreitete Betriebssysteme (OS) für mobile Endgeräte.
(Tablets, Smartphones, Notebooks….)

Native Apps

1. Used in a particular platform or device.
2. Operating based on local operating systems
   IOS, Android, WP

3. Applications are expensive to develop

3.3 Native App

1.between native app and web app

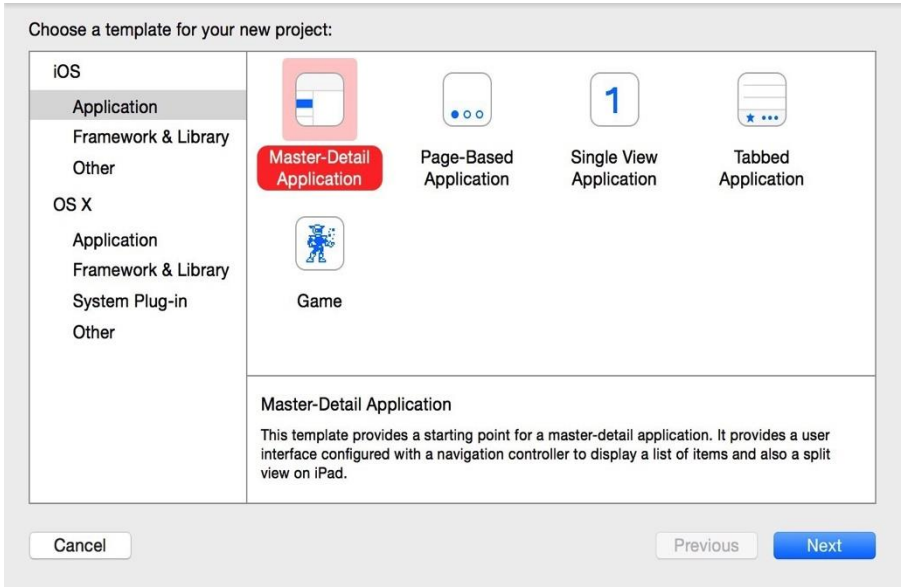
2.coded in both browser-supported and computer language

1.software that runs in a web browser

2.creat in a browser-supported ramming language

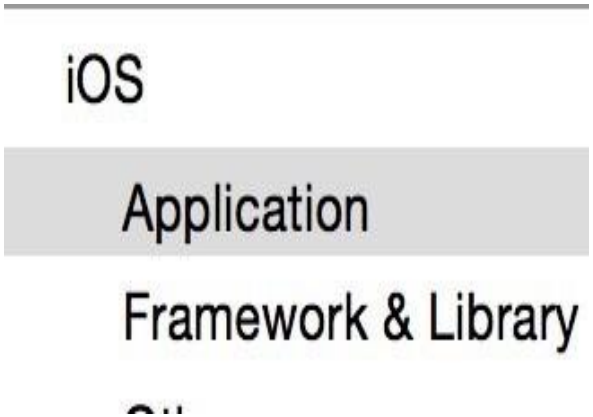
3.3 Native App

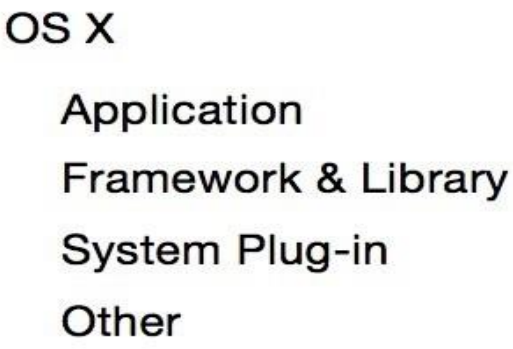
- A new project Interface in Xcode, as shown below.

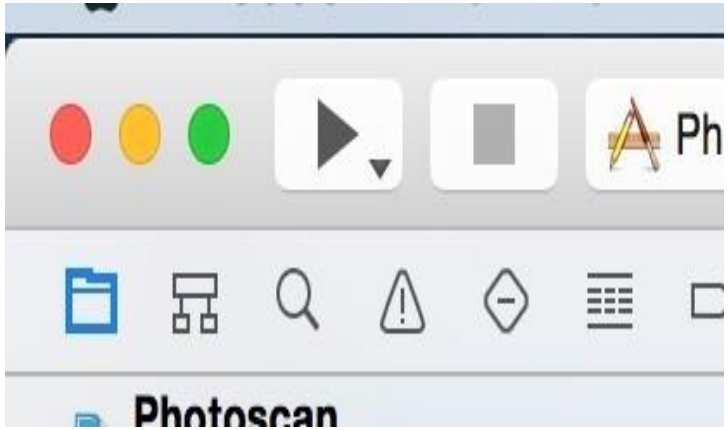


3.3 Native App

- A new project Interface in Xcode, as shown below.

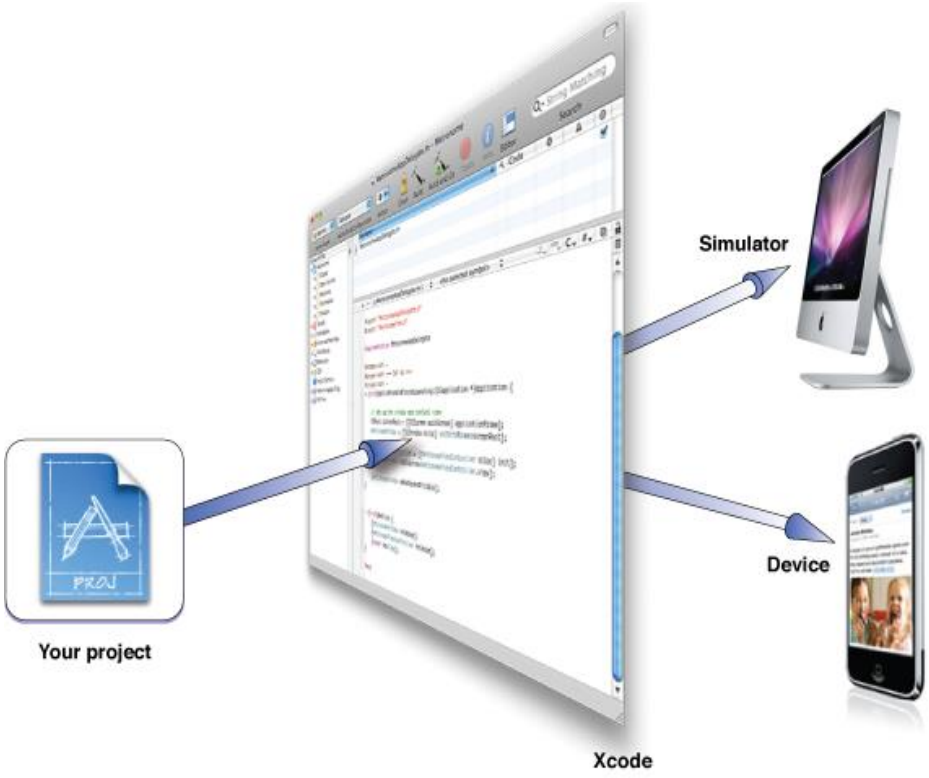


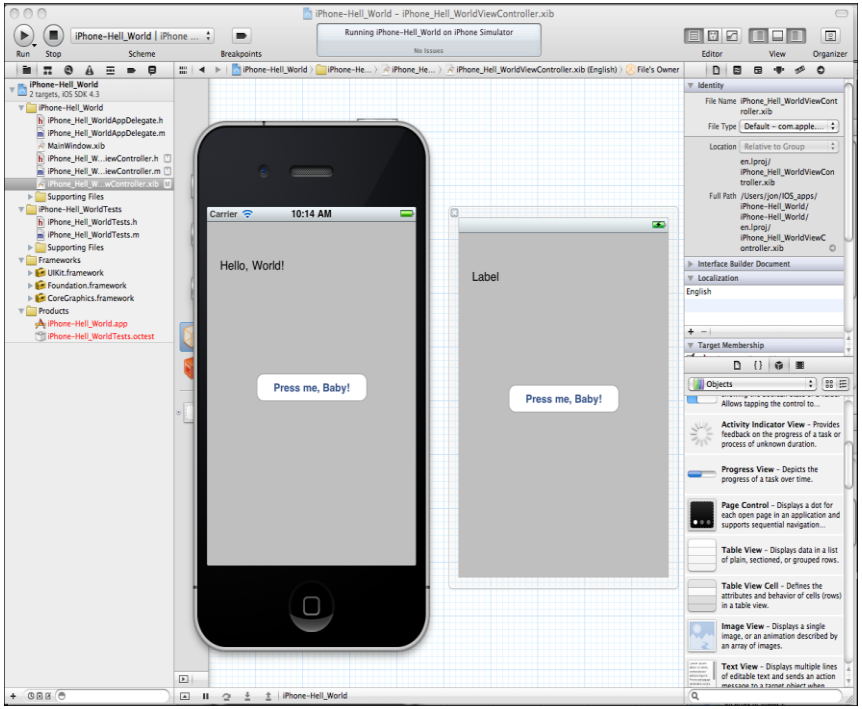




3.3 Native App

- Test the App by using computer or Iphone,Create a simulator for iPhone，the simulator provides a local environment to test the application





Features

- Multimedia
- APIs für
  - Geolocation
  - Sensoren
  - Kamera
  - Multimedia
  - Dateihandling
- UI-Elemente z.B. Drag and Drop
- Steuerelemente

Heute können Web-Anwendungen erstellt werden, die effektiv eingesetzt werden können.

Service Management System

Praxis-Beispiel: Entwicklung eines Web Portals für „Field Service Management“

“Produkt”: Tracking Device

- Batterie betrieben
- GPS Empfänger
- Satelliten Kommunikation
- Verschiedene Sensoren

SMARTONE
SATELLITE MANAGED ASSET READY TRACKER

Problem: (Kundennutzen?)

- Fischerboote, oftmals ohne Funk
- Keine Information über Notfall / Seenot
- Keine Kontrolle von
  - ⇒ Fanggebieten
  - ⇒ Schonzeiten

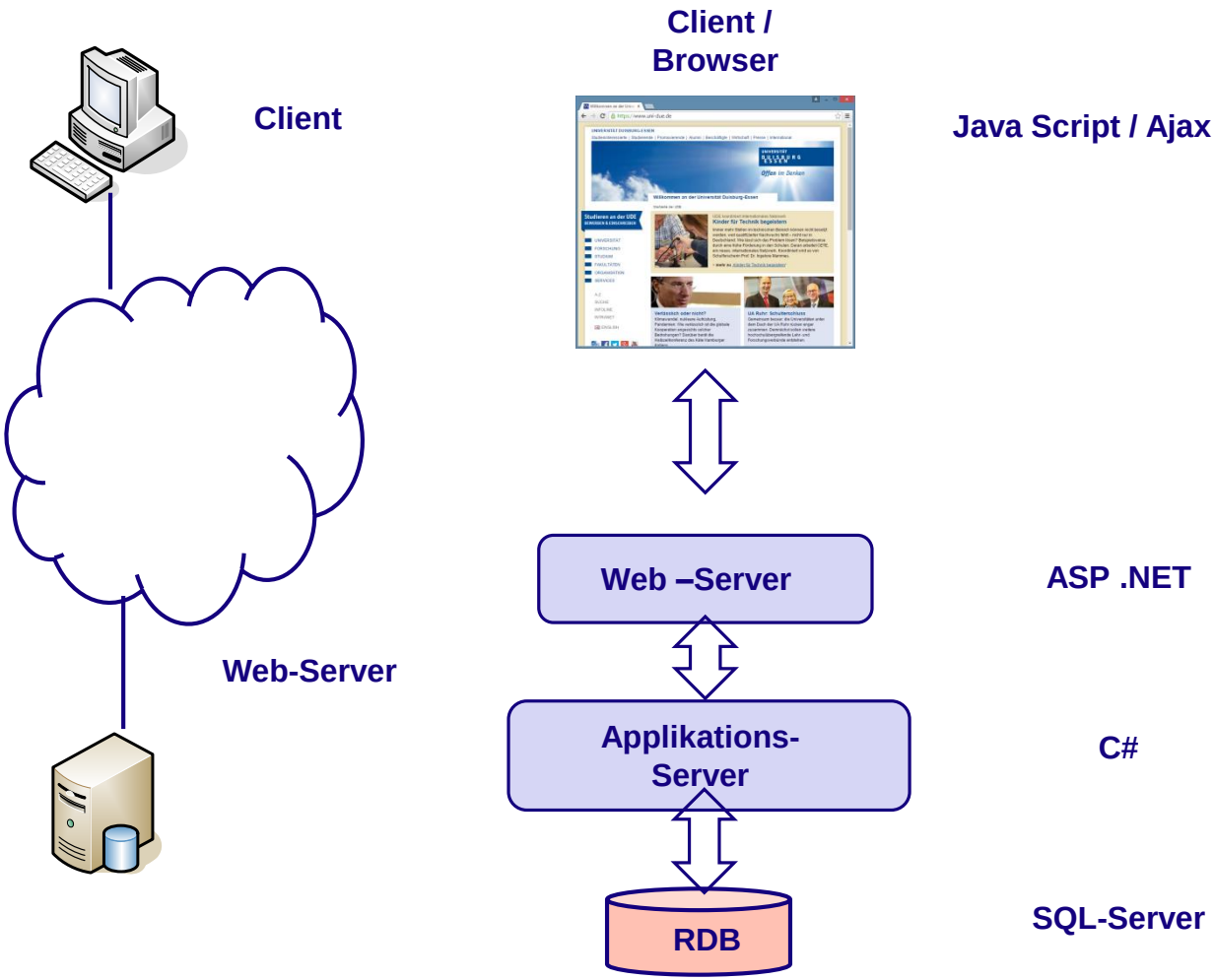
Dienstleistungs-Produkt:

- Installation der Hardware
- Wartung der Hardware
- Datenbank mit relevanten Schiffsdaten
- Cloud Applikation für “Logistik”
- Cloud Applikation für Tracking

Montage

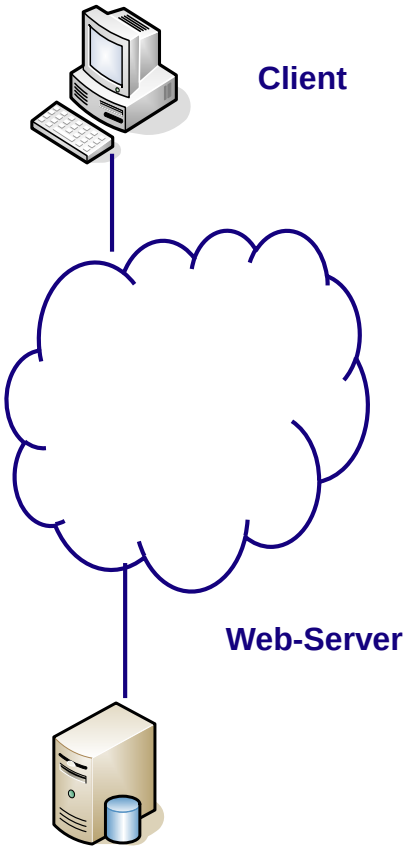
Realisierung mit Cloud – Technologie

Microsoft ASP.NET



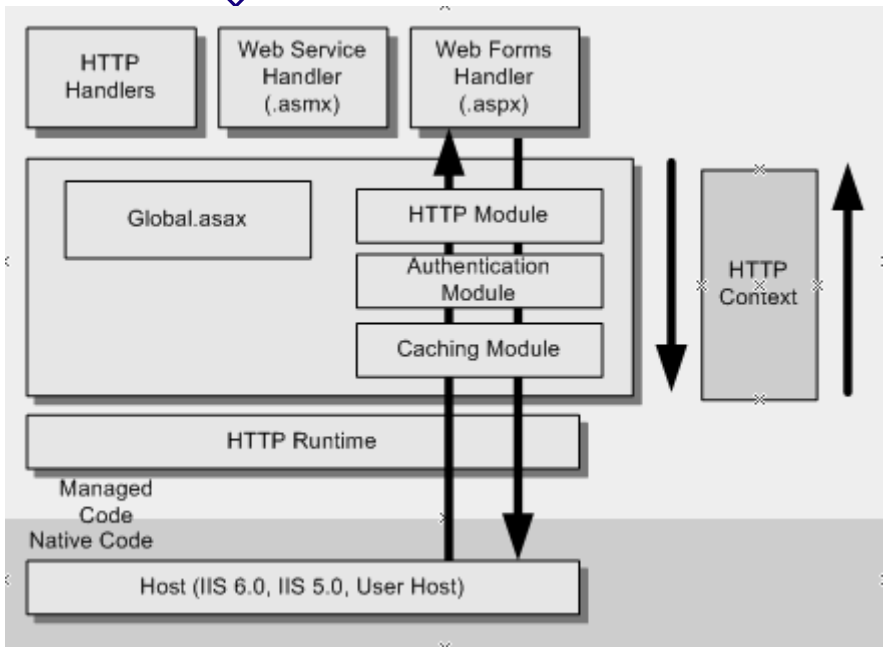
Realisierung mit Cloud – Technologie

Microsoft ASP.NET



Client / Browser

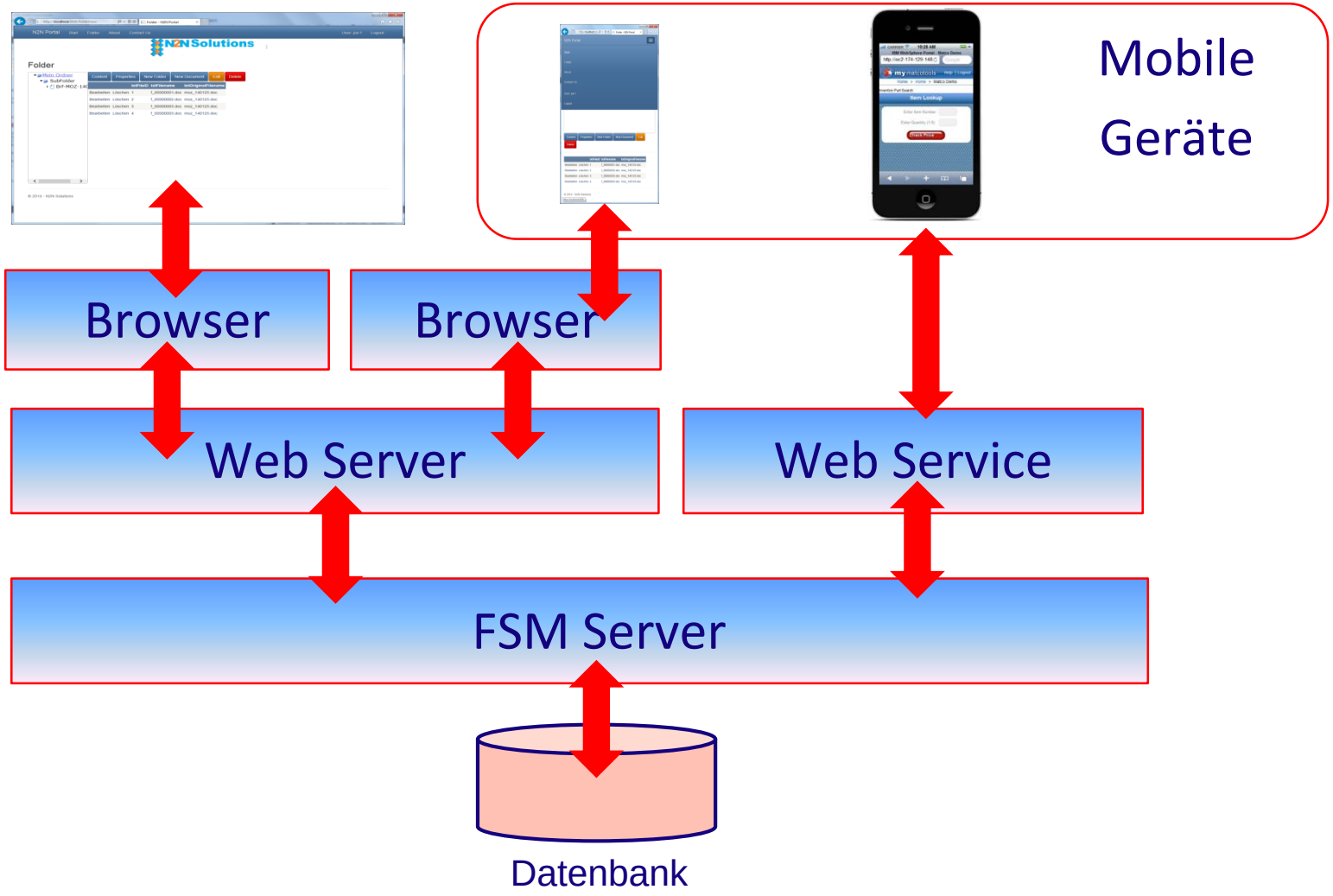
Java Script / Ajax



Field Service Management Anwendung

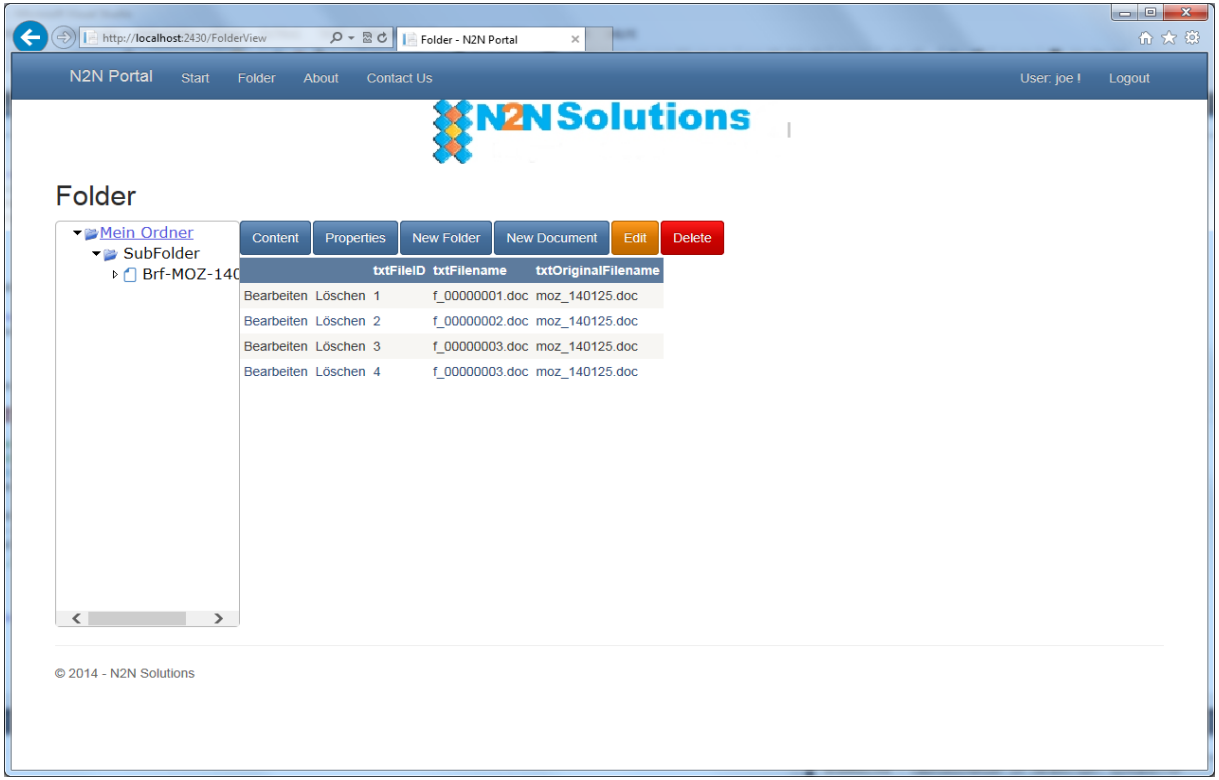
- Basierend auf Microsoft Technologien
- Architektur: ASP.NET Cloud Application
- Datenbank: MS SQL-Server
- Schlagworte / Verwendete Technologien:
  - ⇒ Entity Framework (Database Design by Code First Development)
  - ⇒ Bootstrap UI (Web Anwendungen für Desktop- und mobile Geräte)
  - ⇒ HTML 5, CSS 3
  - ⇒ Programmiersprachen: C#, JavaScript
  - ⇒ Globalisierung / Lokalisierung: ASP.NET Global Ressources
  - ⇒ Intuitive Benutzeroberfläche

Architektur

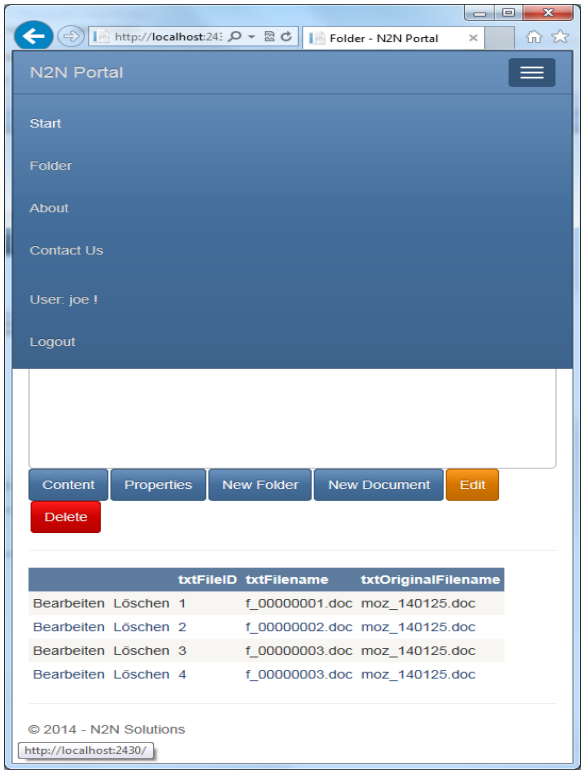


Erscheinungsbild

Steuerelemente und Inhalt passen sich dem Gerät an.

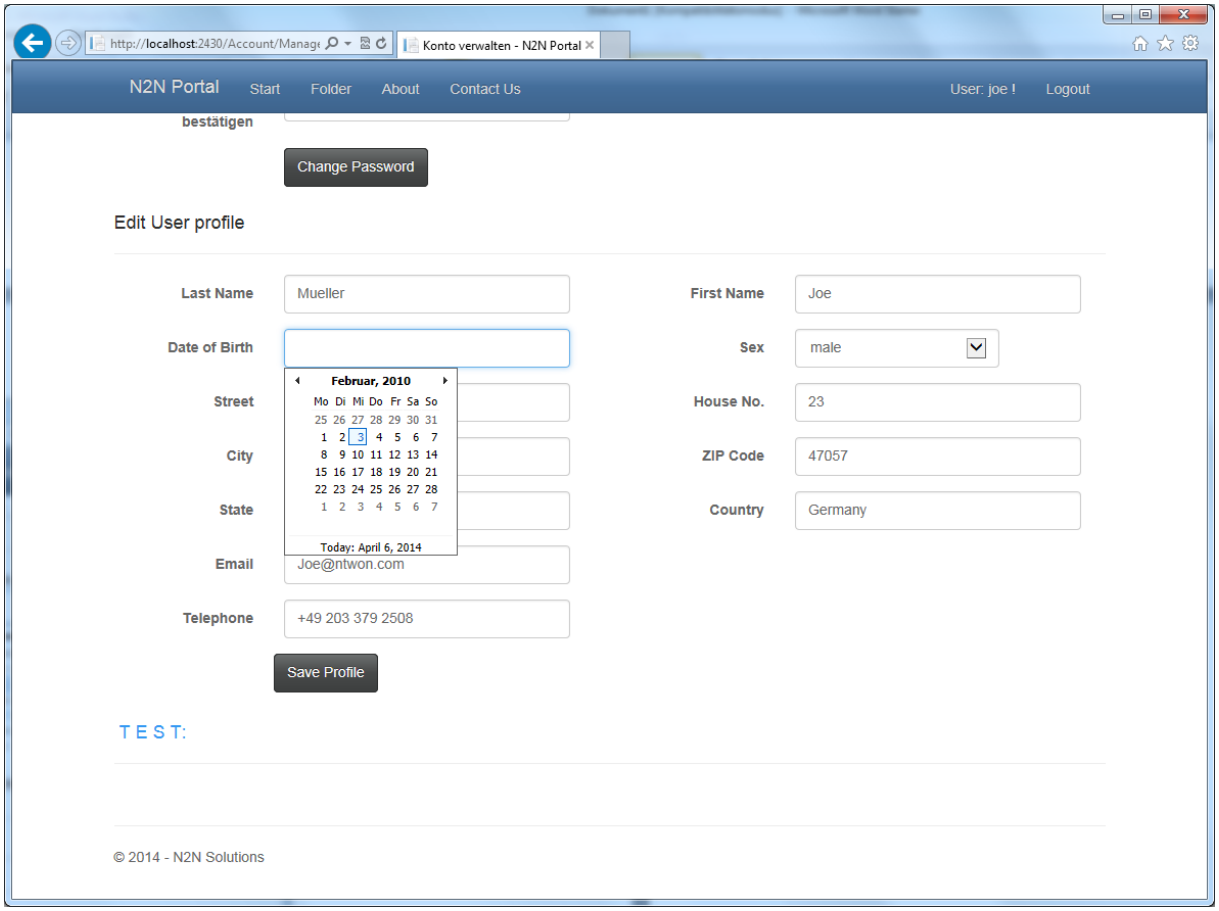


Desktop

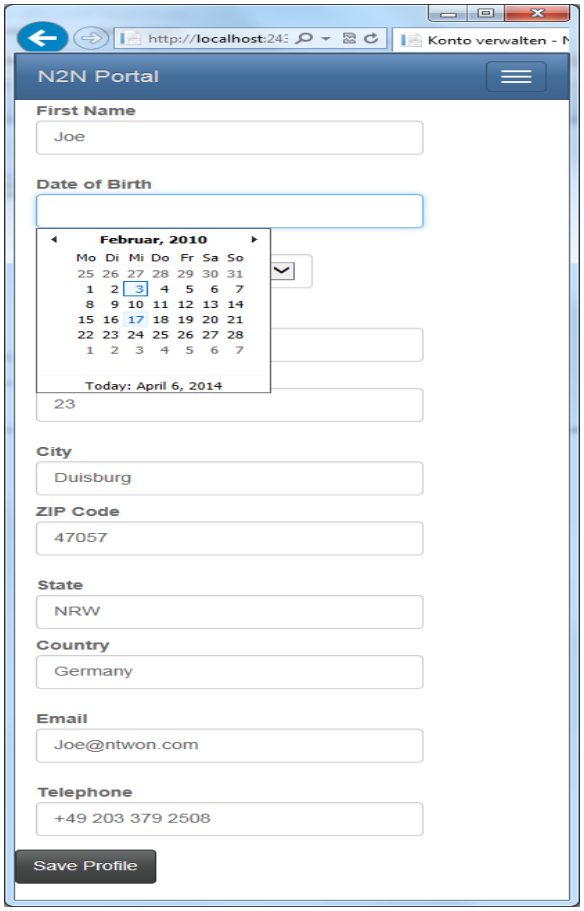


Mobile Device

Form: User Profile Data



Desktop



Mobile Device

Alle Texte (Label, Menu, Buttons, Überschriften…) sind in globalen Ressourcen hinterlegt.

Beim Starten der Webseite wird die Standardsprache / Kultur des Browsers verwendet. Zur Laufzeit kann die Sprache per Listenauswahl selektiert werden.

Unterstützte Sprachen: Englisch, Deutsch, Spanisch

(Hinzufügen weiterer Sprachen beschränkt sich auf das Übersetzen einer Ressource-Tabelle.)

Features / Offline Mode

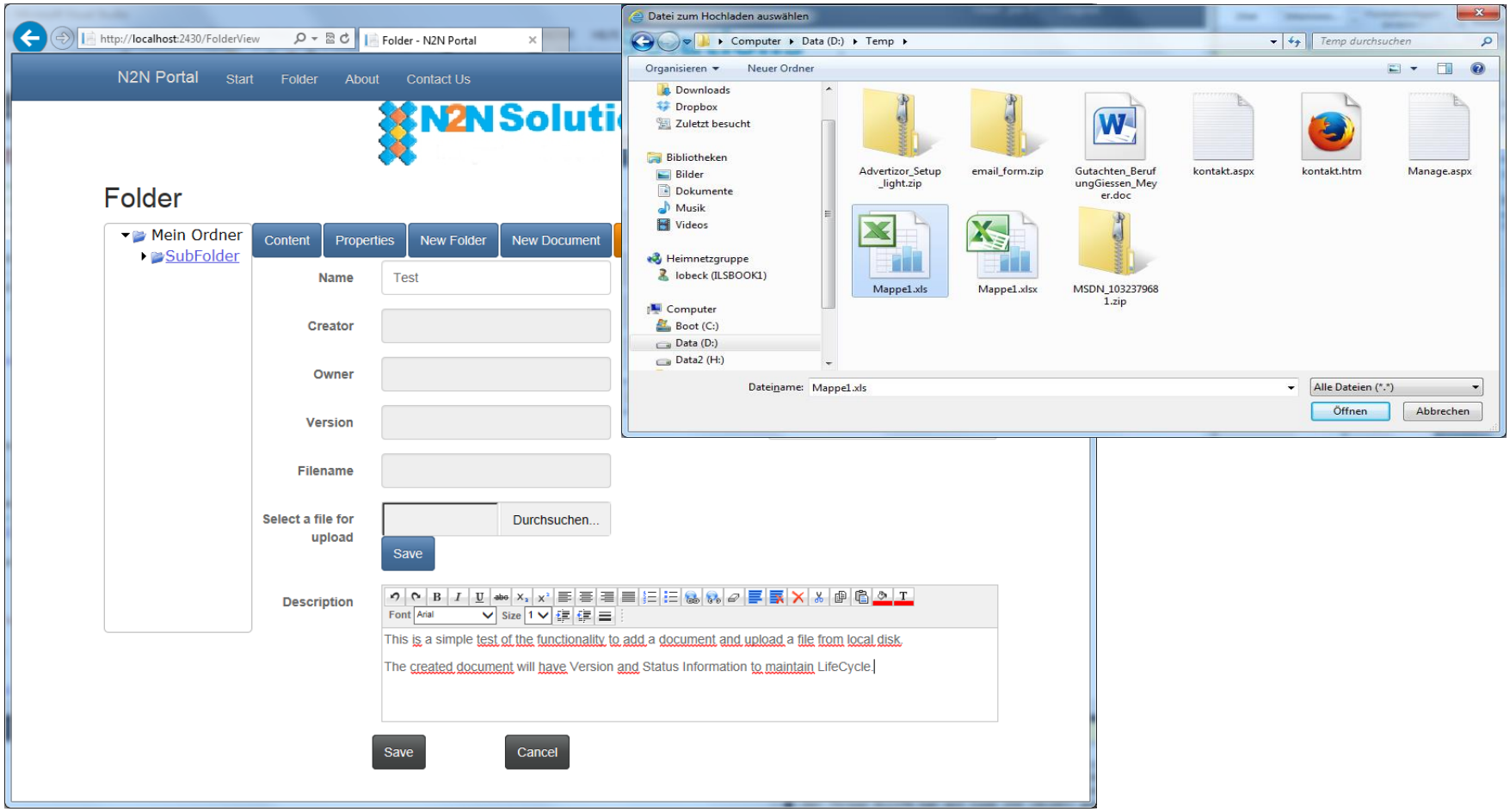
Da nicht an jedem Ort eine drahtlose Verbindung zum Server erwartet werden kann, wird ein “Offline” Modus für die Client-App des Service-Mitarbeiters verwendet.

Dazu werden die Möglichkeiten von HTML5 verwendet.

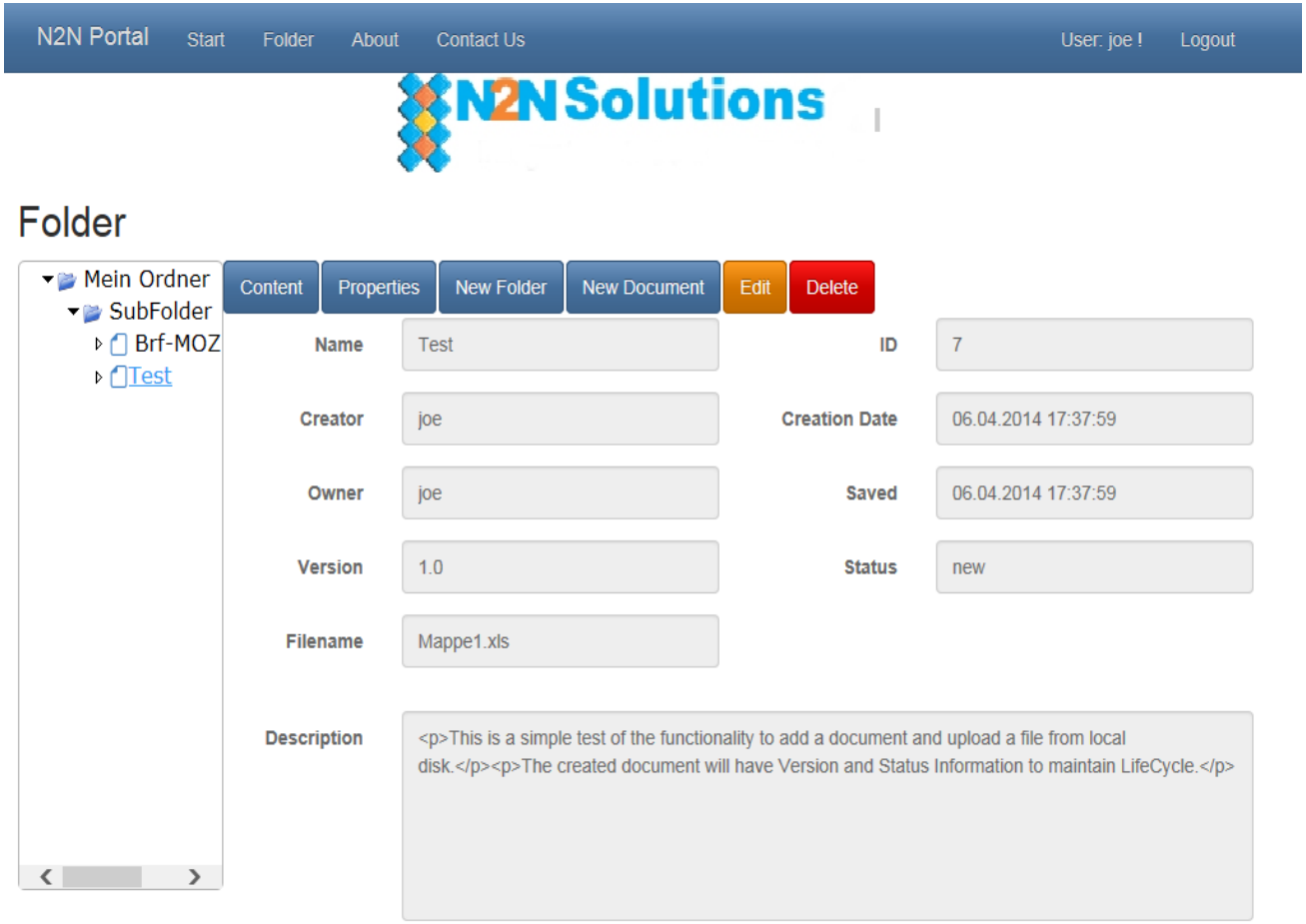
Ein angemeldeter Benutzer kann nach verschiedenen Kriterien Aufträge auschecken. (“Check Out”-Funktion) Diese Daten werden auf das mobile Gerät geladen und sind lokal verfügbar.

Durch eine “Check In”-Funktion, werden Daten von dem lokalen Gerät in das Portal hochgeladen.

Dokumentenverwaltung

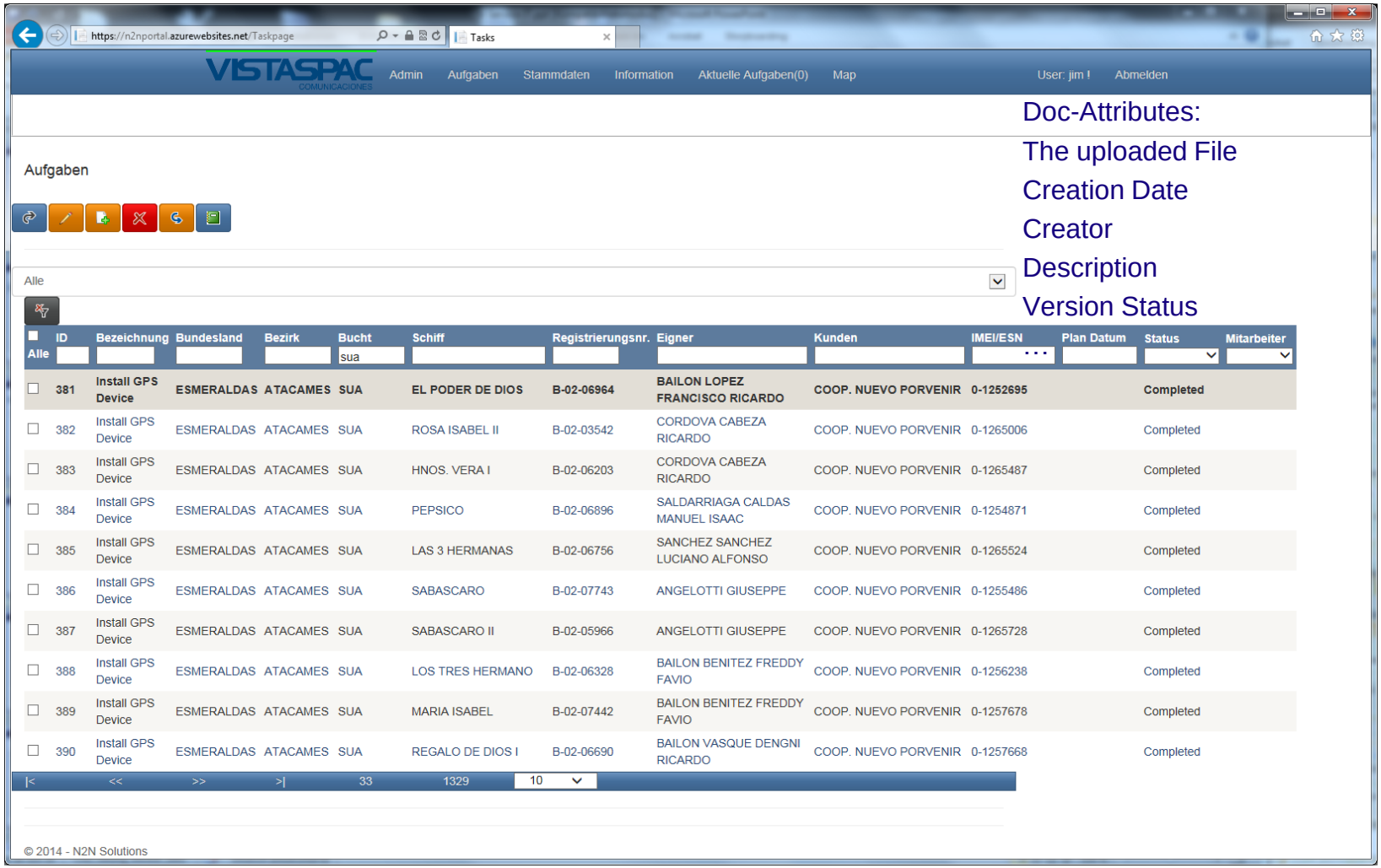


Erzeugtes Dokument



Doc-Attributes:
The uploaded File
Creation Date
Creator
Description
Version Status
…

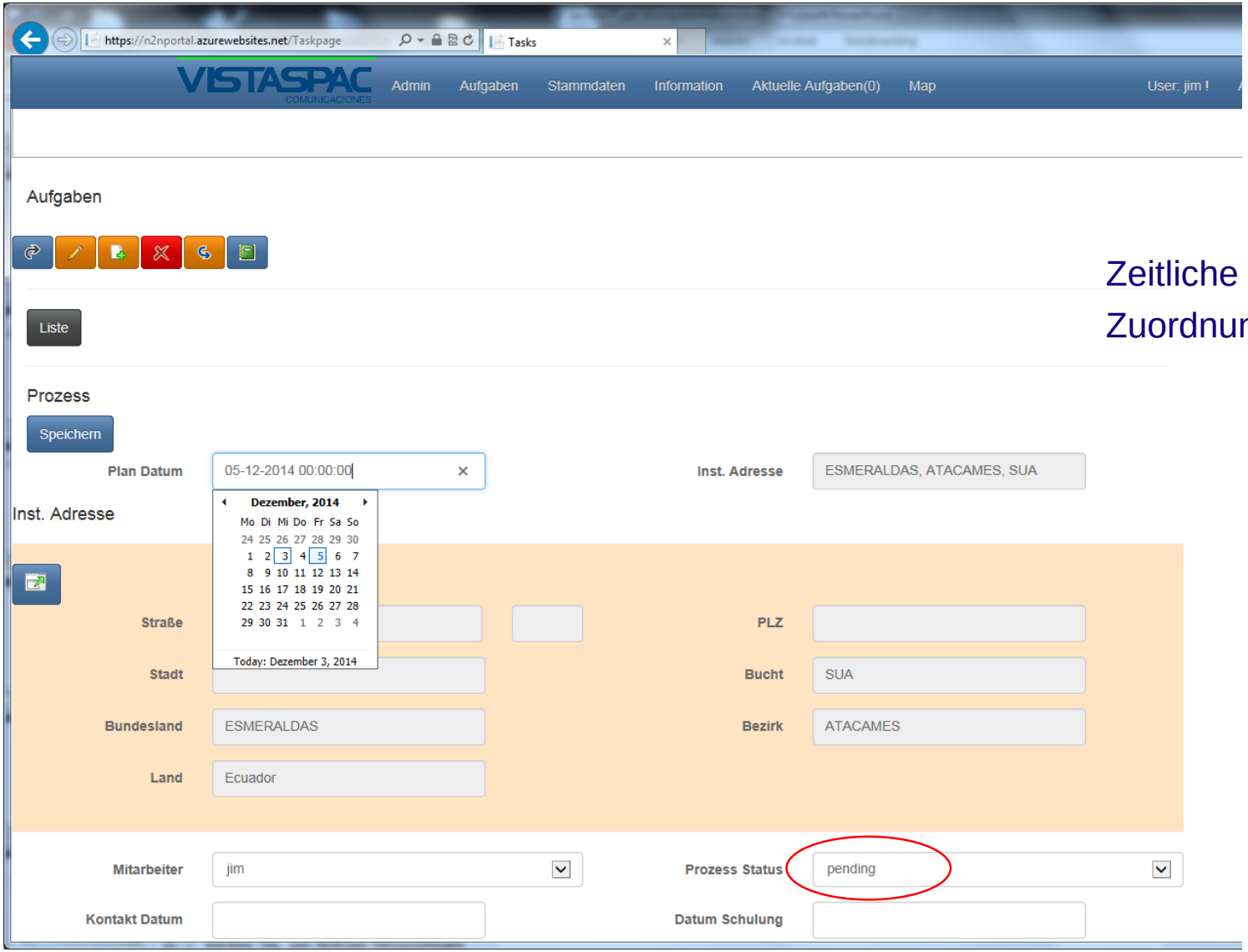
Prozess Ablauf



Alle	ID	Bezeichnung	Bundesland	Bezirk	Bucht	Schiff	Registrierungsnr.	Eigner	Kunden	IMEI/ESN	Plan Datum	Status	Mitarbeiter
					sua								
	381	**Install GPS Device**	**ESMERALDAS**	**ATACAMES**	**SUA**	**EL PODER DE DIOS**	**B-02-06964**	**BAILON LOPEZ FRANCISCO RICARDO**	**COOP. NUEVO PORVENIR**	**0-1252695**		**Completed**	
	382	Install GPS Device	ESMERALDAS	ATACAMES	SUA	ROSA ISABEL II	B-02-03542	CORDOVA CABEZA RICARDO	COOP. NUEVO PORVENIR	0-1265006		Completed	
	383	Install GPS Device	ESMERALDAS	ATACAMES	SUA	HNOS. VERA I	B-02-06203	CORDOVA CABEZA RICARDO	COOP. NUEVO PORVENIR	0-1265487		Completed	
	384	Install GPS Device	ESMERALDAS	ATACAMES	SUA	PEPSICO	B-02-06896	SALDARRIAGA CALDAS MANUEL ISAAC	COOP. NUEVO PORVENIR	0-1254871		Completed	
	385	Install GPS Device	ESMERALDAS	ATACAMES	SUA	LAS 3 HERMANAS	B-02-06756	SANCHEZ SANCHEZ LUCIANO ALFONSO	COOP. NUEVO PORVENIR	0-1265524		Completed	
	386	Install GPS Device	ESMERALDAS	ATACAMES	SUA	SABASCARO	B-02-07743	ANGELOTTI GIUSEPPE	COOP. NUEVO PORVENIR	0-1255486		Completed	
	387	Install GPS Device	ESMERALDAS	ATACAMES	SUA	SABASCARO II	B-02-05966	ANGELOTTI GIUSEPPE	COOP. NUEVO PORVENIR	0-1265728		Completed	
	388	Install GPS Device	ESMERALDAS	ATACAMES	SUA	LOS TRES HERMANO	B-02-06328	BAILON BENITEZ FREDDY FAVIO	COOP. NUEVO PORVENIR	0-1256238		Completed	
	389	Install GPS Device	ESMERALDAS	ATACAMES	SUA	MARIA ISABEL	B-02-07442	BAILON BENITEZ FREDDY FAVIO	COOP. NUEVO PORVENIR	0-1257678		Completed	
	390	Install GPS Device	ESMERALDAS	ATACAMES	SUA	REGALO DE DIOS I	B-02-06690	BAILON VASQUE DENGNI RICARDO	COOP. NUEVO PORVENIR	0-1257668		Completed	

|< << >> >| 33 1329 10

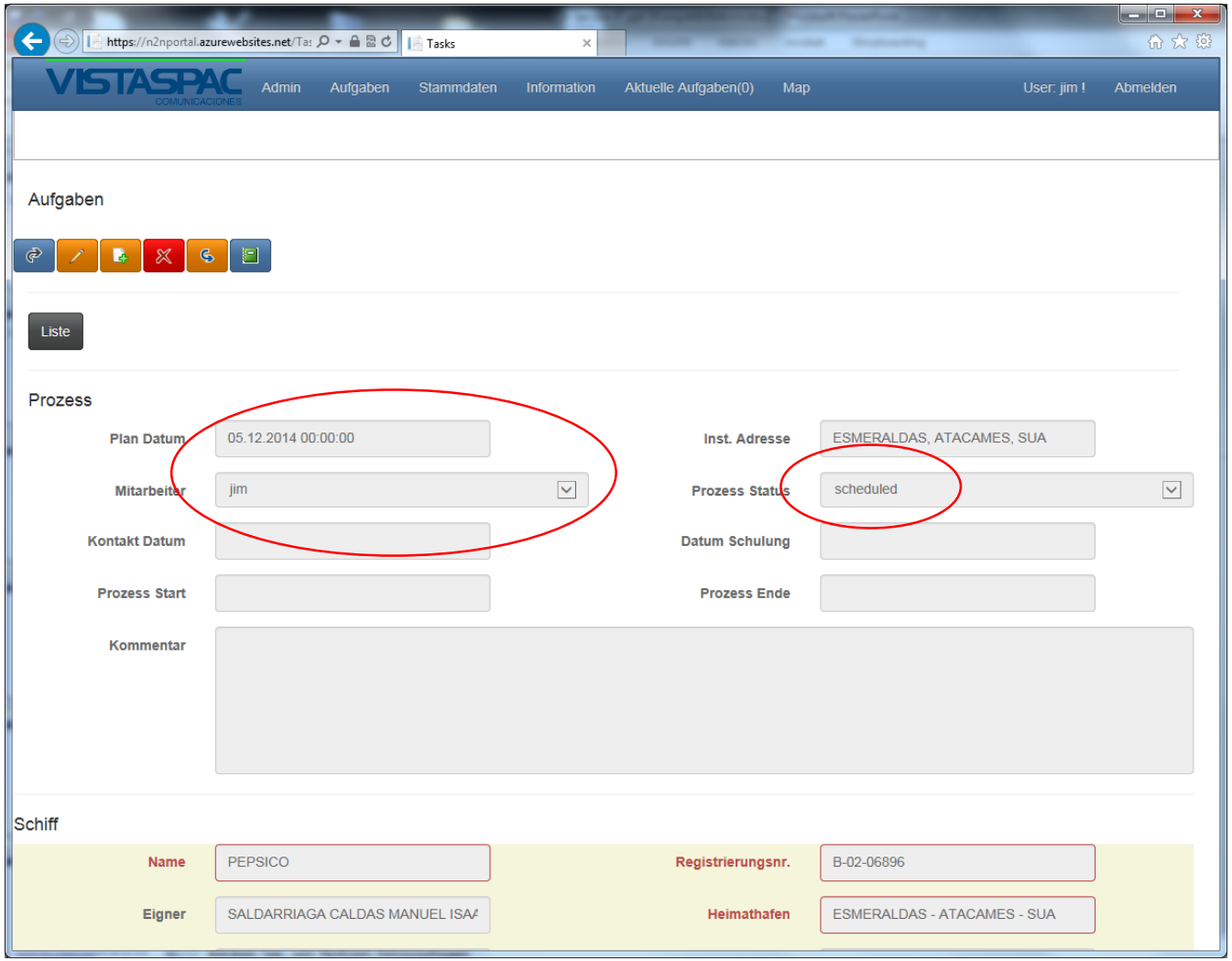
Einplanung von Aufträgen



Zeitliche Planung

Zuordnung Mitarbeiter

Geplanter Auftrag



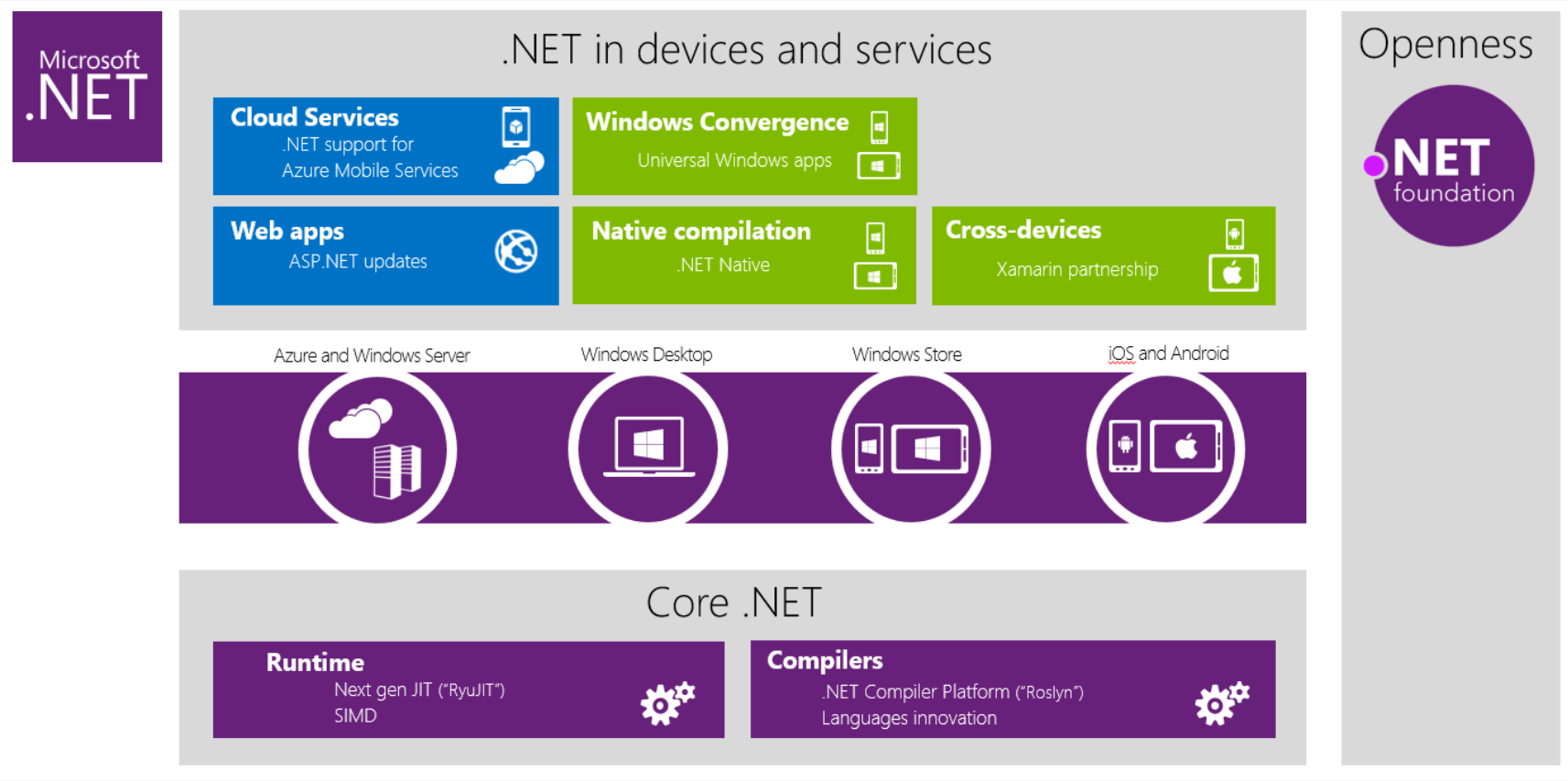
„Field-App“

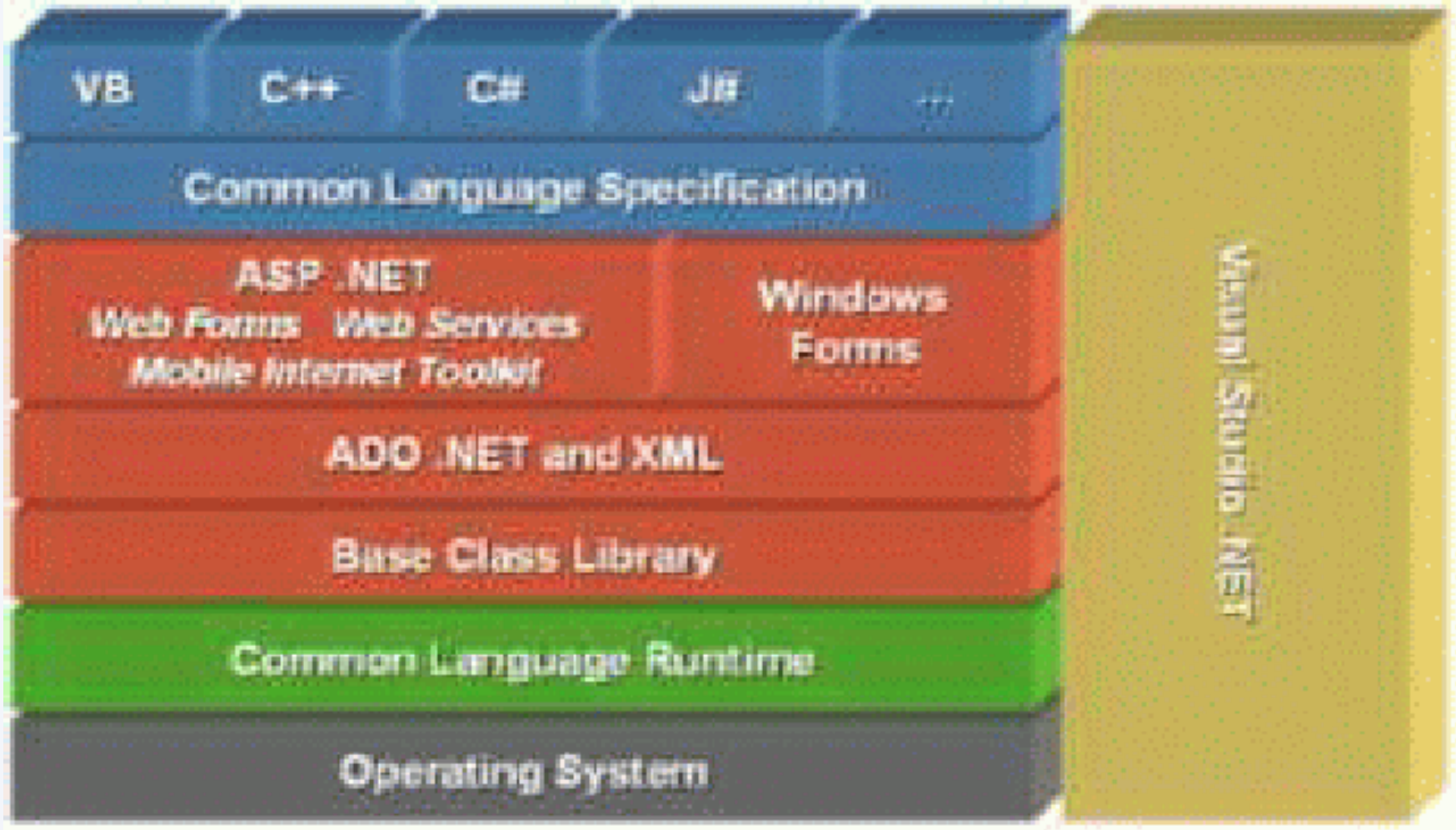
Mitarbeiter vor Ort verfügt über Smartphone

Zugriff über Webportal oder Hybrid App:

- Laden der geplanten Aufträge auf das mobile Gerät (Check Out)
  - Überprüfen / Erfassen der Informationen
  - Bestätigung des Auftrags durch Unterschrift (Touchscreen)
  - Zuordnung des Tracking Geräts durch Scannen des Barcodes
  - Fotos der Besatzungsmitglieder
  - Fotos des Schiffes
- Aktualisieren der abgeschlossenen Aufträge (Check In)

ENDE


Microsoft .NET
.NET in devices and services
Cloud Services
.NET support for Azure Mobile Services
Windows Convergence
Universal Windows apps
Web apps
ASP.NET updates
Native compilation
.NET Native
Cross-devices
Xamarin partnership
Azure and Windows Server
Windows Desktop
Windows Store
iOS and Android
Core .NET
Runtime
Next gen JIT ("RyuJIT")
SIMD
Compilers
.NET Compiler Platform ("Roslyn")
Languages innovation
Openness
.NET foundation


VB
C++
C#
J#
...
Common Language Specification
ASP .NET
Web Forms Web Services
Mobile Internet Toolkit
Windows Forms
ADO .NET and XML
Base Class Library
Common Language Runtime
Operating System
Visual Studio .NET