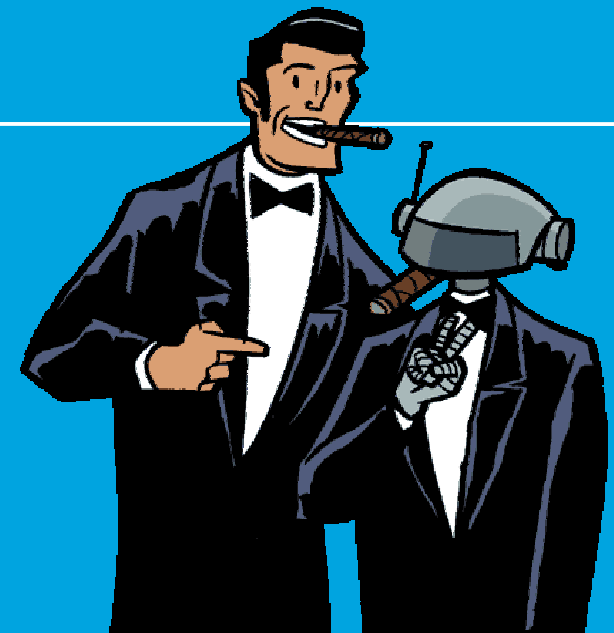


Klassen, Objekte und Assoziation realisiert in C

Thomas Batt
t.batt@microconsult.com



1. Aufgabenstellung

2. Analyse

- Objektanalyse
- Klassenidentifikation

3. Design

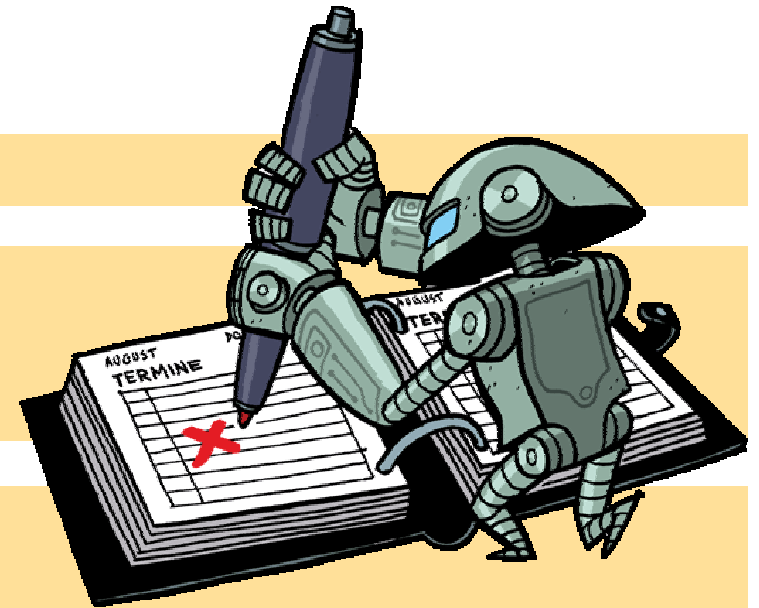
- Klassenmodell
- Instanziierungskonzept

4. Implementierung in C

- Klassen
- Applikation

5. Zusammenfassung

6. Wie geht's weiter?

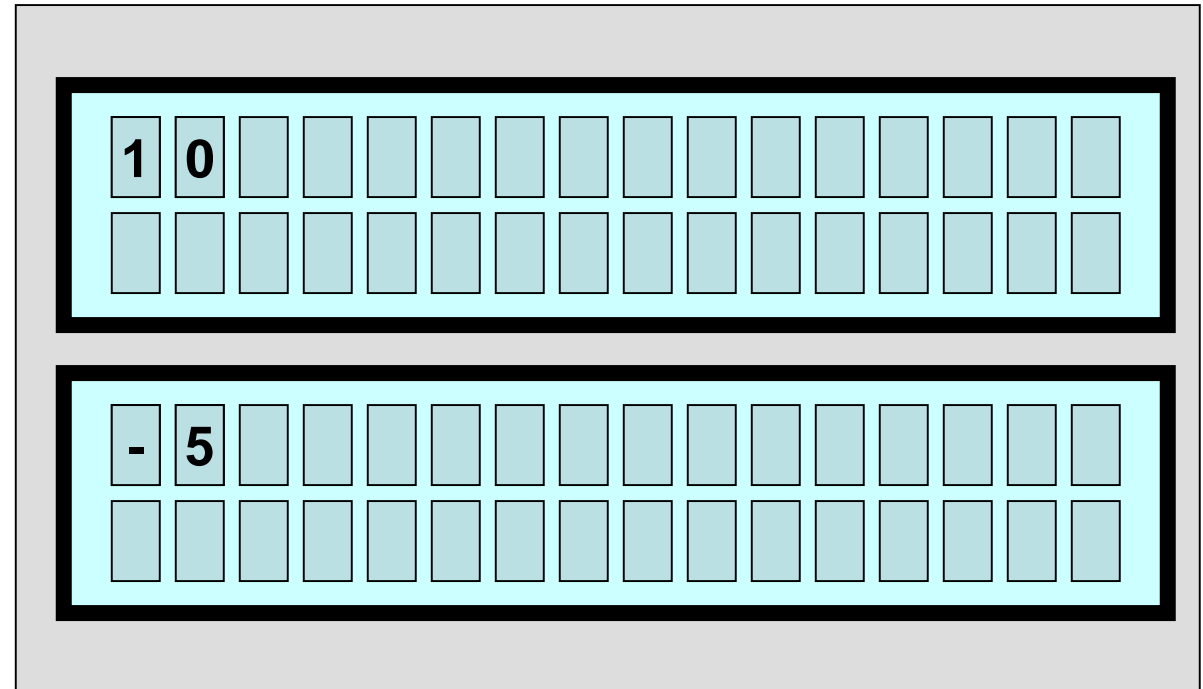


In einem Embedded System

werden auf

zwei verschiedenen Displays

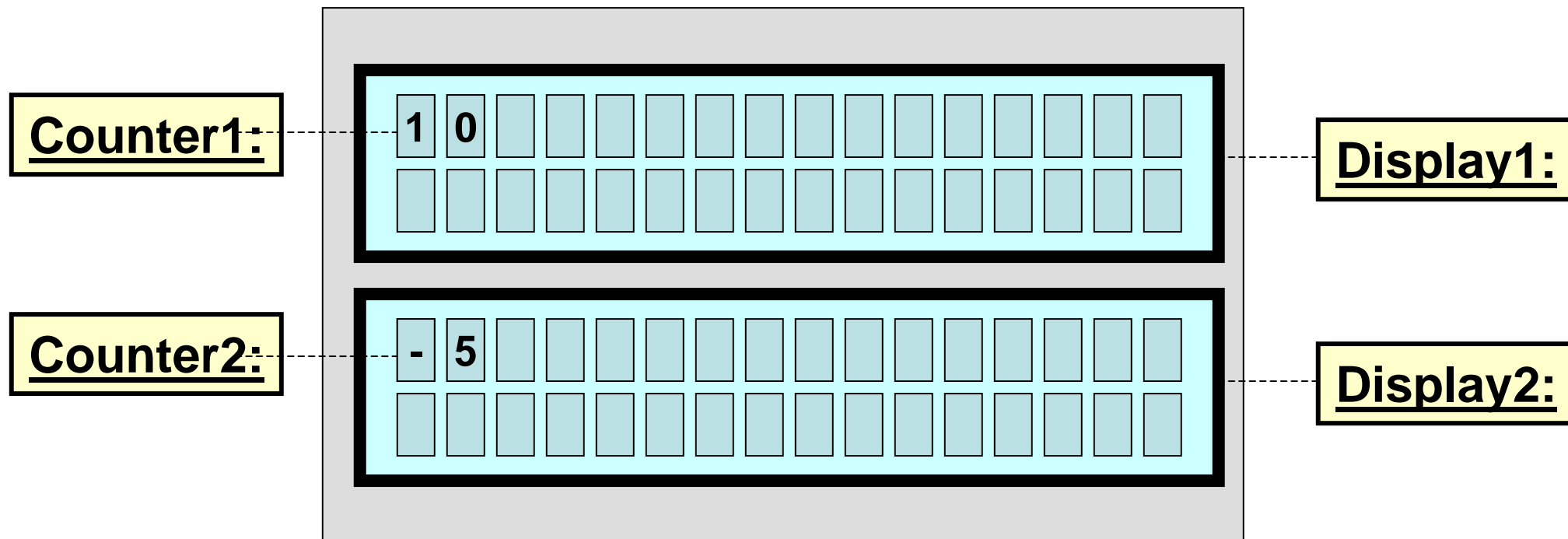
zwei verschieden Zählerwerte
angezeigt.



Jeder Zähler ist in seiner **Zählrichtung**, in seinem **Start-** und **Endwert** konfigurierbar.

Die **Ausgabe** eines jeden Zählwertes erfolgt wahlweise per Software auf **einem der zwei Displays**.

Objektidentifikation in der Realität:



Klassenidentifikation für das Softwaremodell:

Objekte der Realität

Display1:

Display2:

Counter1:

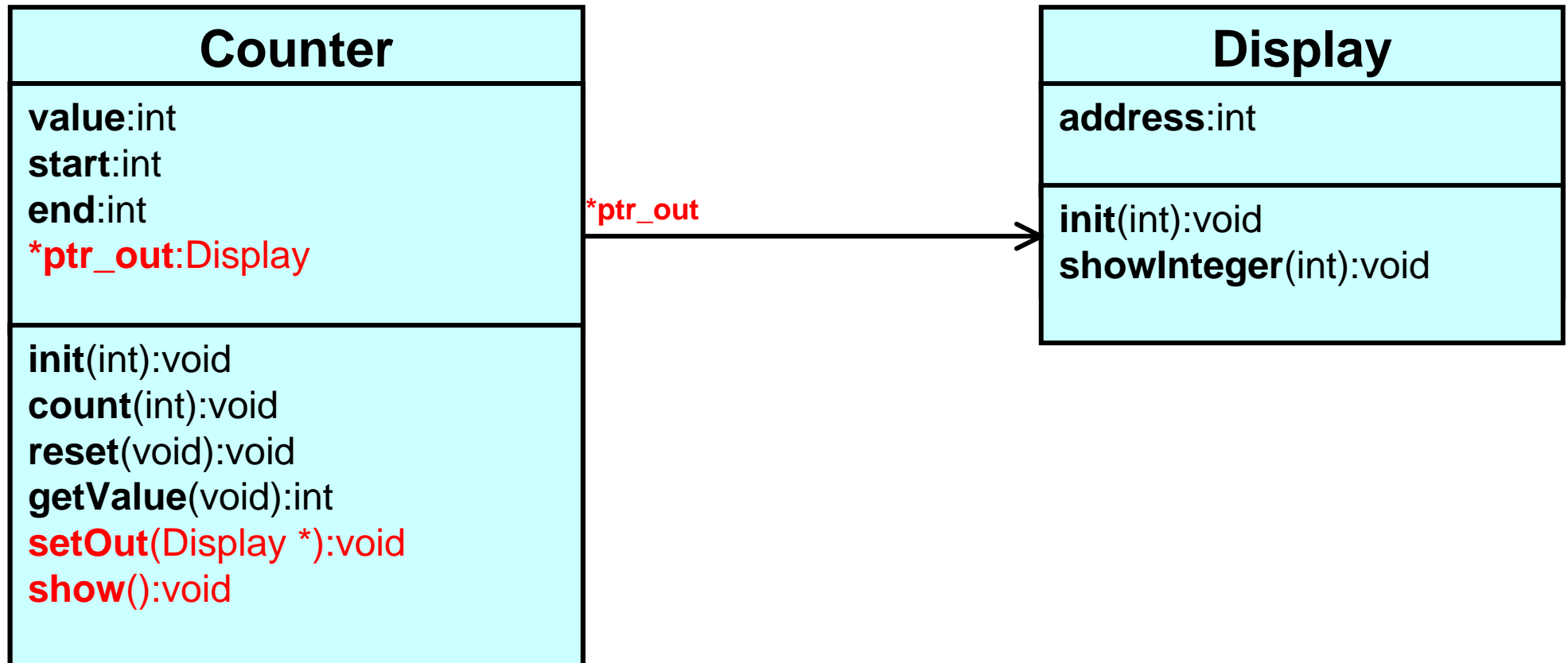
Counter2:



Klassen im Softwaremodell

Display

Counter



————> : gerichtete Assoziation “Nutzbeziehung”

Instanziierungskonzept für die Applikation:

Klassen im Softwaremodell

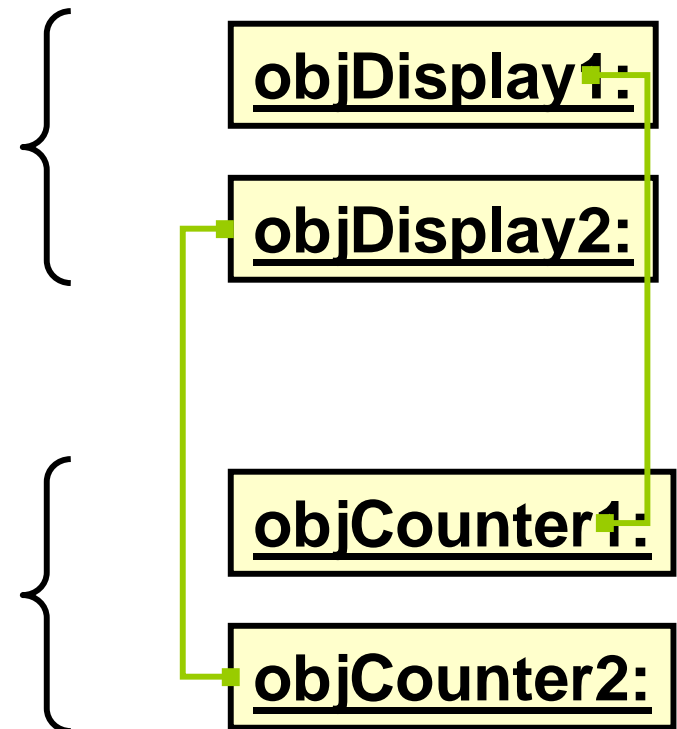
Display



Counter



Objekte in der Applikation



— : **Linkverbindung** in der Objektebene repräsentiert die Assoziation der Klassenebene

Datei: **Display.h**

```
#ifndef DISPLAY_H
#define DISPLAY_H
```

Verhindert
Mehrfacheinbindung

```
struct Display_ {
    int address;
};
```

Alle **Attribute** werden in
Struktur zusammengefasst

```
typedef struct Display_ Display;
```

Die Struktur bekommt
den **Klassennamen**

```
void Display_init (Display *ptrDisplay, int address_par);
void Display_showInteger (Display *ptrDisplay, int value_par);
```

```
#endif
```

Deklaration der
Operationen der Klasse

Parameter zur Objektselektion
bei Aufruf der Operation

Datei: **Display.c**

```
#include "Display.h"  
#include <stdio.h>
```

Einbindung der
Klassendeklaration

```
void Display_init (Display *ptrDisplay, int address_par)  
{  
    ptrDisplay -> address = address_par;  
}
```

Implementierung der
Operationen

```
void Display_showInteger (Display *ptrDisplay, int value_par)  
{  
    printf("Address = %i, Integer Value = %i\n",  
        ptrDisplay -> address, value_par);  
}
```

Zugriff auf Attribute
der Klasse

Datei: Counter.h

```
#ifndef COUNTER_H
#define COUNTER_H

#include "Display.h"
```

Klasseneinbindung
für Assoziation

```
struct Counter_ {
    int value;
    int start;
    int end;
    Display *ptr_out;
};

typedef struct Counter_ Counter;
```

Zeigerdeklaration
für Assoziation

```
void Counter_init      (Counter *ptrCounter, int value_par, int start_par,
                        int end_par);

void Counter_count    (Counter *ptrCounter, int step_par);
void Counter_reset    (Counter *ptrCounter);
int Counter_getValue  (Counter *ptrCounter);
void Counter_setOut   (Counter *ptrCounter, Display *ptrDisplay);
void Counter_show     (Counter *ptrCounter);

#endif
```

Parameter zur Initialisierung der
Assoziation (Display Auswahl)

Datei: Counter.c – 1. Teil

```
#include "Counter.h"
```

```
void Counter_init (Counter *ptrCounter, int value_par, int start_par,
                  int end_par)
{
    ptrCounter -> value    = value_par;
    ptrCounter -> start   = start_par;
    ptrCounter -> end     = end_par;
    ptrCounter -> ptr_out = 0;
}
```

Der Zeiger für die Assoziation wird auf Null gesetzt

```
void Counter_count (Counter *ptrCounter, int step_par)
{
    if (ptrCounter -> value != ptrCounter -> end)
        ptrCounter -> value = ptrCounter -> value + step_par;
    else
        ptrCounter -> value = ptrCounter -> start;
}
```

Zählen vom Startwert bis Endwert, step_par bestimmt die Richtung

Bei Overflow wieder mit dem Startwert beginnen

Datei: Counter.c – 2. Teil

```
void Counter_reset (Counter *ptrCounter)
{
    ptrCounter -> value = 0;
}

int Counter_getValue (Counter *ptrCounter)
{
    return (ptrCounter -> value);
}

void Counter_setOut (Counter *ptrCounter, Display *ptrDisplay)
{
    ptrCounter -> ptr_out = ptrDisplay;
}

void Counter_show (Counter *ptrCounter)
{
    if (ptrCounter -> ptr_out != 0)
        Display_showInteger (ptrCounter -> ptr_out, ptrCounter -> value);
}
```

Initialisierung der
Assoziation (Display-Auswahl)

Nur bei initialisierter
Assoziation erfolgt der Zugriff
auf ein Display-Objekt

Datei: Application.c – 1.Teil

```
#include "Counter.h"
```

```
void main()
{
```

```
    Counter objCounter1;
    Counter objCounter2;
    Display objDisplay1;
    Display objDisplay2;
    int i = 0;
```

```
    Counter_init (&objCounter1, 0, 0, 5);
    Display_init (&objDisplay1, 10000);
    Counter_setOut (&objCounter1, &objDisplay1);
```

```
    Counter_init (&objCounter2, 5, 5, -1);
    Display_init (&objDisplay2, 20000);
    Counter_setOut (&objCounter2, &objDisplay2);
```

Instanziierung von
zwei Counter Objekten

Instanziierung von
zwei Display Objekten

Objekt Initialisierung

Aufbau der Assoziation
zwischen den Objekten
objCounter1 u. objDisplay1

Objekt-Initialisierung

Aufbau der Assoziation
zwischen den Objekten
objCounter2 u. objDisplay2

Objekt-Auswahl für
Operationsaufruf

Datei: **Application.c** – 2. Teil

```
for (i = 0; i <= 7; i ++)  
{  
    Counter_show (&objCounter1);  
    Counter_count (&objCounter1, 1);  
}  
Counter_show (&objCounter1);
```

Zähl- und Ausgabeschleife
(Test des ersten Zählers)

```
for (i = 0; i <= 7; i ++)  
{  
    Counter_show (&objCounter2);  
    Counter_count (&objCounter2, -1);  
}  
Counter_show (&objCounter2);
```

Zähl- und Ausgabeschleife
(Test des zweiten Zählers)

```
}
```

Ausgabe Console

```
CA "C:\DOCUMENTS AND SETTINGS\THOMAS.MICROCONSULT\MY DOCUMENTS\DOCUMENTS\PRESENTATION...
Address = 10000, Integer Value = 0
Address = 10000, Integer Value = 1
Address = 10000, Integer Value = 2
Address = 10000, Integer Value = 3
Address = 10000, Integer Value = 4
Address = 10000, Integer Value = 5
Address = 10000, Integer Value = 0
Address = 10000, Integer Value = 1
Address = 10000, Integer Value = 2
Address = 20000, Integer Value = 5
Address = 20000, Integer Value = 4
Address = 20000, Integer Value = 3
Address = 20000, Integer Value = 2
Address = 20000, Integer Value = 1
Address = 20000, Integer Value = 0
Address = 20000, Integer Value = -1
Address = 20000, Integer Value = 5
Address = 20000, Integer Value = 4
Press any key to continue_
```

Klasse

Attribute

Operationen

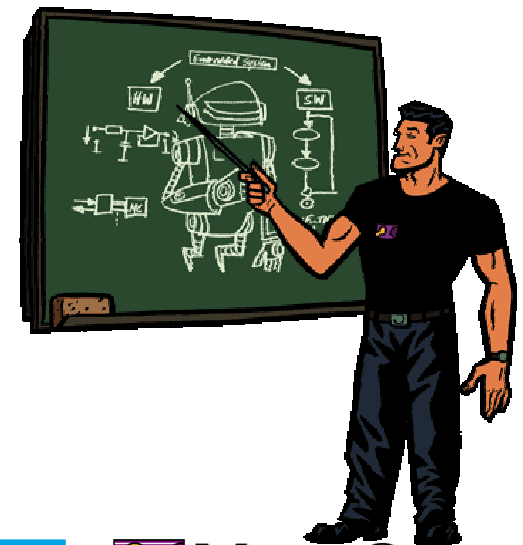


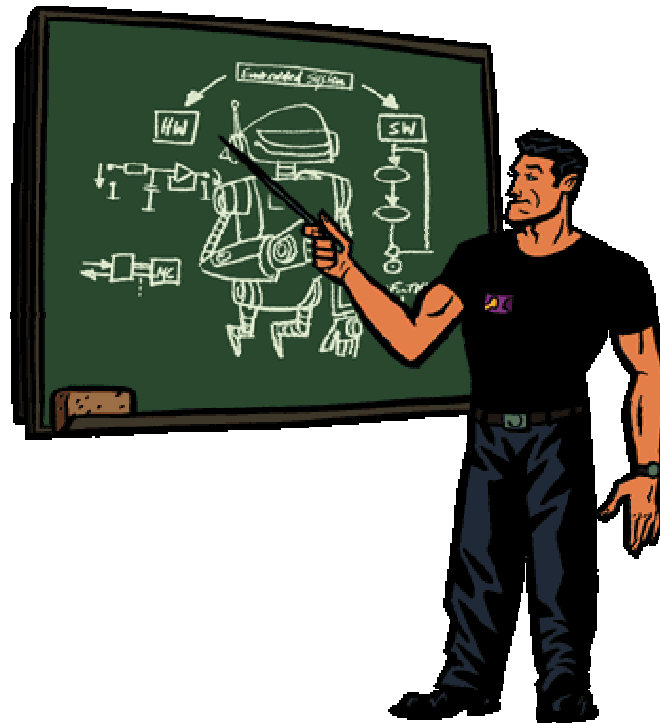
Objekt

- Alle **Attribute** werden in einer **Struktur** zusammengefasst.
- Die **Struktur** bekommt den **Klassennamen**.
- Alle **Operationen** erhalten als **Parameter** einen **Zeiger** vom Typ der Struktur zur **Objektauswahl** beim Aufruf.
- Die gerichtete **Assoziation** wird durch einen **Zeiger** vom Typ der Klasse B in der Struktur vom Typ der Klasse A realisiert.
- Durch das Anlegen einer **Strukturvariablen** erfolgt die Instanziierung, und es wird Speicher für das Objekt angelegt.

Nahezu alle objektorientierten Konstrukte lassen sich mit der Programmiersprache C umsetzen!

Objektbasierende oder objektorientierte Programmiersprachen wie (E)C++ und JAVA vereinfachen die Umsetzung durch entsprechende Mechanismen und Schlüsselworte!



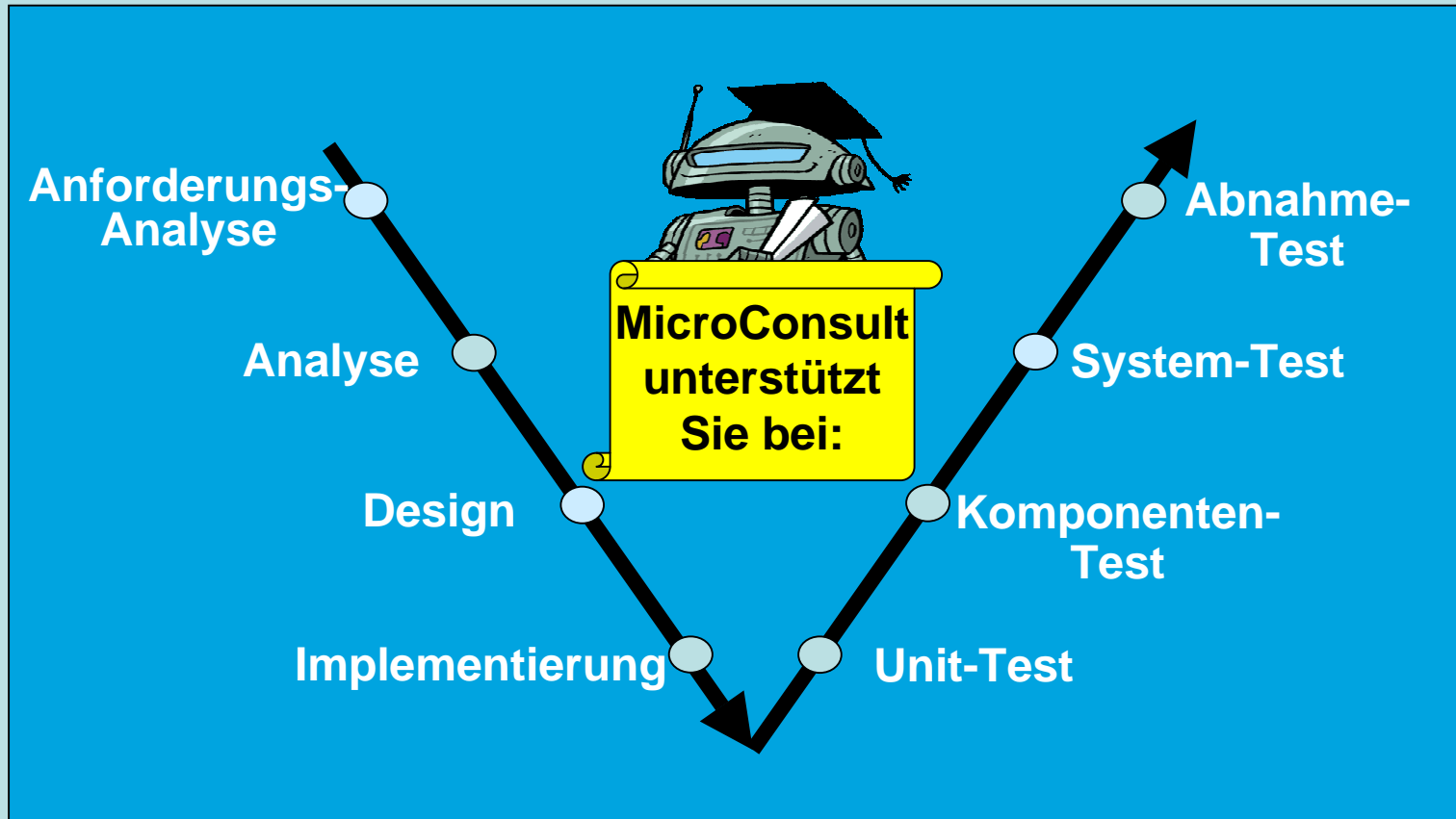


Seminar: **Objektorientierte Programmierung für Mikrocontroller** (Dauer: 5 Tage)

Seminar: **Objektorientierte Analyse für Embedded Systeme** (Dauer: 5 Tage)

Weitere Informationen finden Sie im Internet: <http://www.microconsult.de>

Training, Coaching, Engineering



HW-/SW-Technologien, Tools, Methoden, Prozess, Team