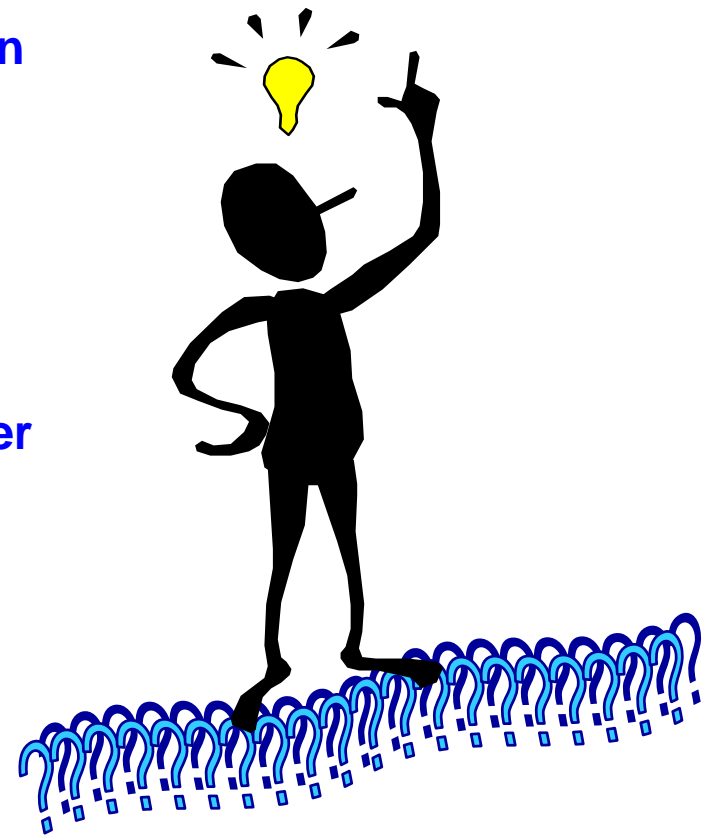


Motivation zur objektorientierten Softwareentwicklung für Embedded Systeme

- ↓ **Objekte stammen aus der Realität oder sind wohldefinierte Begriffe im Umfeld des Problems.**
- ↓ **Objekte sollten die Realität bzw. das Problemumfeld mit allen relevanten Eigenschaften, Funktionen und Beziehungen modellieren.**
- ↓ **Bei Bedarf machen Objekte wiederverwendbare Software-Bausteine (Komponenten) verfügbar.**
- ↓ **Um dies leisten zu können, müssen sie in der Regel leicht an geänderte Anforderungen anpassbar sein.**



Modellierung

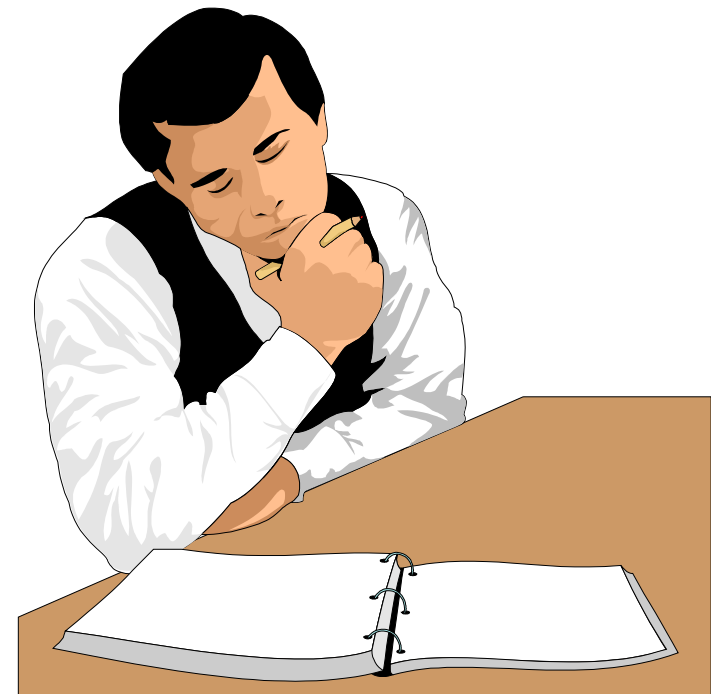
UML (Unified Modeling Language)

Die wichtigsten Diagrammarten:

- Use-Case-Diagramm
- Klassendiagramm
- Paketdiagramm
- Sequenzdiagramm
- Zustandsdiagramm

Vorteile:

- ✓ Direkte Abbildung des Systems
- ✓ Gute Basis zur Codierung
- ✓ Gleichzeitige Dokumentation



Umsetzung:

- Modularisierung auf Grundlage der objektorientierten Analyse
- Objektorientierte Programmierung mit C
- Objektorientierte Programmierung mit C++

Vorteile:

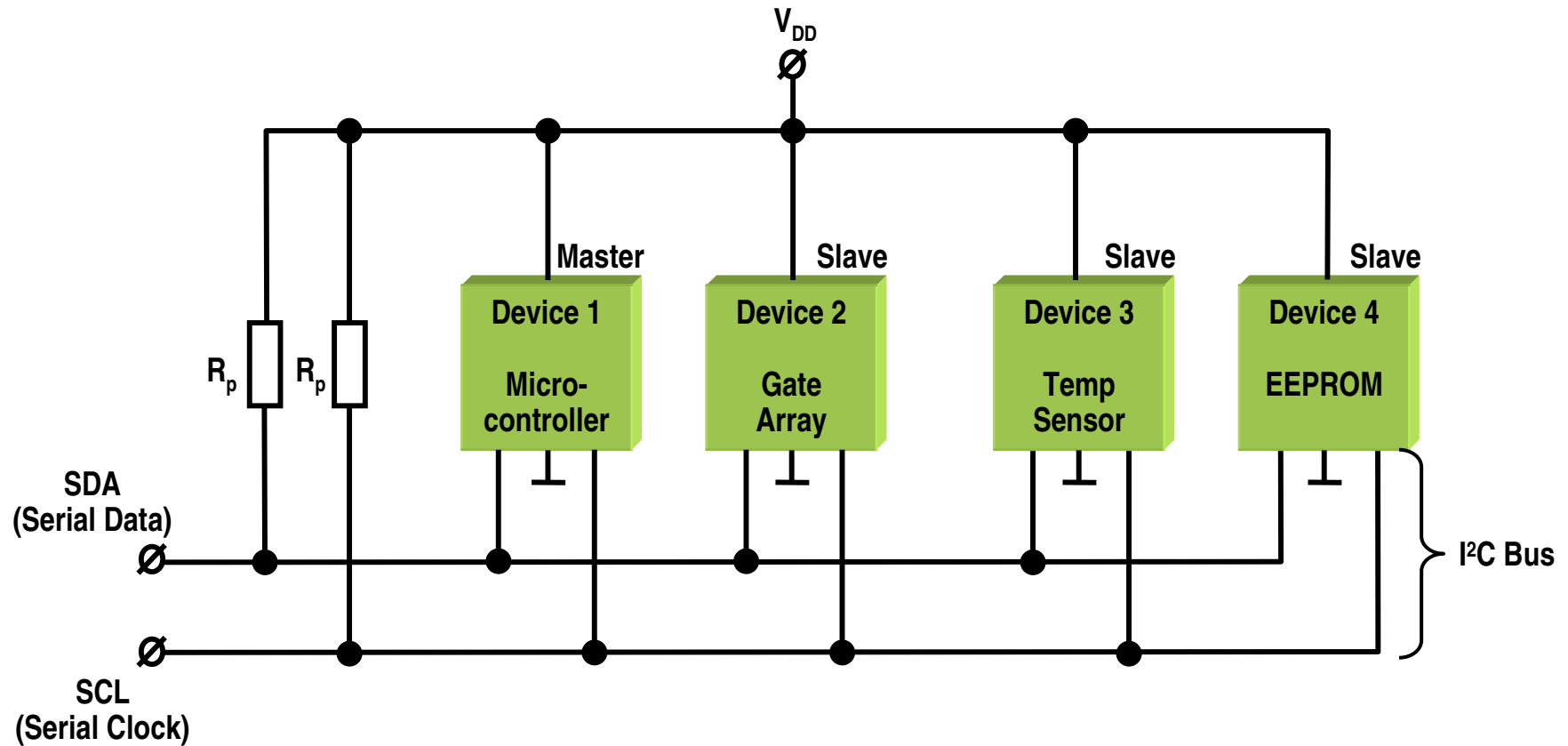
- ✓ Minimierung von globalen Elementen
- ✓ Geschützte Daten (Kapselung)
- ✓ Hoher Strukturierungsgrad im Quellcode

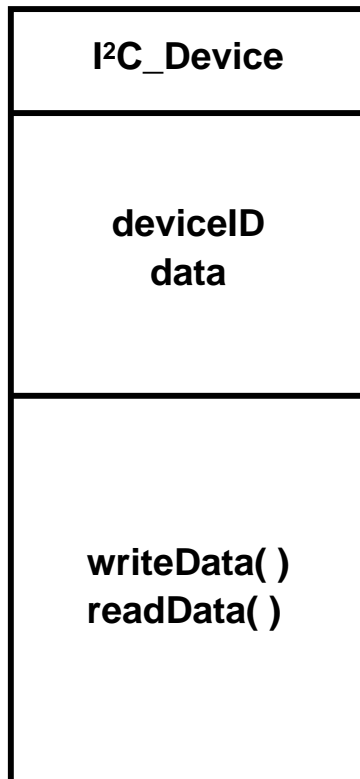


Verbesserung der ...

- Anwendernähe
- Änderungsfreundlichkeit und Erweiterbarkeit
- Möglichkeit zur Wiederverwendung
- Standardisierung
- Produktivität
- Qualität







Klasse: I²C_Device

Attribute:

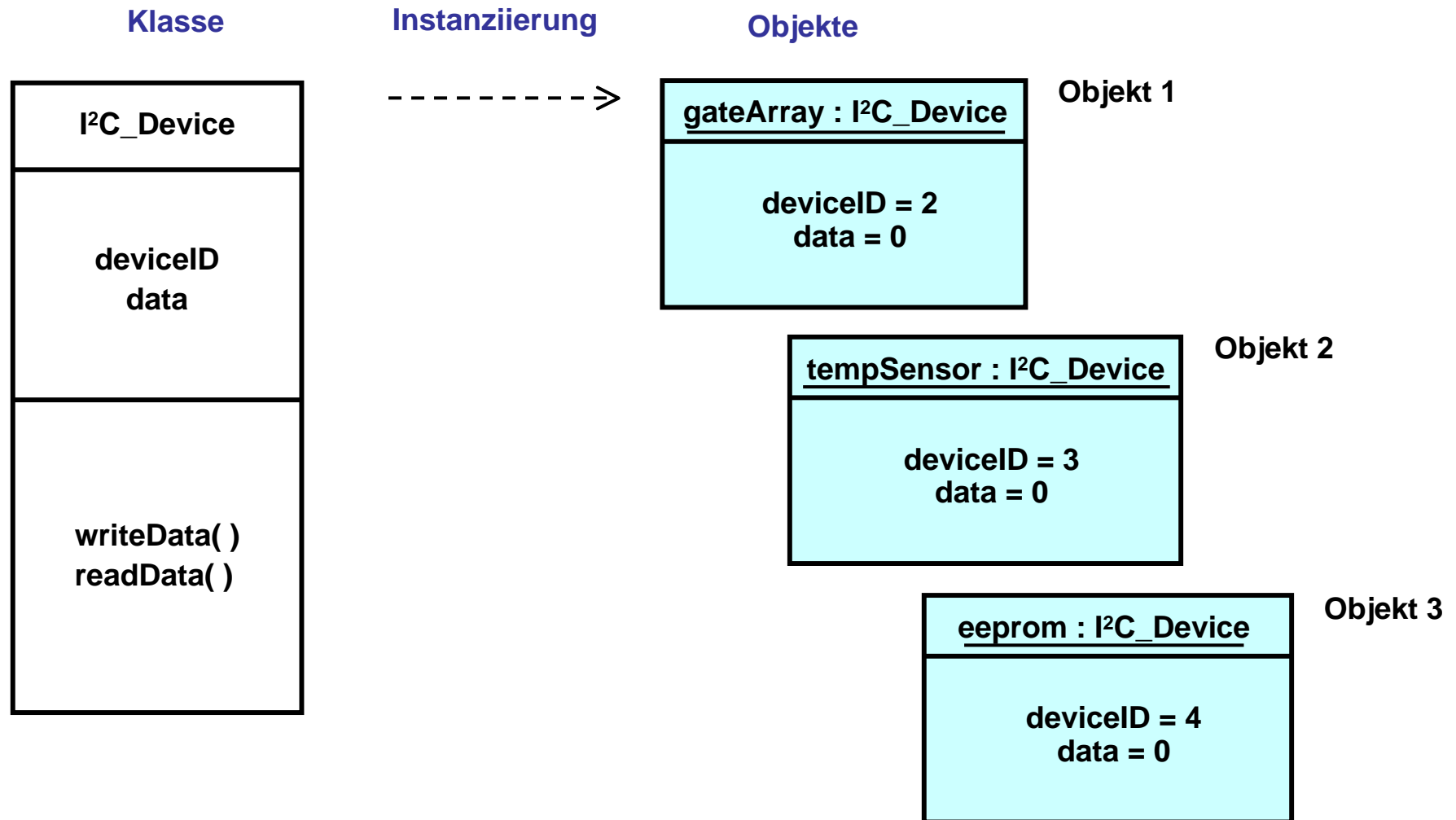
deviceID: Device-Identifikation, repräsentiert Adresse des I²C-Busteilnehmers

data: Daten, die geschrieben bzw. gelesen werden

Operationen:

writeData: Schreibt Daten in das entsprechend adressierte Device

readData: Liest Daten aus dem entsprechend adressierten Device



```
/* Header File */

#ifndef _I2C_Device_H
#define _I2C_Device_H

/*
 * I2C device data structure
 */
struct I2C_Device
{
    unsigned char deviceID;      /* I2C device address */
    unsigned char data;         /* I2C data */
};

/*
 * declare operations
 */
void I2C_Device_init(struct I2C_Device* ptr, unsigned char deviceID_par,
                    unsigned char data_par);

void I2C_Device_readData(struct I2C_Device* ptr);

void I2C_Device_writeData(struct I2C_Device* ptr, unsigned char data_par);

#endif
```

```
/* Implementation File */

#include "I2C_Device.h"

void I2C_Device_init(struct I2C_Device* ptr, unsigned char deviceID_par,
                    unsigned char data_par)
{
    ptr->deviceID    = deviceID_par;
    ptr->data        = data_par;
}

void I2C_Device_readData(struct I2C_Device* ptr)
{
    /* implementation depends on hardware */
}

void I2C_Device_writeData(struct I2C_Device* ptr, unsigned char data_par)
{
    /* implementation depends on hardware */
}
```

```
/* Application File (example only) */

#include "I2C_Device.h"

int main(void)
{
    /* define objects */
    struct I2C_Device gateArray;
    struct I2C_Device tempSensor;
    struct I2C_Device eeeprom;

    /* initialize objects */
    I2C_Device_init(&gateArray, 0x02, 0x00);
    I2C_Device_init(&tempSensor, 0x03, 0x00);
    I2C_Device_init(&eeeprom, 0x04, 0x00);

    /* execute read operations */
    I2C_Device_readData(&gateArray);
    I2C_Device_readData(&tempSensor);
    I2C_Device_readData(&eeeprom);

    /* execute write operations */
    I2C_Device_writeData(&gateArray, 0xAA);
    I2C_Device_writeData(&tempSensor, 0xAA);
    I2C_Device_writeData(&eeeprom, 0xAA);

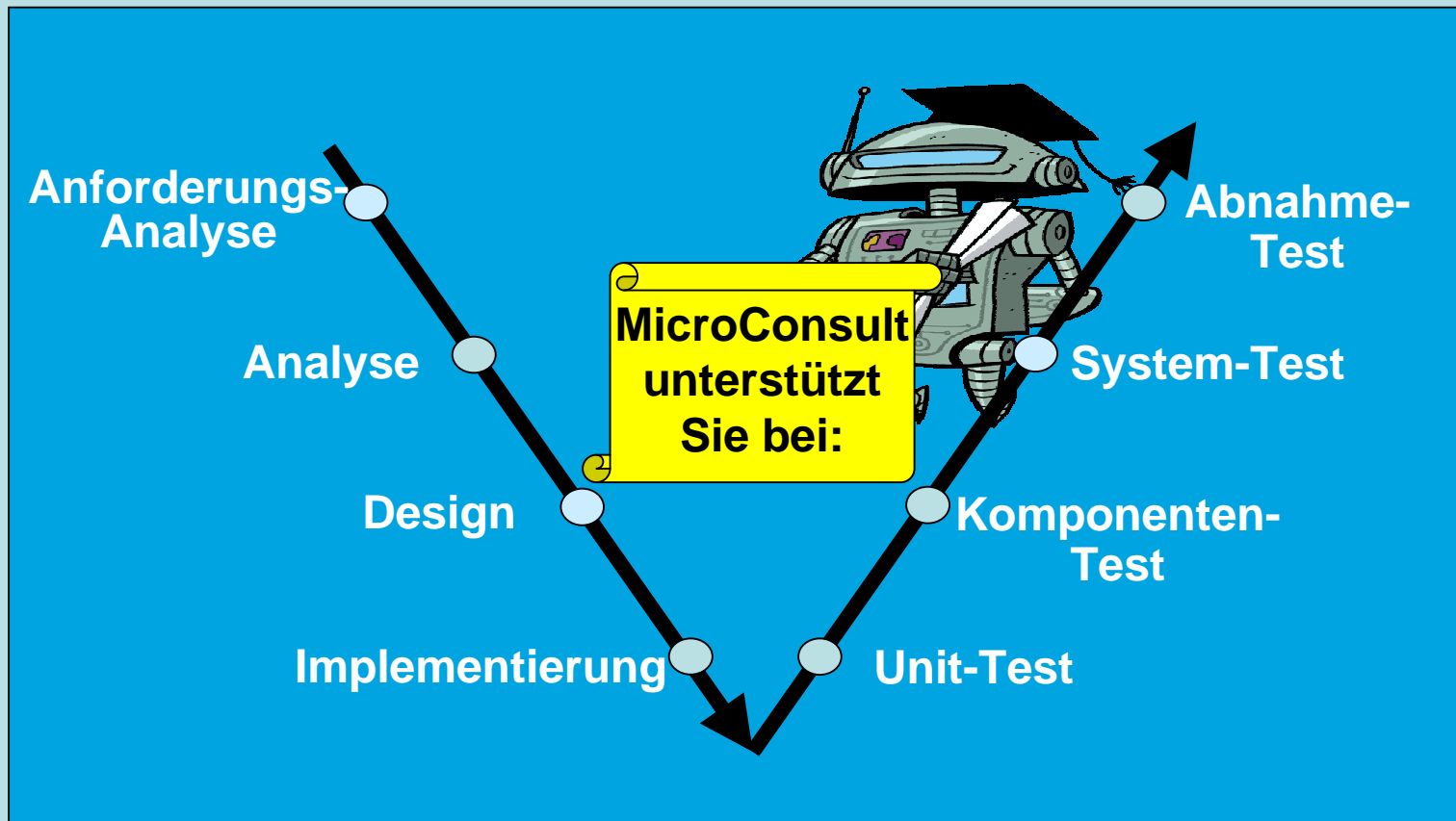
    return 0;
}
```

```
// -----  
// Header File  
// -----  
  
#ifndef _I2C_Device_H  
#define _I2C_Device_H  
  
// I2C device class declaration  
//  
class I2C_Device {  
private:  
    unsigned char deviceID;        // I2C device address  
    unsigned char data;            // I2C data  
  
public:  
    // declare constructor  
    I2C_Device(unsigned char deviceID_par, unsigned char data_par);  
  
    // declare destructor  
    ~I2C_Device();  
  
    // declaration operations  
    void readData();  
    void writeData(unsigned char data_par);  
};  
  
#endif
```

```
// -----  
// Implementation File  
// -----  
  
#include "I2C_Device.h"  
  
// define constructor and destructor  
//  
I2C_Device::I2C_Device(unsigned char DeviceID_par, unsigned char Data_par) {  
    deviceID = deviceID_par;  
    data     = data_par;  
}  
  
I2C_Device::~I2C_Device() {  
    /*empty*/  
}  
  
// define operations  
//  
void I2C_Device::readData() {  
    // implementation depends on the hardware  
}  
  
void I2C_Device::writeData(unsigned char data_par) {  
    // implementation depends on hardware  
}
```

```
// -----  
// Application File (example only)  
// -----  
  
#include "I2C_Device.h"  
  
int main() {  
    // define and initialize objects  
    I2C_Device gateArray(0x02, 0x00);  
    I2C_Device tempSensor(0x03, 0x00);  
    I2C_Device eeprom(0x04, 0x00);  
  
    // execute read operations  
    gateArray.readData();  
    tempSensor.readData();  
    eeprom.readData();  
  
    // execute write operations  
    gateArray.writeData(0xAA);  
    tempSensor.writeData(0xAA);  
    eeprom.writeData(0xAA);  
  
    return 0;  
}
```

Training, Coaching, Engineering



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