Informatik 7bi Schuljahr 2020/2021 als PDF exportieren

Informatik 7. Klasse - Schuljahr 2020/21

Lehrinhalte

- Lehrplaninhalte

Remote-Zugriff auf Schulserver

Kapitel

Leistungsbeurteilung

- **Schularbeiten (SA)**
  - 2x SA (2h) pro Semester
- **Mitarbeit (MA)**
  - Aktive Mitarbeit im Unterricht (aMA)
  - Mündliche Stundenwiederholungen (mMA)
  - Schriftliche Stundenwiederholungen (sMA)
- **Praktische Arbeiten (PA)**
  - 1x praktischer Arbeitsauftrag pro Woche

Stoff für die 2. Schularbeit in Informatik – 7BI – 10.5.2021 1.+2.Stunde

- Kapitel 2.5 - 3.x

Stoff für die 1. Schularbeit in Informatik – 7BI – 17.12.2020 1.+2.Stunde

- Kapitel 1 - 2.4

Förderkurs

- 20.04.2021 13:30 - 15:10
- 27.04.2021 13:30 - 15:10
- 04.05.2021 13:30 - 15:10
- 11.05.2021 13:30 - 15:10
- 18.05.2021 13:30 - 15:10

Was wird in der 7. Klasse gemacht?

5. Semester

Sicherung der Nachhaltigkeit

- Notwendiges Vorwissens für die Kompetenzbereiche dieses Moduls wiederholen und aktivieren
- Grundlagen für die Kompetenzbereiche dieses Moduls ergänzen und bereitstellen

Gesellschaftliche Aspekte der Informationstechnologie

Verantwortung, Datenschutz und Datensicherheit

- Die Bedeutung von Informatik in der Gesellschaft beschreiben, die Auswirkungen auf die Einzelnen und die Gesellschaft einschätzen und Vor- und Nachteile an konkreten Beispielen abwägen können.
- Maßnahmen und rechtliche Grundlagen im Zusammenhang mit Datensicherheit, Datenschutz und Urheberrecht kennen und anwenden können.

Informatiksysteme – Hardware, Betriebssysteme und Vernetzung

Technische Grundlagen und Funktionsweisen (Hardware)

- Aktuelle Entwicklungen auf dem Hardwaresektor kennen
- Mit Hardware-Komponenten praktisch umgehen können

Virtualisierung

- Virtualisierungs-Software für Betriebssysteme und Server einsetzen können
- Betriebssysteme und Software virtualisieren können

Betriebssysteme (Linux)

- Die Arbeitsumgebung (Oberflächen, Tools, Script-Programmierung) von Linux einrichten können
- Anwenderprogramme unter Linux installieren und verwenden können
- Die Bedeutung von Linux als Betriebssystem für Internetdienst kennen
- Im LAN und im Internet mit Linux arbeiten können

Netzwerke

- Grundlagen der Vernetzung von Computern beschreiben und lokale und globale Computernetzwerke nutzen können.
- Netz-Topologien kennen
- Den Datenverkehr in Netzen simulieren und analysieren können
- Netzhardware kennen und praktisch damit umgehen können
- Protokolle und Adressierung kennen und anwenden können
- Sicherheitskonzepte für Netzwerke kennen
- Netztypen (Peer-to-Peer, Server-Client) kennen
- Peer-to-Peer und Server-Client-Netz installieren können
- Einfache Netzadministration durchführen können
- Programme für die Verwendung im Netz installieren können

Angewandte Informatik, Datenbanksysteme und Internet

Suche, Auswahl und Organisation von Information

- Informationsquellen erschließen, Inhalte systematisieren, strukturieren, bewerten, verarbeiten und unterschiedliche Informationsdarstellungen verwenden können.

Datenmodelle und Datenbanksysteme

- Datenbanken durch ein Programm ansteuern können
- Datenbanken benutzen und einfache Datenmodelle entwerfen können.
- Die Charakteristika von SQL kennen
- PHP-Scripts zur Anbindung einer Webseite an eine MySQL-Datenbank erstellen können

6. Semester

Sicherung der Nachhaltigkeit

- Notwendiges Vorwissens für die Kompetenzbereiche dieses Moduls wiederholen und aktivieren
- Grundlagen für die Kompetenzbereiche dieses Moduls ergänzen und bereitstellen

Angewandte Informatik, Datenbanksysteme und Internet

Web-Techniken

- PHP-Scripts zur Anbindung einer Webseite an eine MySQL-Datenbank erstellen können
- MySQL-Datenbank mittels PHP verwalten können

Gesellschaftliche Aspekte der Informationstechnologie

Verantwortung, Datenschutz und Datensicherheit

- Kommunikation mit einem externen Server herstellen können
- Die Bedeutung der Absicherung extern verwalteter Daten einschätzen können

Algorithmik und Programmierung

Algorithmen und Datenstrukturen

- Algorithmen erklären, entwerfen, darstellen können.
- Klassen und Objekte kennen und verwenden können
- Files kennen und einsetzen können
- Datenstrukturen Zeiger und Listen kennen und visualisieren können
- Algorithmen mit Zeigern und Listen erstellen können

Programmierung (Objektorientierte visuelle Programmiersprache)

- Weitere Komponenten der visuellen Programmierung kennen und anwenden können
- Mit einer visuellen Programmiersprache auf fortgeschrittenem Niveau arbeiten können
- Mit Files arbeiten können
- Graphikelemente einsetzen können
- Algorithmen in der visuellen Programmiersprache implementieren können
- Algorithmen mit Zeigern und Listen programmieren können

```
7. Klasse (3 Stunden,  je eine 2h Schularbeit pro Semester)
```

C++

- C++ Skriptum

Arrays

Bisher haben wir alle Variablen einzeln definiert. Das kann aber ziemlich aufwendig werden, wenn man eine größere Anzahl von Variablen benötigt:

```
int x1,x2,x3,x4,x5; /* Die Definition von 1000 Variablen
wäre sinnvoll nicht mehr zu verwalten.*/
```

Auch die Arbeit mit diesen Variablen ist recht umständlich: Da jede nur unter ihrem Namen angesprochen werden kann, ist es nicht möglich, sie alle in einer Schleife zu durchlaufen. Diese Nachteile lassen sich vermeiden, wenn die Variablen nicht einzeln, sondern gemeinsam als **Array** definiert werden. Ein Array ist eine Zusammenfassung von Variablen desselben Datentyps unter einem einzigen Namen. Die einzelnen Variablen können über den Namen des Arrays und einen **Index** angesprochen werden und werden auch als Elemente des Arrays bezeichnet. Der Index kann ein Ausdruck und damit insbesondere eine Variable sein.

Eindimensionale Arrays

Grundlagen

Ein Feld ist ein strukturierter Datentyp mit einer bestimmten Anzahl von Elementen, die alle vom selben Typ sein müssen. In der C++-Literatur werden Felder auch als Vektoren bezeichnet. Mit der Typvereinbarung werden eigene Typnamen für Felder vergeben. Diese neuen Namen können wie Standard-typenbezeichner verwendet werden.

Beispiele:

```
#define max 7
typedef float Vektor[4];
typedef char Zeile[80];
typedef int Besetzt[max];
typedef int Zaehler[26];
```

Die Vereinbarung eines Feldtyps enthält die **Anzahl** der Elemente und den **Elementtyp**.

Die Anzahl muss eine **Konstante** sein und wird in eckige Klammern [] eingeschlossen.

Wird ein Feldtyp beispielsweise mit `float Vektor[4];` vereinbart, bedeutet dies, dass dieses Feld 4 Elemente (jedes vom Typ `float`) hat.

Mit diesen Feldtypen können (wie mit jedem anderen Typ) Variablen vereinbart werden.

```
Vektor v;
Zeile satz;
Besetzt besetzt;
```

```
Zaehler anz;
```

Auf ein einzelnes Feldelement wird über den Namen der Variablen, die mit einem Index versehen wird, zugegriffen. Der Index kann jeder beliebige int-Ausdruck sein. Der Wert des Indexausdrucks muss innerhalb der Indexgrenzen liegen. Die Untergrenze ist immer 0, die Obergrenze ist gleich der Anzahl minus 1.

Der Index wird in eckige Klammern eingeschlossen.

Der Index wird nicht überprüft.

Beispiele:

```
v[0]=25.2;
v[l]=37.1;
v[2]=66.3;
v[3]=97.7;

ch=satz[i*2];

if (zimmer[i])  ...;
anz[ch-'A']=anz[ch-'A']+l;
/* besser: */
anz[ch-'A']++;
```

Da Indizes nicht überprüft werden, kann beispielsweise die folgende Zuweisung vom Compiler nicht als fehlerhaft erkannt werden:

```
v[4]=99.0;
```

Wird diese Anweisung ausgeführt, kann die Reaktion des Programms nicht vorhergesagt werden. Da die Variablenspeicherplätze nacheinander vergeben werden, *kann eine andere Variable verändert werden* - ein Fehler, der nur sehr schwer zu finden ist. Oft stürzt auch das Programm ab! Dieser Fehler kann auch bei zu klein dimensionierten Zeichenketten auftreten.

Ein expliziter Name für einen Feldtyp darf auch fehlen, das heißt, der Feldtyp kann direkt in der Variablenvereinbarung stehen.

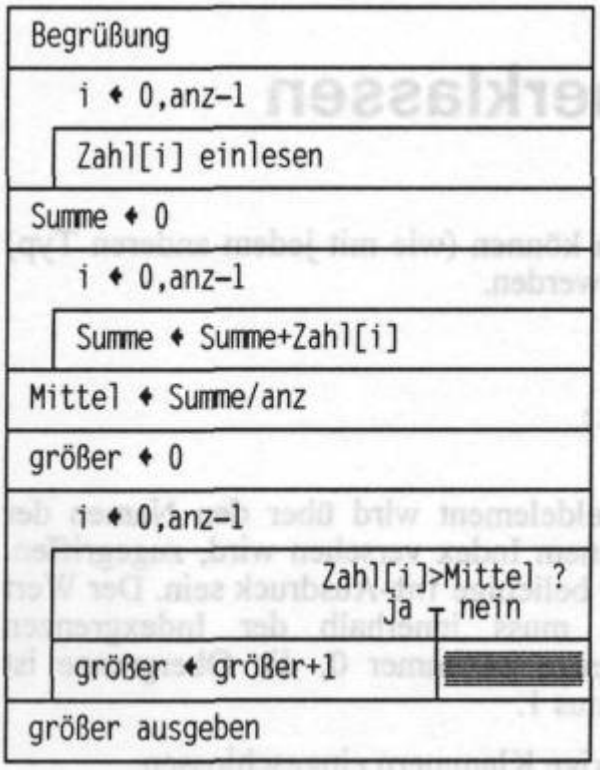
Beispiele (diese Vereinbarungen entsprechen den schon verwendeten Vereinbarungen):

```
#define max 7
int main()
{float v[4];
 char satz[80];
 int zimmer[max];
 int anz[26];
}
```

Die erste Schreibweise ist aber bei längeren Programmen übersichtlicher und daher zu bevorzugen.

Beispiel 1

10 Zahlen sollen eingelesen werden. Es ist das arithmetische Mittel zu berechnen. Die Anzahl der Zahlen, die größer als das Mittel sind, ist auszugeben.



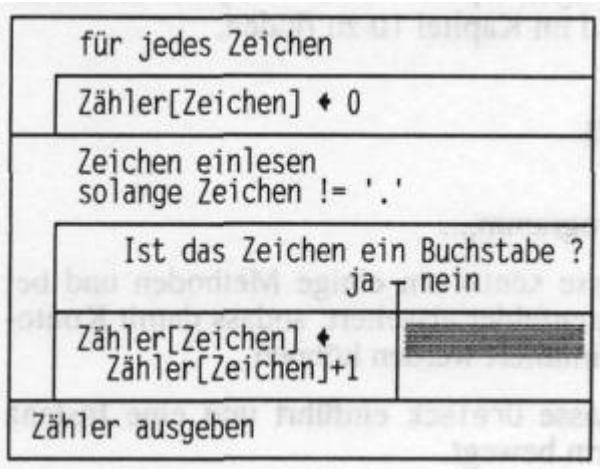
CPP Code

```
/* Berechnet die Anzahl der Zahlen, die
   größer als das arithmetische Mittel
   der Zahlenfolge sind */

#include <iostream.h>

const int anz=10;

int main ()
{ float zahl[anz],summe=0,mittel;
  int i,groesser=0;
  cout << "Bestimmt die Anzahl der Zahlen > arithm. Mittel\n";
  cout << anz << " Zahlen:\n";

  for (i=0;i<anz;++i)
  { cout << i+1 << ". Zahl ? ";
    cin >> zahl[i];
  }
  for (i=0;i<anz;++i)
    summe+=zahl[i];
  mittel=summe/anz;
```

```
  for (i=0;i<anz;++i)
    if (zahl[i]>mittel) ++groesser;
  cout << groesser << " Zahlen sind größer als das arithmetische Mittel(" <<
mittel << ")\n";

  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Beispiel 2

Ein beliebiger Text soll eingelesen und die absolute und relative Häufigkeit der einzelnen Großbuchstaben in Tabellenform ausgegeben werden. Der Text wird durch einen Punkt abgeschlossen.



CPP Code

```
/* Absolute und relative Häufigkeit der Buchstaben
   in einem beliebigen Text bestimmen */

#include <iostream.h>
#include <conio.h>
#include <ctype.h>
using namespace std;

int main ()
{ char i,j,zeichen;
  int anz=0,zaehler[26];
  for (i='A';i<='Z';++i) zaehler[i-'A']=0;
  printf("Häufigkeit von Großbuchstaben.\n"
     "Text ? ('.' = Abbruch)\n");

  while ((zeichen=getche())!='.')
  { ++anz;
    if (isupper(zeichen))
      ++zaehler[zeichen-'A'];
```

```
  }

  if (anz==0) anz=1;
  printf
    ("\nBuchstabe   Abs.H.     Rel.H.     "
    "Buchstabe   Abs.H.     Rel.H.\n"
    "============================     "
    "============================\n");
  for (i='A';i<='M';++i)
  { if (anz==0) anz++;
    printf("%4c: %10d %10.1f%%      ",
      i,zaehler[i-'A'],
      zaehler[i-'A']*100.0/anz);
    j=i+13;
    printf("%4c: %10d %10.1f%%\n",
      j,zaehler[j-'A'],
      zaehler[j-'A']*100.0/anz);
  }
  system("PAUSE");
  return EXIT_SUCCESS;
```

Übergabe von eindimensionalen Feldern an Unterprogramme

Die Übergabe von Feldern an Unterprogramme erfolgt

- über Typdefinition (`typdef`) oder
- über Angabe der Adresse auf das betreffende Feld oder
- über allgemeine Typangabe.

Programmbeispiel 1 sieht mit Unterprogrammen so aus:

CPP Code

```
/* Berechnet die Anzahl der Zahlen, die
   größer als das arithmetische Mittel
   der Zahlenfolge sind */

#include <iostream.h>

const int anz=10;
typedef float Feld[anz]; // nur bei erster Variante notwendig

void lesen(Feld z)
{ int i;
  for (i=0;i<anz;++i)
  { cout << i+1 << ". Zahl ? ";
    cin >> z[i];
```

```
  }
}

float mittel(Feld z)
{ int i;
  float summe=0;
  for (i=0;i<anz;++i)
    summe+=z[i];
  return summe/anz;
}

int zaehlen(Feld z,float m)
{ int i,groesser=0;
  for (i=0;i<anz;++i)
    if (z[i]>m) ++groesser;
  return groesser;
}


/* Alternative Variante der Parametervereinbarung: mit Hilfe von Zeigern
void lesen(float *z)
{ int i;
  for (i=0;i<anz;++i)
  { cout << i+1 << ". Zahl ? ";
    cin >> *(z+i);
  }
}

float mittel(float *z)
{ int i;
  float summe=0;
  for (i=0;i<anz;++i)
    summe+=*(z+i);
  return summe/anz;
}

int zaehlen(float *z,float m)
{ int i,groesser=0;
  for (i=0;i<anz;++i)
    if (*(z+i)>m) ++groesser;
  return groesser;
}
*/


/* Alternative Variante der Parametervereinbarung: Angabe, dass z ein
eindimensionales Array von float-Werten ist
void lesen(float z[])
{ int i;
  for (i=0;i<anz;++i)
  { cout << i+1 << ". Zahl ? ";
```

```
    cin >> z[i];
  }
}

float mittel(float z[])
{ int i;
  float summe=0;
  for (i=0;i<anz;++i)
    summe+=z[i];
  return summe/anz;
}

int zaehlen(float z[],float m)
{ int i,groesser=0;
  for (i=0;i<anz;++i)
    if (z[i]>m) ++groesser;
  return groesser;
}
*/

int main ()
{ float zahl[anz];
  float m;
  cout << "Bestimmt die Anzahl der Zahlen > arithm. Mittel\n";
  cout << anz << " Zahlen:\n";

  lesen(zahl);
  m=mittel(zahl);
  cout << zaehlen(zahl,m) << " Zahlen sind groesser als das arithmetische
Mittel(" << m << ")\n";

  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Grundaufgaben

Erstelle unter Verwendung eines eindimensionalen Arrays ein Programm, dass

- (a) ein Array mit 20 Zufallszahlen im Bereich von 1 - 100 belegt und dieses ausgibt;
  Unterprogramme: `erstelle`, `ausgabe`
- (b) die Summe aller Zahlen des Arrays ausgibt;
  Unterprogramm: `summe`
- (c) das arithmetische Mittel dieses Zufallsarrays berechnet;
- (d) nach Eingabe eines Index angibt, wie die zugehörige Zahl lautet;
  Unterprogramm: `index2zahl`
- (e) nach Eingabe einer Zahl angibt, ob diese im Array vorkommt oder nicht;
  Unterprogramm: `suche`
- (f) nach Eingabe einer Zahl angibt, an welcher Stelle diese Zahl vorkommt (wenn sie im Array

enthalten ist);
Unterprogramm:`zahl2index`

- (g) dieses Zufallsarray verkehrt ausgibt;
  Unterprogramm: `invertiere`
- (h) das nach Eingabe zweier Positionen, die Werte an diesen Positionen tauscht;
  Unterprogramm: `tausche`
- (i) das Minimum, das Maximum und die Spannweite dieses Arrays ausgibt;
  Unterprogramm: `minmax`
- (j) das ein zweites Array anlegt, in dem die Werte in steigender Reihenfolge eingetragen werden;

Eindimensionale Arrays und Funktionen

Beispiel: Ermittlung von n Quicktipps im Lotto 6 aus 45 Wir beginnen mit einer Version ohne und Funktionen und bauen diese dann entsprechend um!

```
void main()
{
  int n;  //Anzahl der Tipps
  int tipp[6];
  srand(time(0));
  cout<<"Wie viele Tipps moechten Sie erhalten?";
  cin>>n;
  for(int i=1;i<=n;i++){     //n Tipps ermitteln
    cout<<"\n"<<i<<". Tipp: ";

    for(int j=0; j<6;j++){
        tipp[j]=rand()%45+1;
        cout<<tipp[j]<<" ";
    }
  }
  getch();
}
```

Bei der Verwendung von Arrays in Funktionen sind einige Besonderheiten zu beachten:

- Eine Funktion kann keinen Array-Typ besitzen.
- Werden Arrays als Parameter verwendet, so sind dies in C++ automatisch Ein-Ausgabe-Parameter. Dh. es wird im Unterprogramm keine Kopie der Array-Variablen angelegt, sondern das Unterprogramm greift direkt auf die Array-Variable der aufrufenden Funktion zu.

Wir erweitern nun das obige Beispiel um den Einsatz von Funktionen:

```
void erzeugeQuicktipp(int a[6]);

void main()
{
  int n;  //Anzahl der Tipps
  int tipp[6];
  srand(time(0));
  cout<<"Wie viele Tipps moechten Sie erhalten?";
  cin>>n;
  for(int i=1;i<=n;i++){     //n Tipps ermitteln
    cout<<"\n"<<i<<". Tipp: ";
    erzeugeQuicktipp(tipp);
  }
  getch();
}

void erzeugeQuicktipp(int a[6]){
  for (int i=0; i<6; i++) {
```

```
      a[i]=rand()%45+1;
      cout<<a[i]<<" ";
  }
}
```

Erweitere das Beispiel nun um eine eingene Funktion für die Ausgabe. Außerdem soll verhindert werden, dass innerhalb eines Tipps Zahlen doppelt vorkommen.

```
    // Programm: lotto.cpp
    // Beschreibung: Erzeugen von Lotto-Tips
    #include <iostream>      // Zusatzbibliothek für Ein-und Ausgaben wird
eingebunden
    #include <conio.h>       // Zusatzbibliothek für Konsole wird
eingebunden
    #include <stdlib.h>      // Wird für Zufallszahl-Generator benötigt
    #include <time.h>        // Wird für Initialisierung des Zufallszahl-
Generator benötigt
    using namespace std;     // Standard-Namensraum wird eingestellt

    void erzeugeQuicktipp(int a[6]);
    void ausgabe(int a[6]);

    int main(){
      int n;  //Anzahl der Tipps
      int tipp[6];
      srand(time(0));
      cout<<"Wie viele Tipps moechten Sie erhalten?"; cin>>n;
      for(int i=1;i<=n;i++){     //n Tipps ermitteln
        cout<<"\n"<<i<<". Tipp: ";
        erzeugeQuicktipp(tipp);
        ausgabe(tipp);
      }
      getch();
      return 0;
    }

    void erzeugeQuicktipp(int a[6]){
      for (int i=0; i<6; i++) {
        a[i]=rand()%45+1;
        //Überprüfe, ob Zahl bereits vorhanden
        for(int j=0;j<i;j++){   //Durchlaufe die vorhandenen Einträge
           if(a[i]==a[j]){  //und überprüfe sie auf Gleichheit mit dem
aktuellen Eintrag
            i--;        //ist die Zahl bereits vorhanden, so setze
                    //Zähler i zurück, damit diese Zufallszahl
                        //automatisch nochmals erzeugt wird!
           }
        }
      }
    }
```

```
    void ausgabe(int a[6]){
      for (int i=0; i<6; i++) {
         cout<<a[i]<<" ";
      }
    }
```

Aufgaben Eindimensionale Arrays

Aufgabe 1: Würfelsimulation

a) Entwickle eine Würfelsimulation, die es erlaubt 10 mal zu würfeln. Die Wurfergebnisse sollen dabei über den Zufallsgenerator erzeugt werden und in einem Integer-Array abgespeichert werden.
b) Ergänze das Programm derart, dass der Mittelwert der 10 Würfe berechnet wird und die Anzahl jener Würfe ermittel wird, die über dem Mittelwert liegen.

Hinweis 1: Es muss zuerst der Zufallsgenerator von CPP initialisiert werden (sonst erhält man stets die selbe Zufallsfolge)

```
srand (time(NULL)); // Intialisierung der srand Funktion
wurf[i]=rand()%6 + 1; // liefert eine Zahl zwischen 1 und 6
```

Aufgabe 2: Prüfsumme

Zum Schutz von Fehlern werden Texte oft mit einem Prüfbyte versehen, das aus den Zeichen eines Textes berechnet wird. Eine einfache Technik zur Berechnung eines Prüfbytes besteht darin, alle Zeichen des Textes mit der Operation XOR (bitweise exklusives oder, ^) zu verknüpfen. Schreibe eine Funktion, der ein Text als char-Array übergeben wird und die daraus ein Prüfbyte nach der beschriebenen Technik berechnet.

Aufgabe 3: Quadratzahlen

Schreibe ein Programm, die in einem Array die ersten 20 Quadratzahlen (1^2, 2^2,...) ablegt. Diese Quadratzahlen sollen nun nach geraden und ungeraden aufgeteilt werden. In ein Array sollen alle geraden kopiert werden und in ein anderes alle ungeraden. Gib alle geraden und ungeraden Quadratzahlen und die jeweilige Anzahl aus!

Aufgabe 4: Zufallszahlen

Ein Array namens Bereich soll mit einer vorgegebenen Anzahl von Zufallszahlen zwischen den Grenzen *ug* und *og* belegt werden. Dabei können sich die Zahlen wiederholen.
Hinweis 1: Es muss zuerst der Zufallsgenerator von CPP initialisiert werden (sonst erhält man stets die selbe Zufallsfolge)

```
srand (time(NULL)); // Intialisierung der srand Funktion  //
```

Hinweis 2: Wie man eine Zufallszahl zwischen den Grenzen *ug* und *og* zu erhalten kann, zeigt die folgende Zeile

```
zahl[i]=rand()%(og-ug+1)+ug;
```

Aufgabe 5: Zufallszahlen - Erweiterung

Gehe von der Aufgabenstellung von Aufgabe 4 aus. Es dürfen sich aber jetzt die erzeugten Zufallszahlen *nicht* wiederholen. Am Beginn wird die erste Zufallszahl erzeugt. Dann erfolgt die schrittweise Bestimmung der restlichen Zahlen. Dabei muss jede neue Zufallszahl mit allen vorher erzeugten Zahlen veglichen werden. Nur wenn sie mit keiner davon übereinstimmt, wird sie in den Bereich übernommen.

Aufgabe 6: - Primzahlenberechnung mit dem Sieb des Erathostenes

Zur Berechnung aller Primzahlen bis zu einer Obergrenze n gibt es ein raffiniertes Verfahren, das vom griechischem Mahtematiker *Erathostenes* stammt. Es wird das *Sieb des Erathostenes* genannt und basiert auf der Idee, dass die Vielfachen einer Primzahl mit Sicherheit keine Primzahlen sind. Wenn wir z.B. wissen, dass 2 eine Primzahl ist, können wir alle Vielfache von 2 aus der Menge der Primzahlenkandidaten streichen. Die kleinste Zahl der dann verbleibenden Menge ist die nächste Primzahl. Wir eliminieren wiederum alle ihre Vielfache.

Implementieren lässt sich dieser Algorithmus sehr schön mit Hilfe eines Arrays der Länge n mit boolean Elementen.

Aufgabe 7: Sequentielle Suche

Schreibe eine Anwendung, die 25 Zufallszahlen zwischen -10 und 10 in ein Array schreibt. Der Anwender soll eine zu suchende Zahl eingeben und die Anwendung soll im Array nach der Zahl suchen. Das Programm soll anschließend ausgegeben, ob die gesuchte Zahl im Array enthalten ist oder nicht.

Programmiere die Suche als sequentielle Suche, d.h. fange beim ersten Arrayelement an und vergleiche es mit dem Suchwert. Falls es nicht das gesuchte Element ist, gehe zum nächsten Arrayelement usw..

Überlege dir, wie du Zufallszahlen zwischen -10 und 10 geschickt erzeugen kannst.

Aufgabe 8: Auswahlsort

Realisiere einen Sortieralgorithmus (Auswahlsort) um 10 zufällig erzeugte Integer-Werte zwischen 0 und 99 eines Arrays zu sortieren:

Um die ersten n Elemente eines Arrays a (z.B. des Datentyps int) aufsteigend zu sortieren, kann man folgendermaßen vorgehen:

- Zuerst sucht man im Indexbereich 0..n–1 nach dem Index des kleinsten Elements:
- Danach vertauscht man a[0] mit a[min], so dass das kleinste Element in Position 0 steht.
- Diese Vorgehensweise wiederholt man für die Indexbereiche 1 .. n–1, 2 .. n–1 bis schließlich n–2 .. n–1. Dadurch wird sukzessive das zweitkleinste Element an die Position 1, das drittkleinste an die Position 2 übertragen usw.

Bsp.:
n=7

arr[0]=5
arr[1]=3
arr[2]=2
arr[3]=19
arr[4]=7
arr[5]=8
arr[6]=9

Vorgangsweise:
5 3 **2** 19 7 8 9 (zu sortierender Bereich 0 bis 6)

2 | **3** 5 19 7 8 9 (zu sortierender Bereich 1 bis 6)

2 3 | **5** 19 7 8 9 (zu sortierender Bereich 2 bis 6)

2 3 5 | 19 **7** 8 9 (zu sortierender Bereich 3 bis 6)

2 3 5 7 | 19 **8** 9 (zu sortierender Bereich 4 bis 6)

2 3 5 7 8 | 19 **9** (zu sortierender Bereich 5 bis 6)

2 3 5 7 8 9 | **19** (zu sortierender Bereich 6 bis 6)

Aufgabe 9: Array-Übung mit Zufallszahlen

Das Programm erstellt 12 Zufallszahlen zwischen 1 und 100

- Die Summe aller Zahlen wird ausgegeben.
- Der Mittelwert wird ausgegeben.
- Die Anzahl der geraden Zahlen wird ausgegeben.
- Benutzer gibt eine Zahl ein, es wird überprüft, ob die Zahl vorkommt.
- Es wird überprüft, ob irgendwelche Zahlen doppelt vorkommen.

Aufgabe 10: Bankomat

Realisiere einen Bankomaten mit einem Array, in welchem der jeweilige Wert der Scheine gespeichert ist und einem zweiten Array, in welchem die jeweilige Anzahl gespeichert ist:

```
wert[0] = 500,
wert[1] = 200,
```

```
wert[2] = 100,
...
anzahl[0] = Anzahl der 500er-Scheine
anzahl[1] = Anzahl der 200er-Scheine
anzahl[2] = Anzahl der 100er-Scheine
...
```

Das Besondere an diesem Bankomat ist, dass man weder Karte noch Code benötigt, um abzuheben 😉

Ein neuer Bankomat ist leer, d.h. alle Werte im Array anzahl haben den Wert 0.

Daher muss der Bankomat befüllt werden. Das Unterprogramm **befuellen(...)** hat 2 Parameter:

- schein: gibt an, welche scheine nachgelegt werden (z.B. 50 - es werden 50er-Scheine nachgelegt).
- anzahl: gibt an, wieviele Scheine nachgelegt werden.

Mit dem Unterprgoramm **auszahlen(...)** kann man den Bankomat veranlassen, einen bestimmten Betrag auszuzahlen. Der Automat muss nun berechnen, ob und mit welchen Scheinen er den Betrag auszahlen kann. Dabei gilt die Regel, dass er immer mit den größtmöglichen Scheinen auszahlt. Die Anzahl der jeweiligen Scheine wird auf der Konsole ausgegeben.

Beispiel: Der Betrag 790 soll ausgezahlt werden - Ausgabe auf der Konsole:

- 500-Scheine: 1
- 200-Scheine: 1
- 50-Scheine: 1
- 20-Scheine: 2

Natürlich muss dabei überprüft werden, ob die benötigten Scheine überhaupt verfügbar sind. Falls nicht, wird eine Fehlermeldung ausgegeben: Betrag nicht verfügbar.

Bei einer Auszahlung müssen auch die Werte im entsprechenden Array reduziert werden.

Es soll intern jeweils eine Schleife verwendet werden und so programmiert sein, dass man ohne großen Aufwand daraus einen Bankomat machen könnte, der beispielsweise auch 5er, 2er und 1er auszahlt.

Der Benutzer wird immer wieder gefragt, ob er einzahlen, auszahlen oder das Programm beenden möchte.

Aufgabe 11 - Verständnis-Aufgabe

Gib an, welche der mit a) bis g) bezeichneten Anweisungen syntaktisch korrekt sind. Falls ja, beschreiben das Ergebnis dieser Anweisungen.

```
void ArrayTest1()
{
int a[10];
for (int i=1; i<=10;i++) a[i] = 0; // a)
```

```
int b[2], c[2], d[3];
b[0]=0; b[1]=1;
c=b; // b)
int x=b[b[b[0]]]; // c)
c[0]=0; c[1]=1;
d[0]=0; d[1]=1; d[2]=2;
if (b==c) x++; // d)
if (c==d) x++; // e)
int s1=sizeof(a); // f)
int s2=sizeof(a)/sizeof(a[0]); // g)
}
```

Weitere Übungsaufgaben

Schreibe ein Programm, dass

Ex 1

...das eine Folge von n ganzen Zahlen einliest und jene Zahlen aus der Folge löscht, die mehrmals vorkommen. Anschließend soll die Folge ausgegeben werden.

Lösung

delmultiple.cpp

```
#include <iostream>
using namespace std;

main(){
int n;
cout<<"Anzahl Zahlen: "; cin>>n; cout<<endl;

int zahl[n];
cout<<"\nZahlen: \n";
for(int i=0;i<n;i++)cin>>zahl[i];

sort(zahl,zahl+n);

cout<<"\nSortiert ohne Wiederholung: \n";
cout<<zahl[0] <<" ";
for(int i=1;i<n;i++)
   if(zahl[i]!=zahl[i-1]) cout<<zahl[i]<<" ";

cout<<endl<<endl;
system("PAUSE");
```

Ex 2

...das eine Folge von n ganzen Zahlen einliest und jene Zahlen aus der Folge löscht, die größer als eine vorgegebene Zahl sind. Anschließend soll die Folge ausgegeben werden.

Lösung

delgreater.cpp

```
#include <iostream>
using namespace std;

main(){
int *zf, max,anz;
cout<<"Anzahl der Zahlen: ";cin>>anz;
zf=new int[anz];
cout<<"\nMaxiaml hoechste Zahl: ";cin>>max;
cout<<"\nZahlen:\n";
for(int i=0;i<anz;i++)cin>>zf[i];
for(int j=0;j<anz;j++){if(zf[j]>max){zf[j]=max+1;}}
cout<<"\nSortierte, begrenzte Folge: \n";
sort(zf,zf+anz);
for(int l=0;l<anz;l++){if(zf[l]!=max+1){cout<<zf[l] <<" ";}}
delete[] zf;

cout<<endl<<endl;
system("PAUSE");
```

}

Ex 3

...das eine maximal 80 Zeichen lange Zeichenfolge umdreht.

Lösung

revers.cpp

```
#include <iostream>
#include <conio.h>
using namespace std;

const char ENTER=13;

int main(){
 int i=0;
 char zeichen, zkette[80];
 cout<<"Zeichenfolge eingeben:\n";
```

```
 do {
   zeichen = getche();
   zkette[i]=zeichen;
   i++;
 } while (zeichen!=ENTER && i<80);
 cout<<endl<<endl;
 reverse(zkette,zkette+i);
 for(int j=0;j<i;j++)cout<<zkette[j];

 cout<<endl;
 system("PAUSE");
}
```

Ex 3a

...das an einer maximal 40 Zeichen langen Zeichenfolge überprüft, ob es sich um ein Palindrom handelt.

Hinweis: Von einem Palindrom spricht man dann, wenn ein Wort von vorne und hinten gelesen gleich aussieht, z.B. LAGERREGAL

Lösung

palindrom.cpp

```
#include <iostream>
#include <conio.h>
using namespace std;
const char CR=13;

main()
{char zeichen; char zkette[100];
 cout<<"Geben Sie Ihre Zeichenkette ein (max. 100): \n";
 // Eingabe der Zeichenkette
 int i=0;
 do {
   zeichen = getche();
   zkette[i]=zeichen;
   i++;
 } while (zeichen!=CR);
 int laenge=i-1;
 // Ausgabe verkehrt
 cout<<"\nRueckwaerts gelesen:\n ";
 for(i=laenge-1;i>=0;i--) cout<<zkette[i];
 // Vergleich auf Palindrom-Eigenschaft
 i=0; while(zkette[i]==zkette[laenge-i-1]) i++;
 if (i==laenge) cout<<"\nEs handelt sich um ein Palindrom.";
 else cout<<"\nEs ist kein Palindrom.";
```

```
 cout<<"\n\n"; system("PAUSE");
}
```

Ex 4

...das zwei steigend sortierte Buchstabenfolgen zu einer sortierten Folge zusammenfasst. In der Ergebnisfolge dürfen keine gleichen Buchstaben vorkommen. Bei der Eingabe soll überprüft werden, ob die Buchstabenfolgen steigend sortiert sind.

Ex 5

...das zwei monoton steigende Zahlenfolgen zu einer monoton steigenden Zahlenfolge zusammenfasst und ausgibt.

Ex 6

...das zwei gleichlange Wörter einliest und den HAMMING-Abstand berechnet: Als HAMMING-Abstand zweier Wörter mit gleichlangen Binärcodes wird die Anzahl jener Stellen bezeichnet, in denen sich die beiden Wörter unterscheiden.

Beispiel:
001010110 101000110
HAMMING-Abstand:2

Ex 7

...das Vektoren verknüpft. Folgende Operationen sollen durchführbar sein:

- Addition zweier Vektoren,
- Subtraktion zweier Vektoren,
- Multiplikation von Vektor und Skalar,
- skalares Produkt zweier Vektoren.

Die Anzahl der Elemente der Vektoren soll mittels Konstante definiert werden. z.B. `#define MAX 5`

Ex 8

...das wie ein Auszählreim arbeitet: n Kinder sind im Kreis aufgestellt. Nach Aufsagen eines Auszählreimes mit m Silben wird das jeweils m-te Kind im Kreis ausgeschieden. Auszugeben ist die Reihenfolge, in der die Kinder ausscheiden.

Anleitung: Jedem Kind ist ein Element eines Arrays zugeordnet. Zu Beginn werden sämtliche Elemente des Arrays gleich 1 gesetzt. Danach werden die Elemente zyklisch durchlaufen und das jeweils m-te Element gleich 0 gesetzt. Die Indizes der 0 gesetzten Elemente entsprechen den Nummern der ausgeschiedenen Kinder.

Lösung

auszaehlreim.cpp

```
// Auszählreim
#include <iostream>
#include <conio.h>
#include <time.h>
using namespace std;

int main()
{int n,m;
 cout << "\nBitte Anzahl der Kinder und Anzahl der Silben eingeben:\n";
 cin >> n >> m;
 int kind[n];
 // Initialierung des Feldes
 for (int i=0;i<n;i++) kind[i]=1;
 // Ausgabe
 // for (int i=0;i<n;i++) cout << "(" << i <<"|" << kind[i] <<")";
 cout << endl;

 clock_t sz,ez;
 sz = clock();

 // Auszählreim
 int pos=-1;
 for (int i=0;i<n;i++) {
    int z=1;
    while (z<=m){
       pos=(pos+1)%n;
       if (kind[pos]) z++;
    }
    kind[pos]=0;z=0;
    cout <<  pos  << ' ';
 }
 ez = clock();
 cout << "\nGesamtzeit: " << (ez-sz)/(double)CLOCKS_PER_SEC << "\n\n";


 system("PAUSE");
 return 0;
}
```

Ex 9

...das alle Primzahlen, die kleiner oder gleich der natürlichen Zahl n sind, ausgibt.

Anleitung (Siebverfahren des ERATOSTHENES): Verwende ein Array mit dem Index 0..n-2, dessen Elemente gleich 1 (wahr) gesetzt werden. Danach betrachtet man mit dem Index 0 beginnend der Reihe nach sämtliche Elemente mit dem Wert wahr und setzt jene Elemente gleich 0 (falsch), deren Indizes Vielfache des gerade betrachteten Indexwertes+2 sind. Die Indizes jener Elemente, die den Wert wahr beinhalten, sind Primzahlen.

Lösung

siebdeseratosthenes.cpp

```
#include <iostream>
using namespace std;

main(){
 int n;
 cout <<"Geben Sie eine natuerliche Zahl ein: "; cin >>n; int z[n+1];
 z[1]=0; for(int i=2;i<=n;i++) z[i]=1;
 for(int i=3;i<=n;i++)
   for(int j=2;j<i;j++) if(i%j==0) z[i]=0;
 cout<<"\nAlle Primzahlen, die kleiner oder gleich der natuerlichen
Zahl "<<n<<" sind:\n\n";
 for(int i=1;i<=n;i++)if(z[i]==1) cout<<i<<' ';
 cout << "\n\n"; system("PAUSE");
}
```

Ex 10

...das ein Kartenspiel von 52 Karten an beliebig viele Spieler austeilt und das Blatt jedes Spielers sortiert ausgibt. Die Anzahl der Spieler soll eingegeben werden. Beispiel:

Spieler1:
Karo: 2, 5, 7, 9, 10, Dame
Herz: 3, 4, König, Ass
Pik : 4, 5, 6, Bub, Ass
Treff: 7, 9, 10

Spieler2:
Karo: 3, 4, König, Ass
Herz: 2, 7, 9, Bub
Pik : Dame, König
Treff: 2, 3, 5, 6, Bub, König, Ass

Spieler3:
Karo: 6, 8, Bub
Herz: 5, 6, 8, 10, Dame
Pik : 2, 3, 7, 8, 9, 10
Treff: 4, 8, Dame

Ex 11

... das einen Text gemäß der Cäsar-Verschlüsselung codiert. Dabei wird jeder Buchstabe eines Textes durch seinen m-ten Nachfolger erstetzt.

z.B. Hallo Welt, m=3
wird zu Kdoor Zhow

Lösung

caesarverschluesselung.cpp

```
// Cäsar-Verschlüsselung
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{int m,i=0;
 char text[100], vtext[100], zeichen;
 cout << "Cäsar-Vershluesselung\n\n";
 cout << "Bitte Verschiebung m angeben: "; cin >> m;
 // Eingabe des Textes
 cout << "\nBitte zu verschluesselnden Text eingeben ('.' fuer Ende)
\n";
 do {
   zeichen = getche(); // cout << '*';
   text[i]=zeichen;
   i++;
 }
 while (zeichen!='.');
 // Ausgabe des Textes
 cout << endl;
 for (int j=0;j<(i-1);j++) cout << text[j];
 cout << endl;

 // Verschluesselung
 for (int j=0;j<(i-1);j++) {
   // Großbuchstaben verschlüsseln
   if (text[j]>= 65 && text[j] <=90) vtext[j]=(text[j]-65+m)%26+65;
   // Großbuchstaben verschlüsseln
   if (text[j]>= 97 && text[j] <=122) vtext[j]=(text[j]-97+m)%26+97;
 }

 // Ausgabe des codierten Textes
 cout << endl;
 for (int j=0;j<(i-1);j++) cout << vtext[j];
 cout << endl;


 system("PAUSE");
 return 0;
}
```

Rekursionen

Die **Rekursion** ist ein **spezieller Aufruf von Funktionen**, nämlich wenn Funktionen sich **selbst aufrufen**. Da bei einem Aufruf sich die Funktion wieder selbst aufruft, benötigt die Funktion wie bei den Schleifen eine **Abbruchbedingung**, damit die Selbstaufrufe nicht endlos sind.

Beispiel

In dem folgenden Beispiel, welches lediglich eine Bildschirmausgabe via Rekursion zeigt, wird der **Abbruch anhand einer Zählvariable** entschieden – wie bei den Schleifen. Ist x größer 0 erfolgt eine Ausgabe und ein rekursiver Aufruf mit einem dekrementierten (=verringerten) Zählwert. Ist der Zählwert bei 0 angelangt, erfolgt kein rekursiver Aufruf mehr.

```
#include<stdio.h>

using namespace std;

printLines(int x);

int main() {
    printLines(5);
    return 0;
}

printLines(int x){
    if(x > 0) {
        cout << "\nZeile Nr. " << x);
        printLines(x-1);
    }
}
```

Ausgabe des Programms

```
Zeile Nr. 5
Zeile Nr. 4
Zeile Nr. 3
Zeile Nr. 2
Zeile Nr. 1
```

Beispiel Fakultät

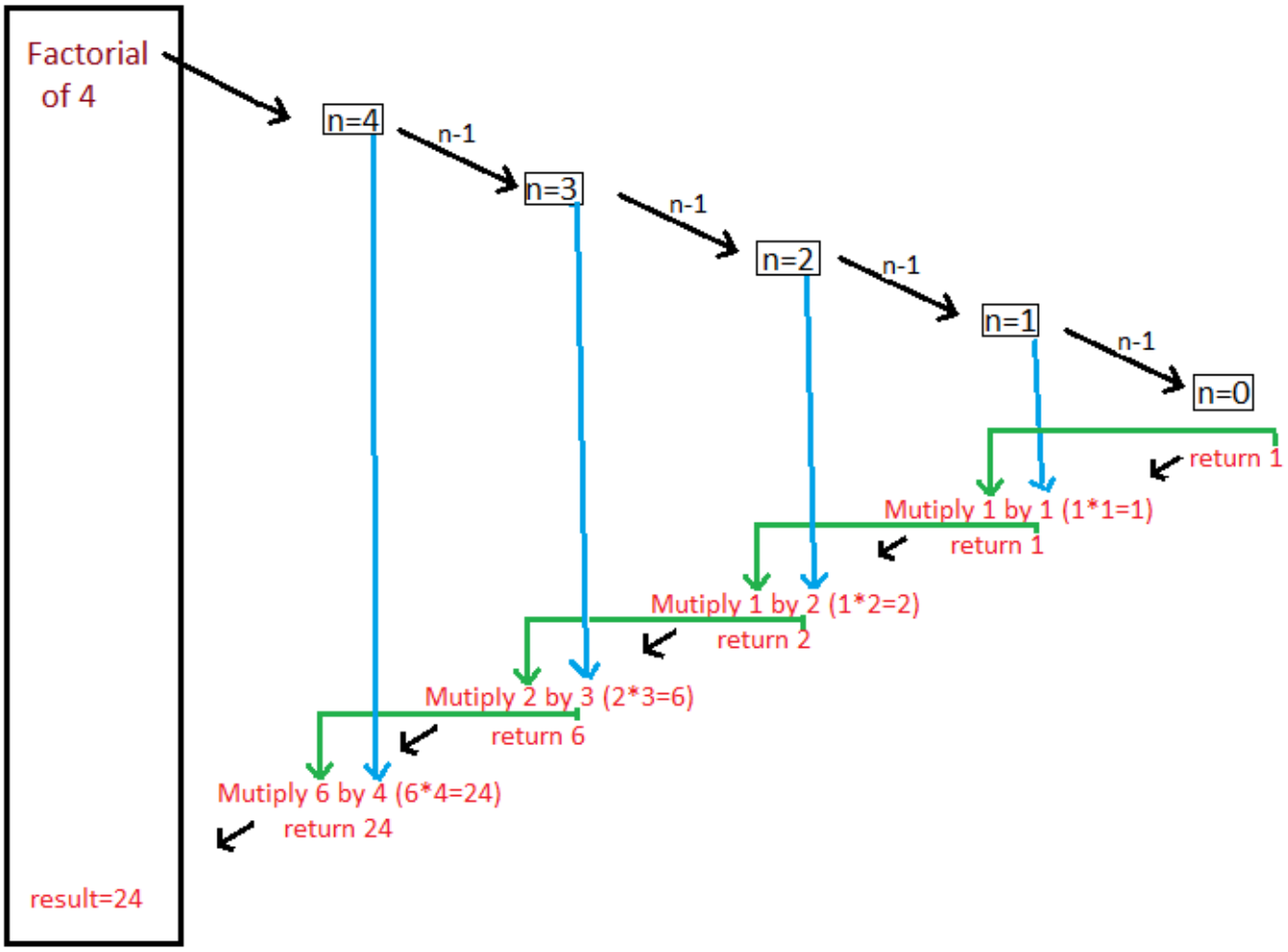
Nun ein sinnvolleres und gern verwendetes Beispiel, die **Berechnung der Fakultät mittels Rekursion**. Bei der Berechnung der Fakultät wird solange ein Produkt mit der dekrementierten Zahl gebildet, bis die Zahl bei der 1 angelangt ist. Zum Beispiel ist die Fakultät Vier: 4! = 4 * 3 * 2 * 1 =

24.

```
#include<stdio.h>

using namespace std;

int fakultaet(int x);

int main() {
    int a = 6;
    cout << "Fakultaet von " << a << " ist " << fakultaet(a);
    return 0;
}

int fakultaet(int x) {
    if(x > 1) {
        return x * fakultaet(x-1);
    }else {
        return 1;
    }
}
```

Ausgabe

```
Fakultaet von 6 ist 720
```



Sortieralgorithmen

Sortierproblem

Das Sortierproblem besteht darin, Datensätze eines gegebenen Datenbestands sortiert anzuordnen. Im Folgenden geht es darum, das Sortierproblem und seine Relevanz in der Informatik genauer zu betrachten.

Lösung des Sortierproblems

Zur Lösung des Sortierproblems sind eine Vielzahl an Verfahren entwickelt worden. Wir werden einige dieser Verfahren hier vorstellen und zur Verdeutlichung der Komplexitätsbetrachtungen in den folgenden Abschnitten nutzen. Um die Ideen und Komplexitätsbetrachtungen möglichst einfach zu gestalten, sollen nur Zahlen anstelle komplexer Datensätze betrachtet werden.

- Grundlagen Sortieralgorithmen
- Beschreibung verschiedener Sortieralgorithmen
- Sortieralgorithmen animiert
- http://www.sorting-algorithms.com/

2) DATENBANKEN

3) PHP

3.1) Was ist PHP?

3.2) Grundlegende Sprachelemente

3.3) Variablen und Operatoren

3.4) Interaktive Webseiten

3.5) Anführungszeichen

3.6) Kontrollstrukturen

3.7) Schleifen

3.8) Felder

3.9) Formulare

3.10) Externe Dateien

3.11) Datum und Zeit

3.12) Funktionen

---

4) Frameworks

Ein Framework ist selbst noch kein fertiges Programm, sondern stellt den Rahmen zur Verfügung, innerhalb dessen der Programmierer eine Anwendung erstellt, wobei u. a. durch die in dem Framework verwendeten Entwurfsmuster auch die Struktur der individuellen Anwendung beeinflusst wird.

- Ein Framework können Sie als eine Vorprogrammierung verstehen. Verschiedene Funktionen und Elemente sind bereits enthalten und müssen nicht jedes mal neu programmiert werden. Auf diese Elemente kann der Entwickler innerhalb des Framework zugreifen.
- Es gibt verschieden Arten von Frameworks. Der Begriff „.NET Framework“ wird Ihnen sicher schon mal begegnet sein, wenn Sie sich für Programm-Code interessieren. Das ist beispielsweise das Framework für Microsoft-Anwendungen. Tipp: In einem anderen Artikel lesen Sie, wofür Sie das .NET-Framework brauchen.
- Frameworks gibt es nicht nur für Programme. Zur Erstellung von dynamischen Webseiten werden etwa Web-Frameworks (z.B. Bootstrap) bereitgestellt. Und Programmierer testen Ihre Software beispielsweise mit sogenannten Test-Frameworks.

4.1) Vorteile

- Wiederkehrende Aufgaben sind im Framework schon „vorprogrammiert“ und können beliebig oft wiederverwendet werden.
- Ein Framework bietet außerdem genormte Schnittstellen zu bestimmten Quellen, etwa zu Datenbanken. Dadurch lässt sich eine Quelle einfacher ansprechen.
- Frameworks erleichtern die Programmierarbeit und sparen dem Entwickler viel Zeit.

4.2) Bootstrap als HTML/CSS & JS -Framework

Bootstrap ist ein **Frontend**-Framework, mit dessen Hilfe Webentwickler geräteübergreifende Websites verschiedenster Art erstellen können. Zu diesem Zweck bietet das Open-Source-Projekt diverse Gestaltungsvorlagen, die auf HTML und CSS basieren, sowie optionale JavaScript-Erweiterungen.

4.2.1) Entwicklung von Bootstrap

Ursprünglich wurde Bootstrap als internes Werkzeug für Twitter entwickelt, um der Nutzung unterschiedlicher Bibliotheken und der damit einhergehenden Inkonsistenz Einhalt zu gebieten. Als sich herausstellte, dass Bootstrap viel weitreichender genutzt werden konnte, veröffentlichte Twitter das Werkzeug im August 2011 als Open Source Projekt. Es steht über den Hosting-Dienst GitHub zur Verfügung und wird seit der Veröffentlichung konsequent von freien Entwicklern erweitert.

4.2.2) Funktionen von Bootstrap

Dank der HTML- und CSS-Vorlagen müssen Sie bei einer guten Idee für eine neue Website nicht mit der Entwicklung komplett von vorn beginnen. Sie übernehmen wie aus einem Baukasten vorgefertigte Inhalte und binden sie in Ihr HTML-Dokument ein. Dadurch entfallen viele der sonst sehr mühsamen CSS-Konfigurationen, was Ihnen eine Menge Arbeit erspart. Abgedeckt sind unter anderem die folgenden Elemente:

- Typografie und Hervorhebungen
- Formulare
- Buttons/Schaltflächen
- Icons
- Dropdown-Menüs
- Tabellen
- Navigationsmöglichkeiten
- Breadcrumbs
- Grid-System
- Pagination
- Labels und Badgets
- Prozessleisten
- Alarmfenster

4.2.3) Aufbau von Bootstrap

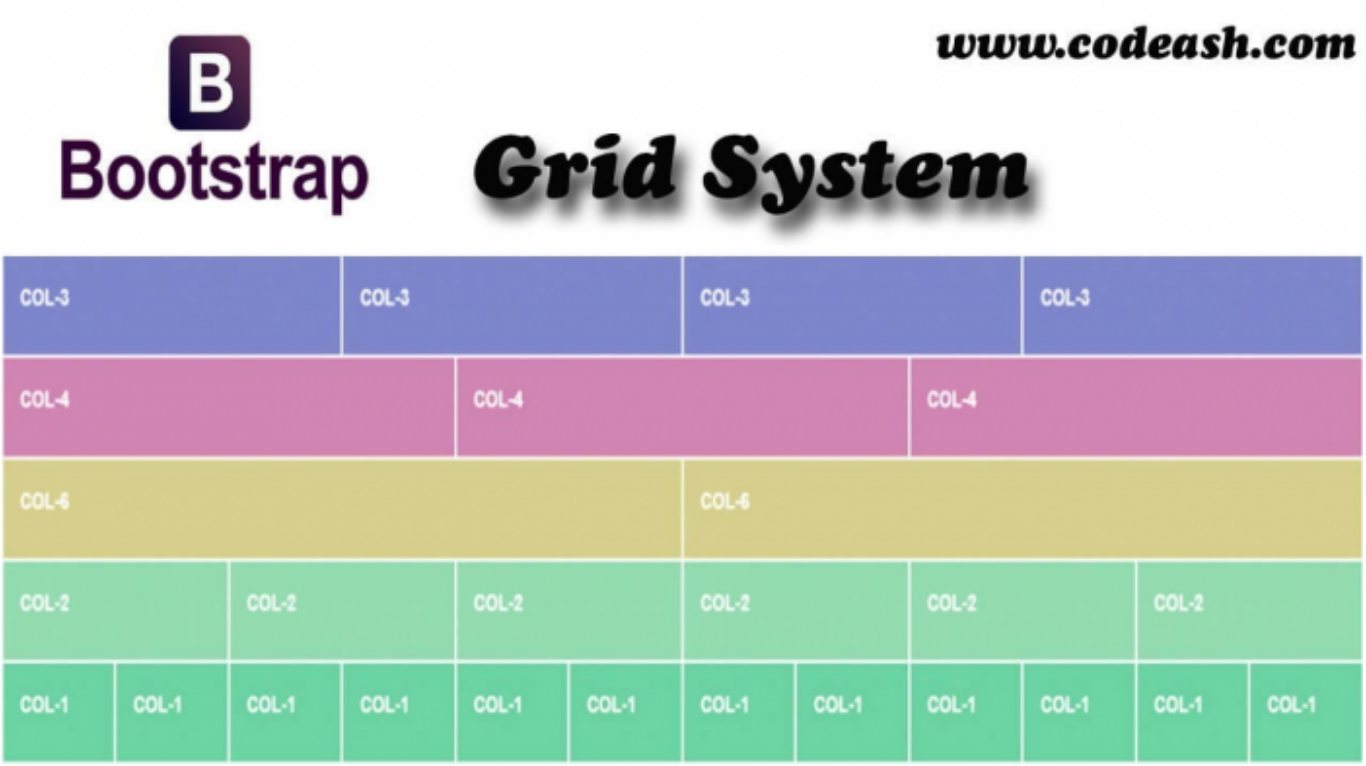
Bootstrap besteht aus einzelnen Komponenten, die du in deine Website einfügen kannst. Dabei handelt es sich um HTML-Codes, die mit CSS-Klassen und ggf. Javascript-Code ausgestattet sind. Bootstrap 4 ist modular aufgebaut und wird in folgende vier Bereiche gegliedert:

- Layout
- Inhalt
- Komponenten
- Werkzeuge

Layout

Das Herzstück von Bootstrap 4 ist das Raster. Standardmäßig verwendet Bootstrap ein 12-spaltiges Raster-Layout mit einer Breite von 1140 Pixel (.container). Dabei wird die Rasterbreite entsprechend der Displaygröße via CSS-Breakpoints angepasst. Eine Alternative bietet die Klasse .container-fluid, welche den Container immer über die komplette Displaygröße darstellt

Ein Raster besteht aus Zeilen (.row) und Spalten (.col), wobei die Spalten mithilfe von CSS-Klassen passend für das jeweilige Endgerät angeordnet werden können.



Inhalt

Nachdem du das Raster verstanden hast, sollen im Folgenden die Möglichkeiten aufgezeigt werden, wie du den Inhalt deiner Website anpassen kannst.

- Überschriften
- Bilder mit Unterschriften
- Tabellen
- …

Komponenten

Das Herzstück von Bootstrap sind die Komponenten, die dir helfen, deine Website zu strukturieren.

- Buttons
- Forms
- Breadcrumbs
- Navbars
- Accordion
- …

4.2.4) Vorteile von Bootstrap

- Es eignet sich, um Responsive Designs umzusetzen.
- Es gibt ein großes Angebot an vorgefertigten Themes und Elementen, die direkt verwendet werden können.
- Das Framework lässt sich erweitern und anpassen.
- Das System ist logisch strukturiert und einfach umsetzbar.
- Es ist durch JavaScript und jQuery-Plugins erweiterbar.
- Es realisiert eine hohe Zeitersparnis durch vorgefertigte Komponenten.
- Bootstrap verfügt über eine hervorragende Kompatibilität zu verschiedenen alten und neuen Browsern.
- Der Wartungsaufwand ist im Vergleich zum Einsatz verschiedener Frameworks geringer.

4.2.5) Nachteile von Bootstrap

Bootstrap ist nicht grade klein. Schon die Grundausstattung Bootstrap mit Grid, Typo, Tabellen und Formularen schlägt mit 90K plus zu Buche. Komplett mit jQuery, Glyphicons usw. müssen 350K geladen werden und somit die Ladezeit erhöhen.

PM - Projektmanagement

Die ersten 90% einer Aufgabe nehmen 10% der Zeit in Anspruch, die letzten 10% erfordern die anderen 90% der Zeit.

Die Deadline ist nur noch einen Schritt entfernt. Der Geschäftsführer erwartet die Abschlusspräsentation. Und der Auftraggeber steht schon in den Startlöchern. Doch die Projektmanager bleiben cool, denn sie haben den Überblick und alles im Griff.

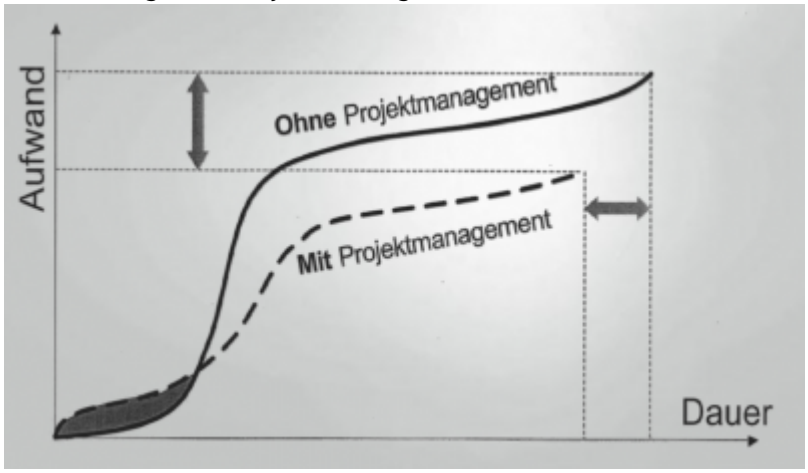
IT-Projektmanager sind die **menschliche Schnittstelle zwischen Anwendungsentwicklung und Anwendern**.

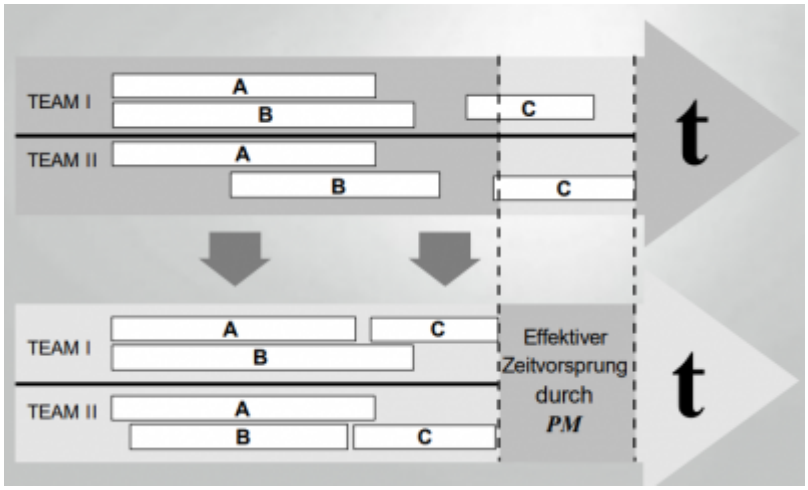
Projektmanager **planen**, **organisieren** und **steuern informationstechnische Projekte**, die die **Zusammenarbeit von Spezialisten aus unterschiedlichen Fachgebieten** erfordern. Sie **koordinieren alle beteiligten Mitarbeiter, Abteilungen und externe Dienstleister**. Während der Umsetzung behalten sie den **Zeitplan** und das **Budget immer im Blick**. IT-Projektmanager tragen eine **doppelte Verantwortung**, weil sie nicht nur die **Ausführung, sondern auch die Zielerreichung** in der Hand haben.
Die Herausforderung: **Mit so vielen Ressourcen wie nötig und so wenigen wie möglich das beste Ergebnis herauszuholen**.

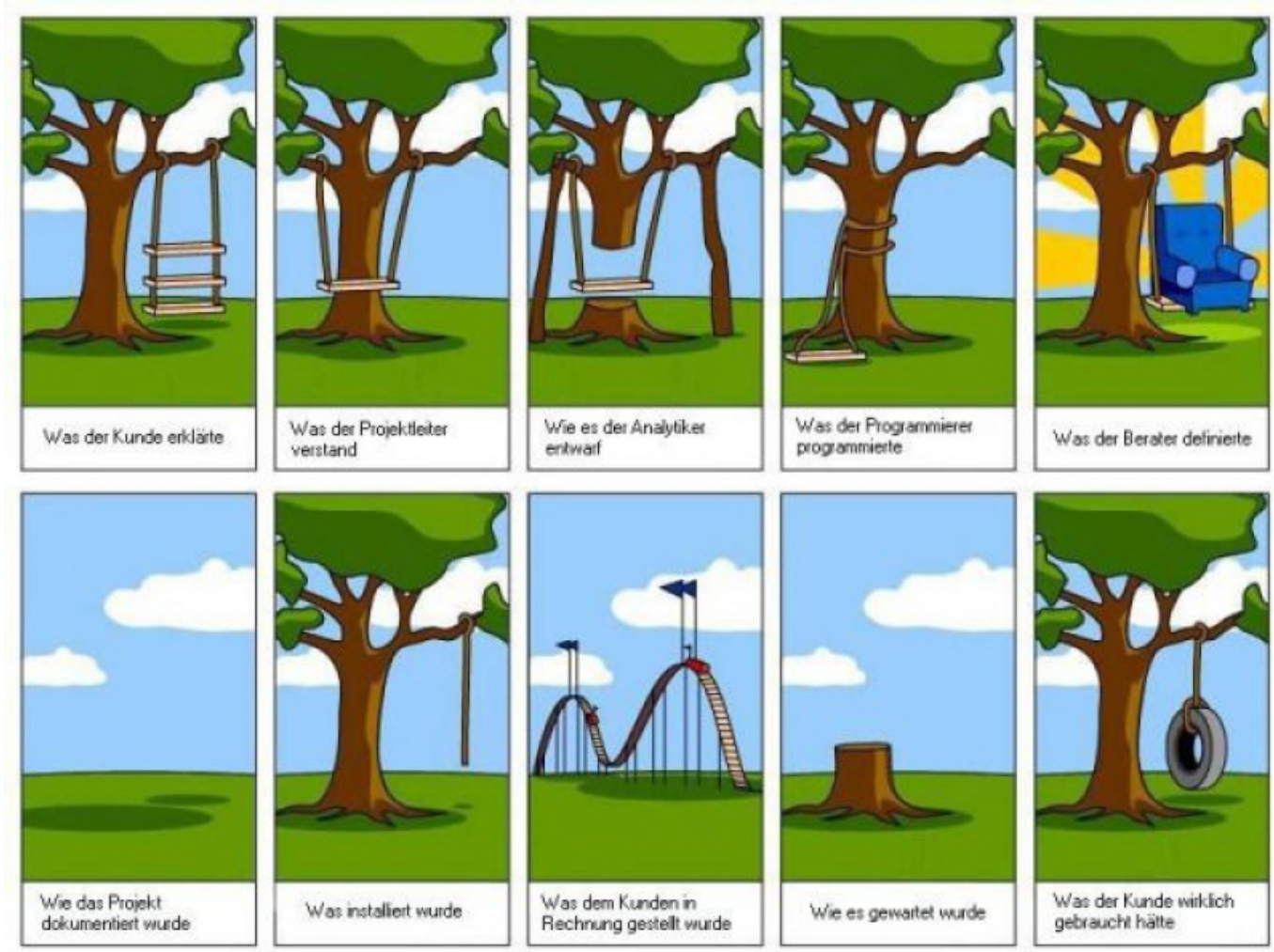
In der **Planungsphase** jonglierst Du als IT-Projektmanager **zwischen System Operator, leitendem Architekt, Head of Development, Chefdesigner und Qualitätsmanager**. Du **förderst Zusammenarbeit** und **Austausch zwischen den Fachabteilungen**. Du definierst die **Anforderungen und Ziele**, erstellst **Release-Pläne** und kümmerst Dich um die **Aufgabenverteilung**. Außerdem triffst Du Entscheidungen über die **eingesetzten Methoden und Werkzeuge**. Denn je nach Projekt können andere Plattformen, Entwicklungsumgebungen, Architekturen und Programmiersprachen sinnvoll sein.

Warum Projektmanagement?

Mit einem guten Projektmanagement kann man **Ressourcen optimal einsetzen**!







}}

IT-Projektmanagement kann manchmal ganz schön schwierig sein – aber Du wirst das schon schaukeln!

Modelle des Projektmanagements

Die traditionellen, sequentiellen Vorgehensmodelle

Traditionelles, klassisches oder konventionelles Projektmanagement meint alle Projekte, die einem

klaren und strengen Plan folgen. Die drei wichtigsten Projektfaktoren sind Zeit, Kosten und Qualität bzw. Umfang. Entsprechend dieser Faktoren werden Projektziele möglichst exakt definiert und Pläne akribisch ausgestaltet. Wie viele Kosten müssen eingeplant werden? Welche Termine müssen eingehalten werden? Der Projektmanager versucht alle Schritte vom Projektanfang bis Projektende zu kalkulieren und den Projektverlauf im Detail zu dokumentieren. Üblicherweise werden folgende Projektphasen unterschieden:

Projektphasen:

Die Projektphasen im klassischen Projektmanagement werden sequenziell, also nacheinander bearbeitet. Diese Reihenfolge ist wichtig. Traditionelle Projektmanagement-Methoden bieten durch ihre systematisches, strukturiertes und geregeltes Vorgehen Schutz vor Risiken. Projekterfolge werden durch Überwachung, Kontrolle und Dokumentation gemacht. Dafür sorgt die hierarchische Rollenverteilung von Auftraggeber, Projektleiter, Projekt Management Office (PMO) und Projekt-Team-Mitarbeitern.

Der Nachteil des traditionellen Projektmanagement liegt in der mangelhaften Flexibilität. In der Realität ändern sich die Rahmenbedingungen eines Projektes oft. Traditionelle Projektmanagement-Methoden stoßen dort an ihre Grenzen, wo uns Regeln und Vorschriften nicht weiterhelfen.

Vorteile

- Verbindlichkeit
- Stabilität
- Zuverlässigkeit
- Sicherheit
- Klarheit

Nachteile

- Aufwändige Planung
- Bürokratische Dokumentation
- Zu Beginn eines Projektes sind nicht alle Fakten bekannt.
- Anforderungen ändern sich im Verlauf des Projektes (z. B. Markt ändert sich).
- Neue Wünsche von Kunden und Stakeholdern im Verlauf des Projektes entstehen.
- Neue Ideen im Projektverlauf können nicht implementiert werden.
- Unflexibel gegenüber dynamischen Projektumgebungen.

Beispiele

- 5.1) Wasserfallmodell

Agiles Projektmanagement

Agilität ist weniger als einheitliches Vorgehensmodell, als eine Grundhaltung oder Philosophie mit Werten zu betrachten. Das agile Projektmanagement existiert nicht. Typisch für agile Ansätze ist jedoch, dass sie kundennah, teamorientiert, lean (schlank) und pragmatisch arbeiten. Die Konzepte setzen auf Kommunikation mit Kunden und im Team, statt auf einen intensiven Wissenstransfer über eine detaillierte Dokumentation. Das Kunden- und Teamfeedback kann unmittelbar und spontan in Entscheidungsprozesse einbezogen werden. Dafür sorgen schlanke Prozesse, Netzwerk-Strukturen und selbstbestimmtes, eigenverantwortliches Arbeiten. Im traditionellen Projektmanagement wird am Ende des Projektes das fertige Produkt geliefert. Im agilen Projektmanagement dagegen wird das „unfertige“ Produkt im Projektverlauf geliefert und gemäß der Kundenmeinungen und Erfahrungen nach und nach verbessert. Scrum ist die populärste agile Methode, die in Sprints abläuft. Sprints sind kurze iterative (sich wiederholende) Zyklen.

Zwar gibt es im Scrum einen geregelten Ablauf, dieser läuft jedoch in sich wiederholenden Schleifen (Sprints) ab. Jeder Sprint liefert Zwischenergebnisse und neue Erkenntnisse für den nächsten Sprint. Die Zwischenergebnisse können so den Anforderungen der Kunden kontinuierlich angepasst und verbessert werden.

In der schnelllebigen, digitalen Welt gibt es zu viele Unklarheiten, auf die man sich nicht vorbereiten kann. Agile Ansätze machen uns anpassungsfähig und flexibel. Andererseits fordert sie sehr viel von Individuen. Agile Werte, wie sie im Manifest beschrieben wurden, müssen gelebt werden, um Erfolge zu erzielen. Ohne Kommunikation und Eigenverantwortung der einzelnen Projektmitarbeiter kann die agile Vision leicht fehlschlagen.

Vorteile

- Projekterfolge seit Einführung agiler Methoden nachweisbar verbessert.
- Offenheit und Flexibilität sorgt für kurze Reaktionszeiten auf Veränderungen.
- Schnelle Projektentwicklung
- Sichtbare Projektfortschritte
- Fokus auf Aufgaben
- Wenig Bürokratie
- Commitment (Selbstverpflichtung und Identifikation des Teams mit Projektzielen)
- Transparenz in Kommunikation sorgt für Vertrauen.
- Schafft Mut zu Innovationen.
- Hohe Kundenzufriedenheit

Nachteile

- Hohe Komplexität von Projekten (z. B. Multiprojekte) schränkt das Prinzip Agilität ein.
- Verteilung von Verantwortung im Team ist auch Verteilung von Risiken. Beispiel: Mangel an Entscheidungsfreudigkeit im Team.
- Agile Projekte stehen und fallen mit Kommunikation.

- Agile Werte müssen von Teammitgliedern verstanden und verinnerlicht werden (Vision muss im Unternehmen geteilt werden).
- Soft Skills zählen (z. B. effizientes Zeit-, Selbst- und Teammanagement notwendig)
- Dokumentation kann zu minimalistisch sein.
- Nicht alle Aspekte einer Organisation können „agil“ sein oder gemacht werden.

Beispiele

- 5.2) Scrum

Hybride Methoden im Projektmanagement

Hybride Methoden oder Kombinationen sind so zahlreich, wie es traditionelle und agile Methoden gibt. Da ein detaillierter Blick auf die einzelnen Methoden den Rahmen dieses Beitrages sprengen würde, nennen wir einige Beispiele für hybride Methoden. Für Details und praktische Anwendungen verweisen wir auf unser Seminar Hybrides Projektmanagement:

Scrumban (agil + agil)

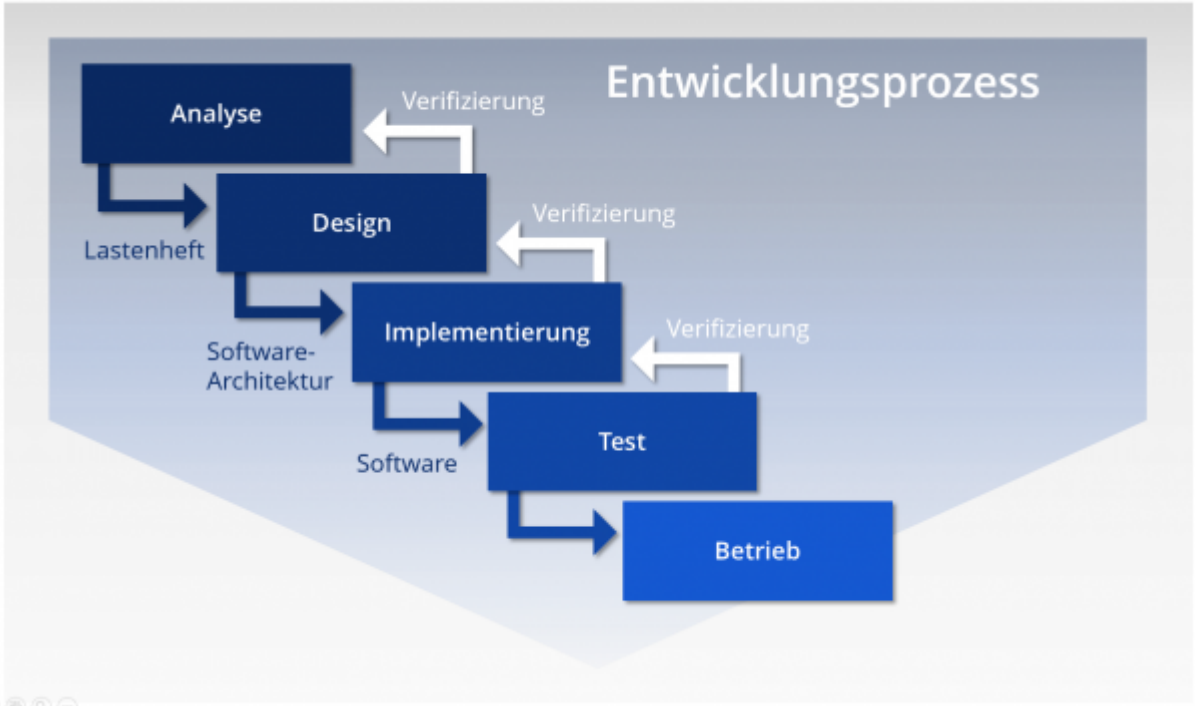
Die Kombination von Scrum und Kanban ermöglicht z. B. den Einsatz eines Kanban-Boards mit dem Ablaufmodell und der Rollenverteilung im Scrum. Die Vorteile: Beim Scrum werden Aufgaben sichtbarer und der Prozessfluss optimierbarer.

Wasserfall + Scrum (traditionell + agil)

Das Gesamtprojekt wird als Wasserfall durchgeführt, während eine Phase – hier bietet sich die Implementierungsphase an – als Scrum mit Sprints durchgeführt wird. Alle anderen Phasen vom Kundenwunsch bis zum Projektabschluss werden traditionell durchgeführt. Dies wird auch als Wasser-Scrum-Fall-Modell bezeichnet, weil Scrum in der Mitte des Wasserfalls steht.

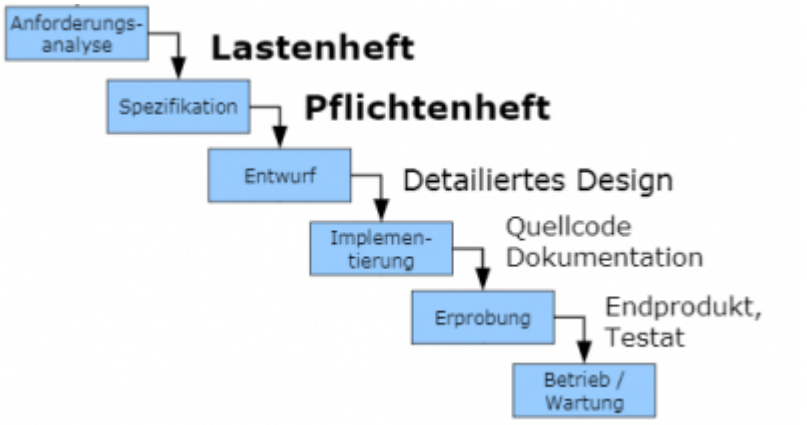
Wasserfallmodell

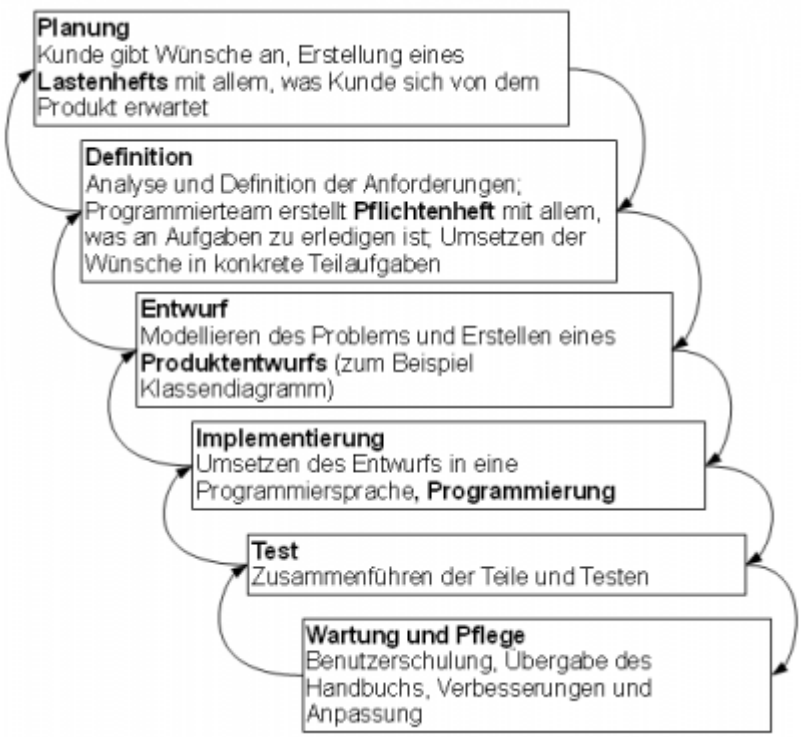
Wenn von klassischem Projektmanagement die Rede ist, ist meistens das **Wasserfallmodell** gemeint. Die Wasserfallmethode bildet **Projektphasen** in **Form einer Kaskade** oder eines **Wasserfalls** ab.



Jede Phase wird sequenziell, d. h. Schritt für Schritt abgearbeitet.

Erst wenn eine Phase abgeschlossen ist, folgt die nächste. Meist beschreibt das Modell auch einzelne Aktivitäten, die zur Herstellung der Ergebnisse durchzuführen sind. Zu bestimmten Meilensteinen und am jeweiligen Phasenende werden die vorgesehenen Entwicklungsdokumente im Rahmen des Projektmanagements verabschiedet. Alle Parameter einer Phase werden im Voraus definiert: Was sind die zu erfüllenden Aufgaben? Welche Dokumente müssen erstellt werden? In wechselhaften Branchen, in denen sich die Rahmenbedingungen eines Projektes leicht ändern können oder Projekte zu komplex sind, ist die Wasserfallmethode nicht zu empfehlen. Dafür punktet das Wasserfallmodell mit seiner klaren und leicht zu formalisierenden Struktur. Ein Klassiker, wenn es um Projektmanagement geht.





Analyse/Anforderungen

Jedes Softwareprojekt beginnt mit einer Analysephase, die eine Machbarkeitsstudie und eine Anforderungsdefinition umfasst. In der **Machbarkeitsstudie** wird das Software-Projekt bezüglich Kosten, Ertrag und Realisierbarkeit eingeschätzt. Aus der Machbarkeitsstudie gehen ein Lastenheft (eine grobe Beschreibung der Anforderungen), ein Projektplan und die Projektkalkulation hervor sowie ggf. ein Angebot für den Auftraggeber.

Anschließend folgt eine detaillierte **Anforderungsdefinition**, die eine Ist-Analyse und ein Soll-Konzept beinhaltet. Während Ist-Analysen den Problembereich skizzieren, wird im Soll-Konzept definiert, welche Funktionen und Eigenschaften das Software-Produkt bieten muss, um den Anforderungen gerecht zu werden. Zu den Ergebnissen der Anforderungsdefinition gehören beispielsweise ein Pflichtenheft, eine detaillierte Beschreibung, wie die Anforderungen an das Projekt zu erfüllen sind, sowie ein Plan für Akzeptanztest.

Abschließend sieht die erste Phase des Wasserfallmodells eine **Analyse der Anforderungsdefinition** vor, in der komplexe Probleme in kleine Teilaufgaben zerlegt und entsprechende Lösungsstrategien erarbeitet werden.

Lastenheft

Das **Lastenheft** (auch Anforderungsliste, Anforderungsspezifikation, Anforderungskatalog, Produktskizze, Kundenspezifikation oder Anwenderspezifikation) beschreibt die Gesamtheit der Anforderungen des Auftraggebers an die Lieferungen und Leistungen eines Auftragnehmers.

Die Anforderungen in einem Lastenheft sollten so allgemein wie möglich und so einschränkend wie nötig formuliert werden. Hierdurch hat der Auftragnehmer die Möglichkeit, eine passende Lösung (z.

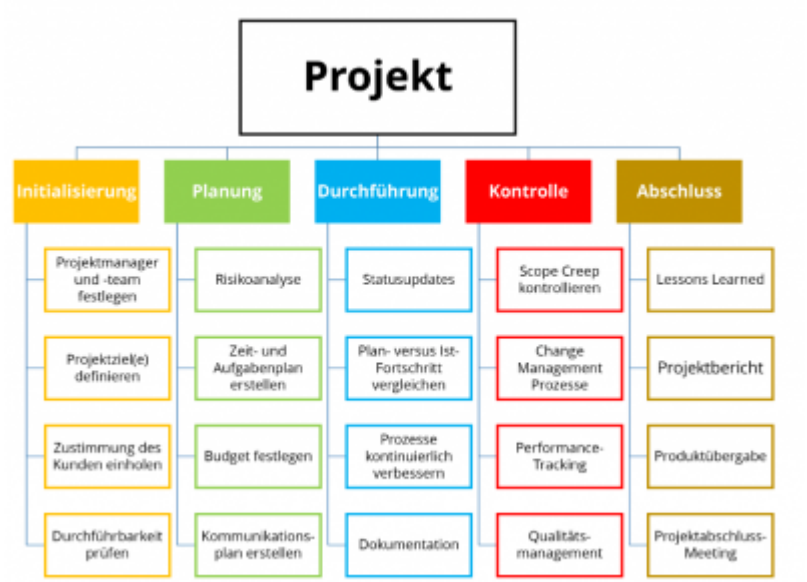
B. eine Software) zu erarbeiten, ohne in seiner Lösungskompetenz durch zu konkrete Anforderungen eingeschränkt zu sein.

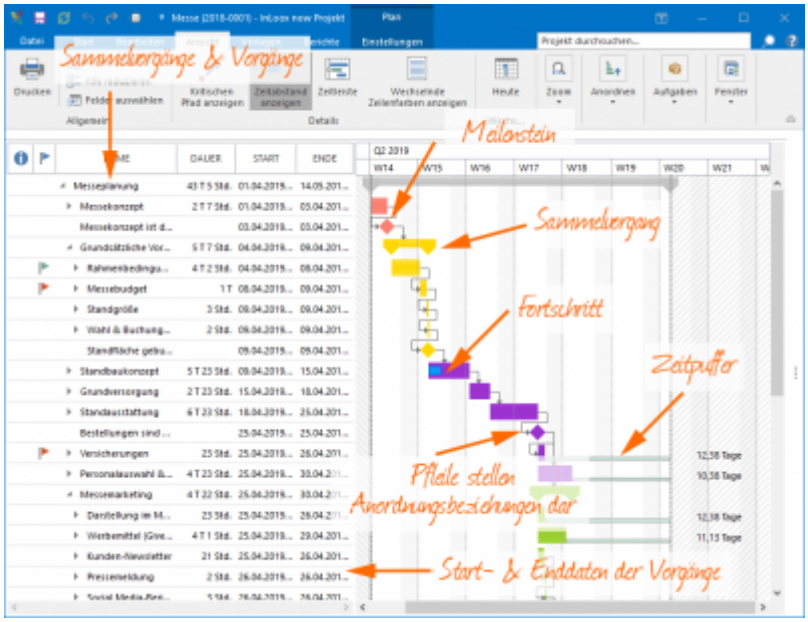
Vorlage in Excel

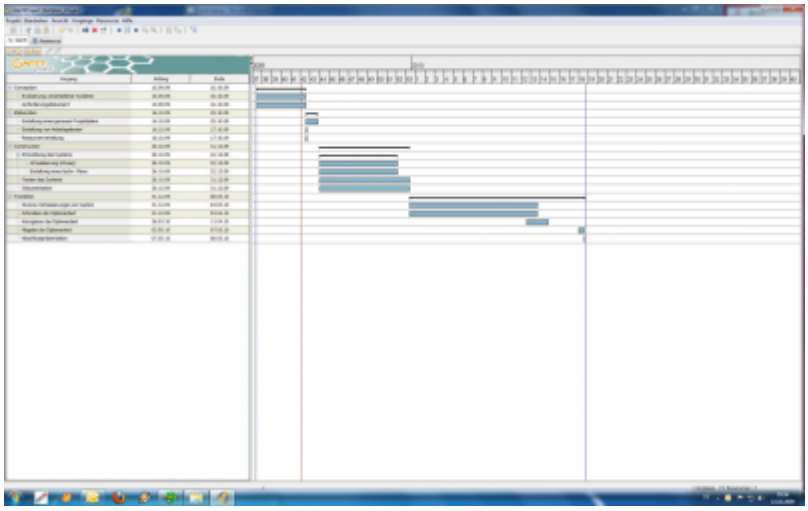
Vorlage in Word
Lastenheft Beispiel

Projektplan

Projektplan (oder Projektmanagementplan) ist ein Begriff aus dem Projektmanagement und hält das Resultat sämtlicher Planungsaktivitäten in einem konsistenten Dokument oder mehreren kohärenten Dokumenten fest.







Design/Entwurf

Die Design-Phase dient der Ausarbeitung eines konkreten Lösungskonzepts auf Basis der zuvor ermittelten Anforderungen, Aufgaben und Strategien. Software-Entwickler erarbeiten in dieser Phase die **Software-Architektur** sowie einen detaillierten **Bauplan der Software** und konzentrieren sich dabei auf konkrete Komponenten wie Schnittstellen, Frameworks oder Bibliotheken. Das Ergebnis der Design-Phase umfasst ein Entwurfsdokument mit Software-Bauplan sowie Testplänen für einzelne Komponenten.

Pflichtenheft

Das Pflichtenheft beschreibt in konkreter Form, wie der Auftragnehmer die Anforderungen des Auftraggebers zu lösen gedenkt – das sogenannte wie und womit. Der Auftraggeber beschreibt vorher im Lastenheft möglichst präzise die Gesamtheit der Forderungen – was er entwickelt oder produziert haben möchte. Erst wenn der Auftraggeber das Pflichtenheft akzeptiert, sollte die eigentliche Umsetzungsarbeit beim Auftragnehmer beginnen.

Pflichtenheft Vorlage

Pflichtenheft RoomBA

Pflichtenheft Airhockey

Pflichtenheft AlicePro

Pflichtenheft Brettspiele

Pflichtenheft Syntax Tool

Pflichtenheft Cluedo

Projekthandbuch

Das Projekthandbuch beinhaltet alle einzuhaltenden Regeln und relevanten Informationen, die für die erfolgreiche Durchführung des Projekts relevant sind. Es ist somit Regelwerk und Übersicht zugleich. Aber was gehört eigentlich in so ein Projekthandbuch? Warum ist es so wichtig? Und von wem wird es überhaupt erstellt? Fragen, die wir für Sie in diesem Artikel beantworten.

Ein Projekthandbuch ist projektspezifisch

Laut DIN 69901-5:2009 ist das Projekthandbuch ein projektspezifisches Informationswerk zur Durchführung des Projekts. Jedes Projekt bekommt sein eigenes.

Ohne Handbuch kein Projekt

Dann fehlen Transparenz, Kontinuität und Reporting – so können Sie kein Projekt managen.

Der Standard verpflichtet

Egal für welche Richtlinie Sie sich entscheiden, der verwendete Standard gibt Ihnen die Mindestinhalte vor, die Sie nicht in Frage stellen sollten.

Die Projektgröße verlangt Anpassungen

Je nach Projektart und dem Umfang des Vorhabens können Sie selbst entscheiden, welche der erforderlichen Aspekte Sie wie ausführlich handhaben.

In der Kürze liegt die Würze

Es wird kein Roman verlangt! Das Projekthandbuch soll so knapp wie möglich und nur so ausführlich wie nötig formuliert sein.

Projekthandbuch Vorlage

Projekthandbuch Beispiel

Implementierung

Realisiert wird die in der Design-Phase konzipierte Software-Architektur in der **Implementierungsphase**, die **Software-Programmierung, Fehlersuche und Modultests** umfasst. In der Implementierungsphase wird der Software-Entwurf in der gewünschten Programmiersprache umgesetzt. Einzelne Komponenten werden separat entwickelt, im Rahmen von Modultest überprüft und Schritt für Schritt in das Gesamtprodukt integriert. Das Ergebnis der

Implementierungsphase ist ein Software-Produkt, das in der nachfolgenden Phase zum ersten Mal als Gesamtprodukt getestet wird (Alpha-Test).

Test/Überprüfung

Die Testphase beinhaltet die Integration der Software in die gewünschte Zielumgebung. In der Regel werden Software-Produkte zunächst als Beta-Version an ausgewählte Endbenutzer ausgeliefert (Beta-Tests). Ob die Software die zuvor definierten Anforderungen erfüllt, lässt sich mithilfe der in der Analysephase entwickelten Akzeptanztests ermitteln. Ein Software-Produkt, das Beta-Tests erfolgreich absolviert hat, ist bereit für den Release.

Betrieb/Wartung

Nach erfolgreichem Abschluss der Testphase wird die Software für den Betrieb im Produktiveinsatz freigegeben. Die letzte Phase des Wasserfallmodells schließt Auslieferung, Wartung und Verbesserung der Software ein.

Vor- und Nachteile

Vorteile	**Nachteile**
Einfache Struktur durch klar abgegrenzte Projektphasen	Komplexe oder mehrschichtige Projekte lassen sich nur selten in klar abgegrenzte Projektphasen unterteilen
Gute Dokumentation des Entwicklungsprozesses durch klar definierte Meilensteine	Geringer Spielraum für Anpassungen des Projektablaufs aufgrund veränderter Anforderungen
Kosten und Arbeitsaufwand lassen sich bereits bei Projektbeginn abschätzen	Der Endanwender wird erst nach der Programmierung in den Produktionsprozess eingebunden
Projekte die nach dem Wasserfallmodell strukturiert werden, lassen sich auf der Zeitachse gut abbilden.	Fehler werden mitunter erst am Ende des Entwicklungsprozesses erkannt.

Fazit

Wasserfallmodelle nutzt man vor allem bei Projekten, bei denen sich Anforderung und Abläufe bereits in der Planungsphase präzise beschreiben lassen und bei denen davon auszugehen ist, dass sich die Vorannahmen während des Projektablaufs höchstens geringfügig ändern. Streng lineare Vorgehensmodelle eigenen sich somit in erster Line für kleine, einfache und klar strukturierte Software-Projekte.

Weitere Vorlagen und Dokumente

Meetingprotokoll Vorlage

Anforderungsliste Vorlage

Scrum

Scrum ist die verbreitetste Form des agilen Projektmanagements. Der Begriff Scrum entspringt dem Rugby und steht für „Gedränge“. In einem Projekt nach Scrum zählt die enge und selbstorganisierte Zusammenarbeit eines interdisziplinären Teams. Das Team arbeitet in sogenannten Sprints zusammen. Sprints sind zeitlich festgelegte „Mini-Projekte“, die einen Schritt (Inkrement) in der Produktentwicklung liefern. Zu einem Sprint gehört z. B. die Planung, Entwicklung und Retrospektive sowie kleinere Daily Scrums (tägliche Meetings). Sprints werden analysiert, bewertet und liefern wichtige Zwischenergebnisse mit neuen Erkenntnissen für die nächsten Sprints. Auf diese Weise kommt im Scrum das finale Produkt immer näher an den Kundenwunsch heran.

Einen klassischen Projektmanager wie im traditionellen Projektmanagement gibt es in Scrum nicht. Die Rollenverteilung sieht wie folgt aus:

- **Product Owner**: Vertritt Kundeninteressen und wirtschaftlichen Erfolg des Projektes. Klärt die Anforderungen an das Produkt in einem Anforderungskatalog (Product Backlog), der eine Prioritätenliste der Kundenwünsche beinhaltet.
- **Scrum Master**: Kümmert sich um die reibungslose Koordination, Planung und Durchführung des Scrum. Dient als Ansprechpartner für Product Owner und Team. Kann auch als Coach fungieren.
- **Scrum Team**: Eine Gruppe von Personen, die sich selbst managen, um die Anforderungen an das Produkt zu erfüllen und Produktinkremente zu liefern.