Einführung in die Programmierung

Prof. Dr. Rudolf Berrendorf

Fachbereich Informatik
Hochschule Bonn-Rhein-Sieg

rudolf.berrendorf@h-brs.de
http://berrendorf.inf.h-brs.de/

Formales zur Veranstaltung (1)

- Vorlesung V2
  - Vorstellung von neuem Stoff
- Übung Ü3
  - 1 Stunde zentrale Übung / 2 Stunden dezentral in Übungsgruppen

- Wöchentliche Hausaufgaben (Details gleich)
- Monatliche Testklausuren zur Selbstüberprüfung (Sie korrigieren selber)
- Wöchentliche Tutorien ab Anfang November
- Literaturhinweise zu den einzelnen Themen über die Webseite (s.u.)

- Alle Unterlagen zur Veranstaltung im Netz:
  http://berrendorf.inf.h-brs.de/
  Menüpunkt Lehrveranstaltungen / Einführung in die Programmierung

Formales zur Veranstaltung (2)

- Abschluss: zweistündige schriftliche Klausur
- Credits werden vergeben, wenn mindestens 50% der Punkte in der schriftlichen Prüfung erreicht werden (siehe Ankündigung)

- **Voraussetzung** zur Teilnahme an der Klausur: mind. 50% der möglichen Punkte in jeder Kategorie (s.u.) der Übungsaufgaben in der letzten Veranstaltung vor der jeweiligen Klausur (siehe Ankündigung).
- Oder anders ausgedrückt: Übungspunkte behalten nur maximal ein Jahr ihre Gültigkeit

- Beispiel: Sie erlangen im WS2015/2016 die nötigen Übungspunkte. Dann können Sie an den Prüfungen im Frühjahr und Herbst 2016 teilnehmen, aber nicht mehr im Frühjahr 2017.

Formales zur Veranstaltung (3)

- Übungspunkte gibt es in 2 Kategorien
  - **Programmieraufgaben**: sind entsprechend auf dem Aufgabenblatt markiert, durch elektronische Einreichung einer individuellen Lösung. Details dazu später.
  - **Präsenzaufgaben**: jeweils ein kurzer Test in einer Übungsstunde in der **eigenen** Übungsgruppe. Dabei muss eine Aufgabe angelehnt an Aufgaben der Hausaufgaben seit dem letzten Test ohne weitere Hilfsmittel (also auch ohne die eigene Unterlagen) individuell gelöst werden.
- Es werden keine Teilpunkte verteilt.

Start und Ziel

- Wo holen wir Sie ab:
  - Wir holen Sie an dem Wissensstand der Fachhochschulreife ab. Nicht mehr und nicht weniger.
  - Keine Vorkenntnisse über Informatik notwendig!
- Wo bringen wir Sie dieses Semester hin:
  - Grundprinzipien der Programmierung „im Kleinen“
  - Grundkenntnisse der Programmiersprache Java
  - Einfache Algorithmen
  - Nachfolgende Semester setzen auf diesem Stoff auf (keine Wiederholung)
  - Zitat Prof. Bürsner aus einem Evaluationsbericht :
    „Bemerkenswert ist aus meiner Sicht die weite Schere zwischen positiven und negativen Kommentaren in der Pflichtveranstaltung Software Engineering II. Eine maßgebliche Ursache für diese Schere ist nach meiner Einschätzung im hohen Anteil an Studierenden begründet, die einen erheblichen Nachholbedarf in Java-Programmierung, aber auch in anderen für das Software Engineering wichtigen Grundlagen aus der Kerninformatik aufweisen.“
- Der Weg:
  - wird beschwerlich sein…
  - **Nur Sie** (weil es so wichtig ist nochmals: **nur Sie**) bringen sich zum Ziel
  - Wir helfen Ihnen nach Kräften

Was wir von Ihnen erwarten…

- Nur durch den praktischen Umgang mit Stoff lernt man ihn.
- Die Stoffmenge und der Fortschritt wird für Sie sehr ungewohnt und herausfordernd sein, ist aber an einer Hochschule normal.
- Die Konsequenz ist, dass Sie **immer** „am Ball“ bleiben müssen!
- Eine oft in Schuljahren angewandte Strategie „x Tage vor der Klausur anfangen zu lernen“ ist eine Strategie, die definitiv nicht funktionieren wird.
- Der durchschnittliche **zusätzliche** Zeitaufwand für diese Veranstaltung beträgt laut Credit-Rechnung 5-6 (Zeit-)Stunden pro Woche.

- Das heißt konkret für ihre Wochenplanung:
  - Sie reservieren einen angemessenen Zeitslot, in dem Sie den Stoff der Vorlesung nacharbeiten (bis Sie ihn verstehen; mindestens 2 Stunden)
  - Später (fester Zeitslot; ca. 4 Stunden) lösen Sie die Hausaufgaben. Eine kleine Arbeitsgruppe ist eine sehr gute Idee.
  - Sie beteiligen sich aktiv in den Übungsgruppen
  - Sie bearbeiten zusätzliche Übungsaufgaben nach ihrer Wahl

Informatik

- Informatik ist die Wissenschaft von der systematischen Verarbeitung von Informationen.

- Informatik wird unterteilt in drei große Teilbereiche:
  - Theoretische Informatik (Berechenbarkeitstheorie, Komplexitätstheorie,…)
  - Praktische Informatik (Programmierung, Datenbanken, Betriebssysteme,…)
  - Technische Informatik (Rechnerarchitekturen, Netzarchitekturen,…)

Software-Entwicklung bei uns (Bachelor Inf.)

Pflichtprogramm Software-Entwicklung

- Einführung in die Programmierung, Java
- komplexere Datenstrukturen und Algorithmen
- Programmierung „im Großen“, Entwicklungsmethodik, Software-Architekturen
- Projektmanagement, Requirements Engineering, Usability

Pflichtprogramm aus Modulgruppe

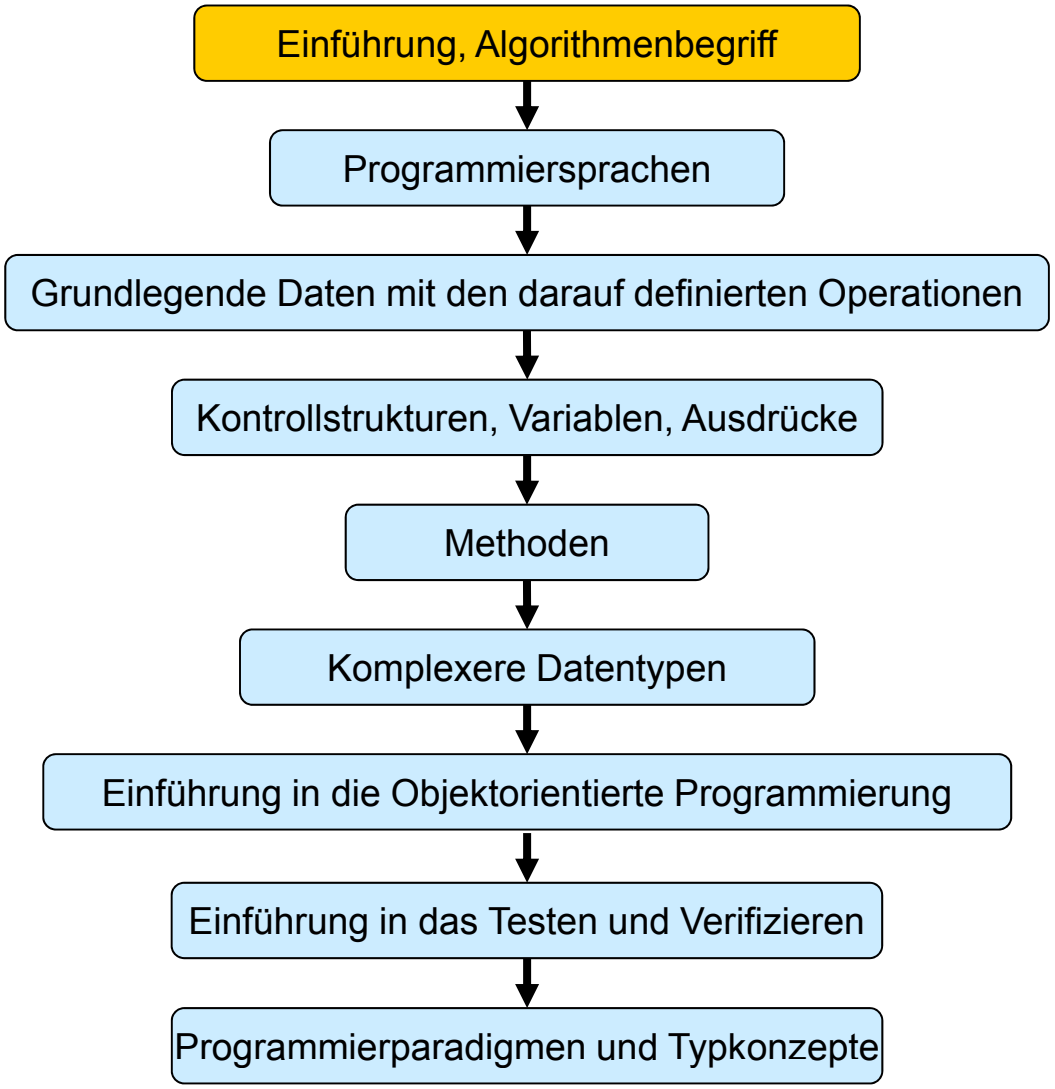
- Programmieren in C
- Datenbank-systeme
- Informations-sicherheit

Ggfs. Spezialisierung Komplexe Software-Systeme

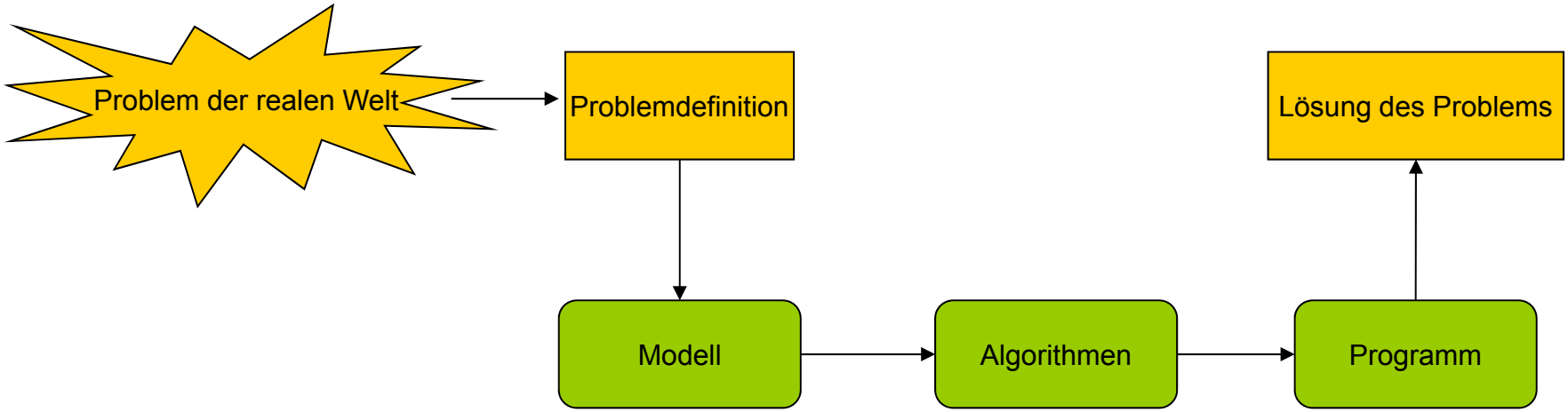
- Einführung in Web Engineering
- Programmierung paralleler Anwendungen
- Web Usability
- Objektrelationale Datenbanken
- Seminar

zusätzlich Wahlpflichtfächer im 4./5. Semester zu speziellen Themen der Software-Entwicklung, Seminar, Bachelor Thesis

Inhalt dieser Veranstaltung



Problemlösungsprozess



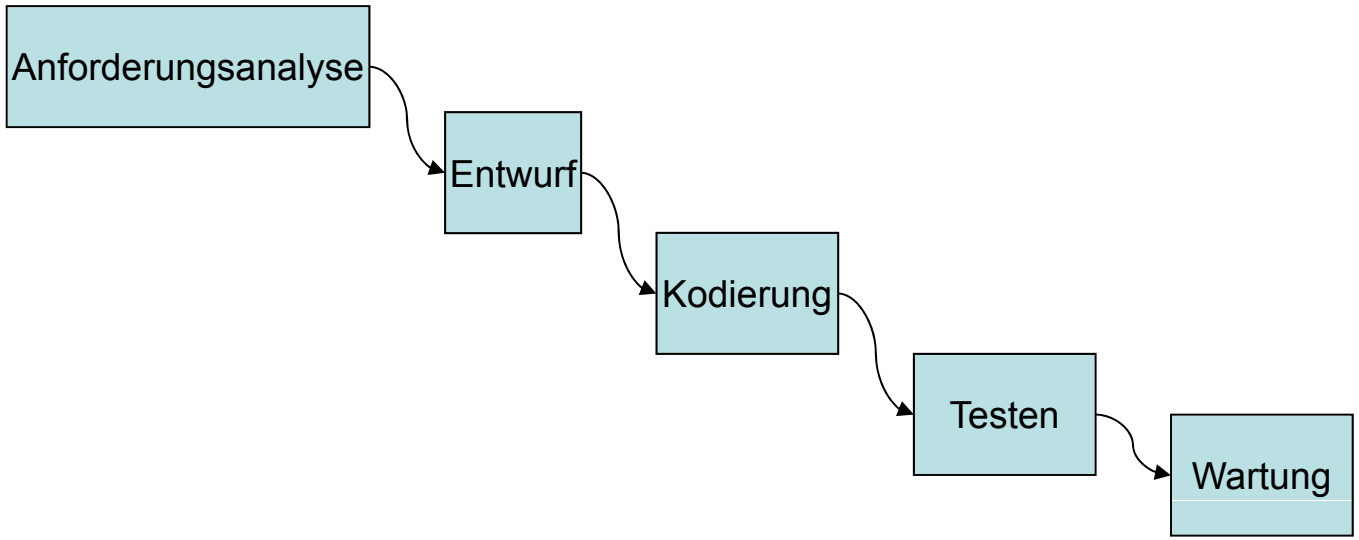
- Frage: wie kommt man (insbesondere bei komplexen Problemstellungen) systematisch von einem Problem zu einer Lösung dieses Problems?
- Antwort für große Probleme: Phasen- und Prozessmodelle

Phasen- und Prozessmodelle

- Phasenmodelle teilen den Entwicklungsprozess in einzelne Phasen mit bestimmten Aktivitäten auf. Die Phasen werden in einer bestimmten Reihenfolge ausgeführt und liefern jeweils bestimmte Ergebnisse (Dokumente, Modelle, Software,…)

- In einem Prozessmodell wird u.a. die Reihenfolge der Phasen festgelegt
- Einfachste denkbare Reihenfolge: Phase i+1 kann erst beginnen, wenn Phase i abgeschlossen ist (Wasserfallmodell)

Beispiel für Phasen in einem Wasserfallmodell:



Phasen der klassischen Software-Entwicklung

- Anforderungsanalyse:
  Fachliche Anforderungen ermitteln, Testfälle beschreiben
- Entwurf:
  Aus den Anforderungen wird ein (grobes) Software-Modell entworfen mit genau spezifizierten Schnittstellen zwischen einzelnen Teilen
- Kodierung / Programmieren:
  Basierend auf dem vorgegebenen Software-Modell werden Lösungen für Teilprobleme erstellt, die exakt die Schnittstellen des Software-Modells einhalten
- Testen:
  Unter Nutzung der Testfälle wird überprüft, ob die Software-Lösung das fachliche Problem löst
- Wartung:
  Durch veränderte Randbedingungen oder aufgetretene Fehler müssen Änderungen an der Software vorgenommen werden

Zwischenstand

- Phasen- und Prozessmodelle dienen dazu, bei komplexen Problemen systematisch vom Ursprungsproblem zu einer Lösung des Problems zu kommen
- Es gibt viele verschiedene solcher Modelle

Reflektion

Beschreiben Sie ihrem Nachbarn mit eigenen Worten die Aufgaben der Phasen in der klassischen Software-Entwicklung.

Algorithmusbegriff

- Ein Algorithmus ist eine Arbeitsanweisung zur systematischen und schrittweisen Lösung einer Klasse von Problemen.

- Die einzelnen Schritte in einem Algorithmus sind damit klar definiert und eindeutig

- Algorithmen sind möglichst auf eine ganze Klasse ähnlicher Problemstellungen anwendbar.
  Beispiel: Sortieren von Matrikelnummern, Klausurnoten, Namen, …

- Algorithmen können auf unterschiedliche Weise formuliert sein (umgangssprachlich, als Programm, als Diagramm,…; gleich mehr dazu)

Beispiel 1

- Problemstellung: berechne den größten gemeinsamen Teiler zweier natürlicher Zahlen x,y, wobei nicht beide Zahlen gleich 0 sein dürfen

- Lösung seit > 2000 Jahren bekannt (Euklid):
- Formulierung 1:

```
ggT(x,y):
Falls x = 0 ist, dann ist y das Ergebnis
ansonsten
    wiederhole, solange y ≠ 0 gilt
        falls x > y ersetze x durch x - y
        ansonsten   ersetze y durch y - x
    x ist das Ergebnis
```

- Formulierung 2:

$$ggT(x,y) := \begin{cases} y & falls\ x = 0 \\ x & falls\ y = 0 \\ ggT(x-y,y) & falls\ x > y \\ ggT(x,y-x) & sonst \end{cases}$$

- Beispiel:
ggT(16,10)=ggT(6,10)=ggT(6,4)=ggT(2,4)=ggT(2,2)=ggT(2,0)=2

Diskussion

```
ggT(x,y):
Falls x = 0 ist, dann ist y das Ergebnis
ansonsten
    wiederhole, solange y ≠ 0 gilt
        falls x > y ersetze x durch x - y
        ansonsten   ersetze y durch y - x
    x ist das Ergebnis
```

$$ggT(x,y) := \begin{cases} y & falls\ x = 0 \\ x & falls\ y = 0 \\ ggT(x-y,y) & falls\ x > y \\ ggT(x,y-x) & sonst \end{cases}$$

- Nutzung von Variablen x und y, die Stellvertreter für konkrete Werte sind, mit denen der Algorithmus genutzt wird (vergleiche Funktionen in der Mathematik)

- Der Algorithmus enthält Anweisungen
  - zur Veränderung von Werten (Beispiel: `ersetze x durch x-y`)
  - mit denen die Reihenfolge der Ausführung dieser Einzelanweisungen beeinflusst wird (`wiederhole, solange…`)

Beispiel 2

- Problemstellung: wie alt ist die älteste Person in diesem Raum?

- Eine Lösung:
  - Wähle eine beliebige Person aus und frage nach dem Alter. Dieses Alter ist das höchste Alter aller bisher befragten Personen.
  - Wiederhole, solange es noch ungefragte Personen in dem Raum gibt:
    - Frage eine Person nach dem Alter
    - Falls dieses Alter höher als das bisher bestimmte höchste Alter ist, so ist dieses Alter das neue höchste Alter.

- In unserer schönen heilen Algorithmuswelt ignorieren wir dabei an dieser Stelle, dass eine Person ihr Alter nicht nennen will, die mögliche zeitliche Varianz (Personen können aus dem Raum gehen / hereinkommen),…

Beispiel 3

Auszug aus einer Online-Meldung:

Google will mit Software Mitarbeiter-Abwanderung stoppen

Der Suchmaschinen-Spezialist Google will einer drohenden Abwanderung wichtiger Mitarbeiter zuvorkommen – mit Hilfe einer speziellen Software. Das Unternehmen sammle derzeit Daten von Mitarbeitern wie Ausbildungsstand und Gehaltsentwicklung, berichtet das *Wall Street Journal* heute. Mit einem speziellen Algorithmus solle aus der Datenmenge ermittelt werden, welche der 20.000 Mitarbeiter sich zum Beispiel unterfordert fühlen könnten und deshalb möglicherweise offen für neue Herausforderungen sind. Der Algorithmus helfe Google über Wechselwillige Bescheid zu wissen, bevor die Betroffenen selbst wüssten, dass sie wechseln wollen, zitiert die Zeitung Googles Personalchef Laszlo Bock.
…

http://www.heise.de/newsticker/Google-will-mit-Software-Mitarbeiter-Abwanderung-stoppen--/meldung/138130

Algorithmische Grundbausteine

- In Mathematik, Maschinenbau, Medizin, Jura,… gibt es oft wiederkehrende fachspezifische Muster, die als Bausteine zur Lösung komplizierter Zusammenhänge genutzt werden
- Teilweise existieren sogar eigene Symbole für solche Muster

- Einfache Beispiele aus der Mathematik:
  - Summe von n Zahlen: $\sum$
  - Produkt von n Zahlen: $\prod$
  - unbestimmtes Integral einer Funktion f(x): $\int f(x)\,dx$

- Beim Entwurf von Algorithmen gibt es ebenfalls solche Grundbausteine
- Alle gängigen Programmiersprachen unterstützen solche einfachen Entwurfsmuster direkt durch entsprechende Sprachkonstrukte
- Im Folgenden noch keine Java-Schreibweise!

- Zentraler Begriff in diesem Zusammenhang: Anweisung

Einzelanweisung

- Ein Einzelanweisung `A` ist eine Anweisung

- Wir legen hier nicht fest, was genau eine Einzelanweisung sein kann. Das kann jede Anweisung sein (z.B. umgangssprachlich notiert), zu der allen Beteiligten die Bedeutung unzweifelsfrei klar ist.

- Beispiele:
  - Ersetze `x` durch `y` (Andere Notationen dafür: `x=y;` `x:=y;` `x←y`)
  - Frage nach dem Alter
  - `x` ist das Ergebnis
  - Man nehme 200 g Mehl
  - Morgens eine halbe Tablette nehmen

Sequenz

- Hat man beliebige Anweisungen A_1 bis A_n, so kann man diese zu einer Sequenz zusammenfassen

  $A_1 ; \ldots ; A_n$

- Eine Sequenz ist wieder eine Anweisung
- Die Bedeutung einer Sequenz ist, dass die einzelnen Anweisungen A_1 bis A_n in genau dieser Reihenfolge hintereinander ausgeführt werden müssen

- Beispiel:
  - Frage nach dem Alter; ersetze x durch den erfragten Wert

Selektion

- Hat man eine Bedingung `B` (ein logischer Ausdruck, der zu wahr oder falsch ausgerechnet werden kann) und eine Anweisung `A`, so kann man damit eine Selektion formulieren

  ```
  if B then A
  ```

- Auch eine Variante davon ist möglich

  ```
  if B then A1 else A2
  ```

- Eine Selektion ist wieder eine Anweisung
- Die Bedeutung einer Selektion ist, dass die Bedingung `B` ausgewertet wird. Anschließend wird die Anweisung `A` nur dann ausgeführt wird, wenn die Bedingung wahr war (Variante: A_1 wird ausgeführt, wenn die Bedingung wahr ist, ansonsten wird A_2 ausgeführt)

- Beispiele:
  - **if** x>y **then** ersetze x durch y
  - **if** x<y **then** ersetze y durch x

Mehrfachselektion

- Falls man einen Wert $x \in \{x_1, \ldots, x_n\}$ hat und n Anweisungen $A_1, \ldots, A_n$, so wird eine Mehrfachselektion angegeben durch

  switch x: **case** x_1:A_1; **case** x_2:A_2; … **case** x_n:A_n; **end switch**

- Eine Mehrfachselektion ist wieder eine Anweisung
- Die Bedeutung einer Mehrfachselektion ist wie folgt:
  - Zuerst wird x ausgewertet.
  - Falls $x = x_i$ für ein $i \in \{1, \ldots, n\}$, wird die Anweisung A_i ausgeführt, womit dann die gesamte Anweisung beendet ist.
  - Falls $x \neq x_i \ \forall \ i=1, \ldots, n$, so ist die Anweisung beendet.

- Beispiel:

```
switch Handyhersteller:
      case Nokia:      nehme Batterietyp 1;
      case Samsung:    nehme Batterietyp 2;
      case HTC:        nehme Batterietyp 1;
end switch
```

Iteration

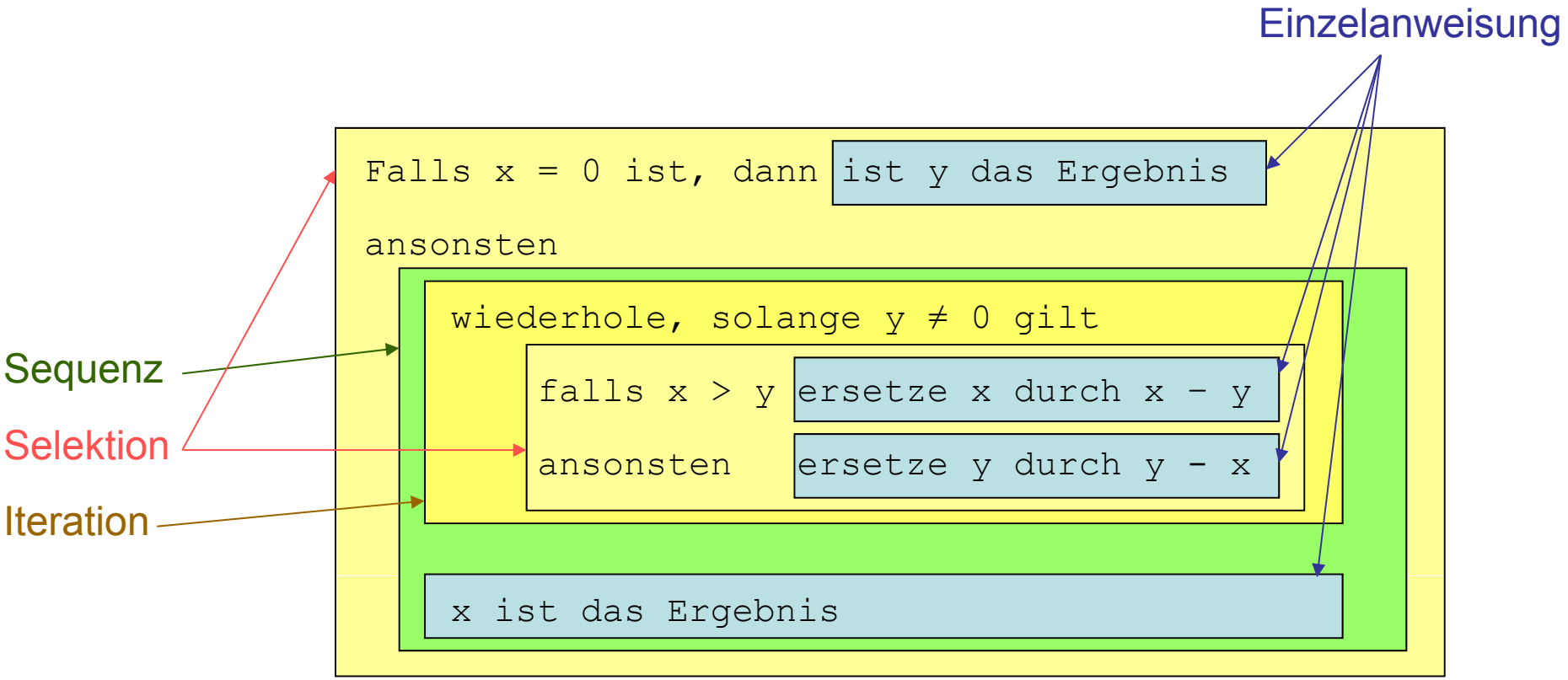
- Hat man eine Anweisung `A` und eine Bedingung `B`, so kann man damit eine Iteration formulieren:

  `while` `B` **`do`** `A`

- Eine Iteration ist wieder eine Anweisung
- Die Bedeutung einer Iteration ist wie folgt. Zuerst wird die Bedingung `B` ausgewertet. Ist der Wert von `B` falsch, so ist die Ausführung der Anweisung damit beendet. Ansonsten (`B` ist wahr) wird die Anweisung `A` ausgeführt und anschließend diese Ausführungsschritte wiederholt.

- Beispiele:
  - **while** ungefragte Personen im Raum sind **do**
    frage nach dem Alter
  - **while** y ≠ 0 **do**
    if x > y **then** ersetze x durch x – y
    else ersetze y durch y - x

Struktur des Beispiels



Zwischenstand

- Ein Algorithmus ist eine Arbeitsanweisung zur Lösung eines (kleinen) Problems
- Aus der Praxis heraus haben sich Programmiermuster entwickelt, die oft wiederkehrende Bausteine beim Programmieren darstellen
- „Anweisung“ ist der Fachbegriff für solch einen Baustein
- Die hier betrachteten Programmiermuster kann man in allen gängigen Programmiersprachen finden/verwenden

Reflektion

- Diskutieren Sie mit ihrem Nachbarn die Vorteile von Mustern/Bausteinen im Maschinenbau und danach die Vorteile beim Programmieren.
- Was würde es bedeuten, wenn es solche Muster nicht geben würde?

Darstellungsmöglichkeiten für Algorithmen

- Sehr, sehr kleine Problemlösung (Einzeiler; für uns weniger interessant):
  - Umgangssprachlich
  - Mathematische Formel
  - Einfache Skizze
  - …
- Überschaubare Problemlösung: Reihe von Beschreibungsmöglichkeiten (das ist unser Thema dieses Semester)
  - UML Aktivitätsdiagramm
  - Struktogramm
  - Programm
- Große bis sehr große Problemlösung: andere Möglichkeiten wie zum Beispiel UML (Unified Modeling Language) (spätere Semester)

Beispielproblem zur Erinnerung

- Problemstellung: berechne den größten gemeinsamen Teiler zweier natürlicher Zahlen x,y, wobei nicht beide Zahlen gleich 0 sein dürfen

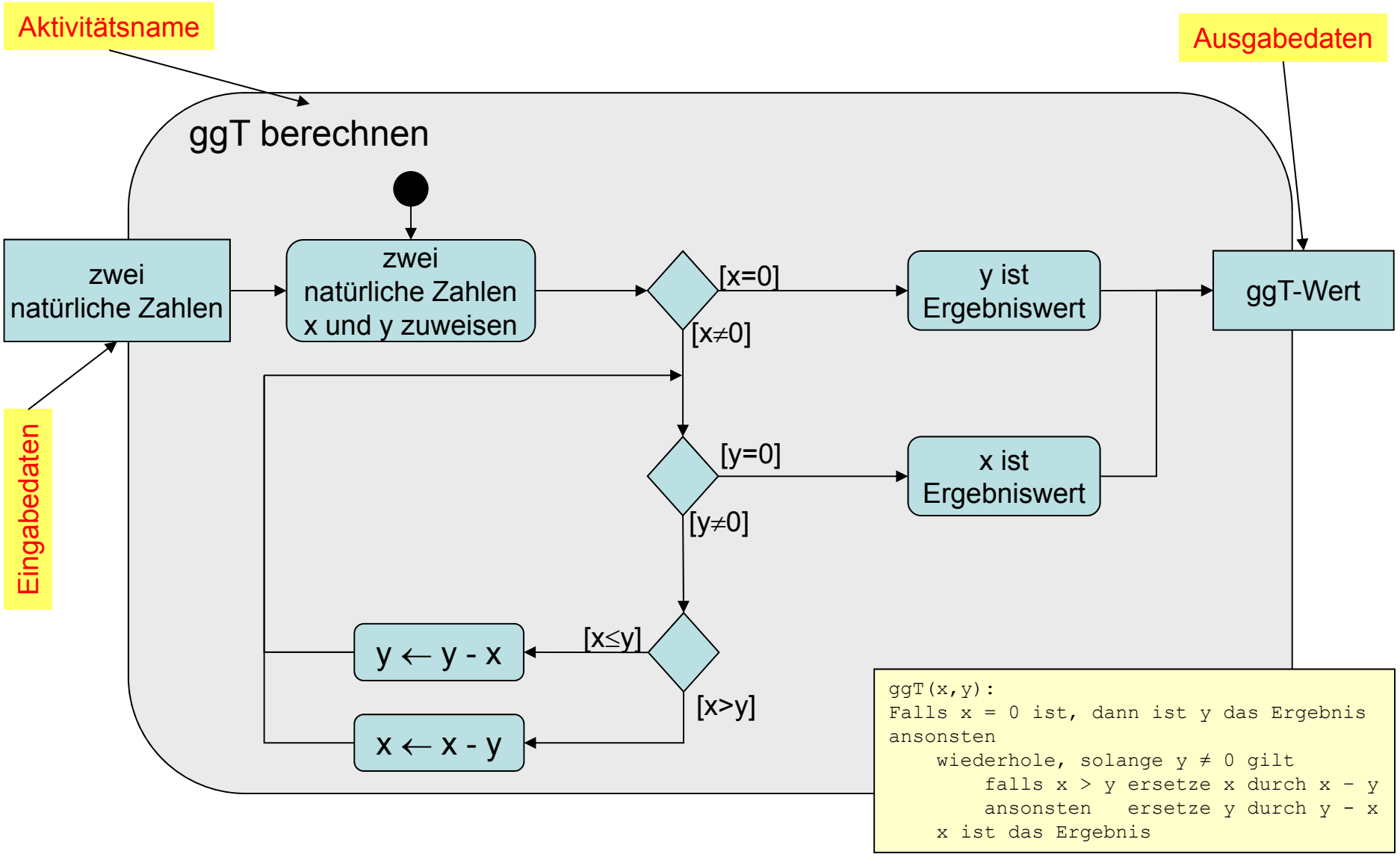
- Lösung 1:

```
ggT(x,y):
Falls x = 0 ist, dann ist y das Ergebnis
ansonsten
    wiederhole, solange y ≠ 0 gilt
        falls x > y ersetze x durch x – y
        ansonsten   ersetze y durch y - x
    x ist das Ergebnis
```

UML Aktivitätsdiagramm

- UML: Unified Modeling Language (sehr umfangreich; später mehr dazu)
- Diagrammtypen zur Veranschaulichung von Zusammenhängen aus unterschiedlichen Sichten
- Hier interessant: Aktivitätsdiagramme der UML
- Ähnlich den Programmablaufplänen (PAP) / Flussdiagrammen
- Grafische Darstellung (Zeichnung)
- Ein Aktivitätsdiagramm besteht aus Aktionselementen, die durch Pfeile miteinander verbunden werden können
- Pfeile geben mögliche Wege durch den Algorithmus an
- Eine Berechnung ist das Durchlaufen eines Aktivitätsdiagramms von einer Start- zu einer Stop-Markierung, wobei alle Aktionen auf dem durchlaufenden Weg ausgeführt werden
- Nur der hier relevante Teil von Aktivitätsdiagrammen wird beschrieben
- Nachteil von Aktivitätsdiagrammen:
  - schnell unübersichtlich
  - nicht alle erkannten Programmiermuster sind direkt darstellbar (Iteration)

Beispiel ggT



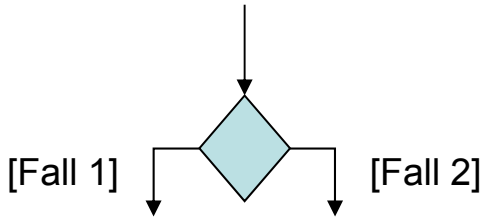
Auswahl an grafischen Elementen

Start

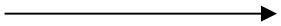
Stop

einzelne Aktion

Daten-/Objektsammlung



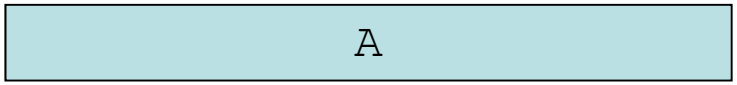
Entscheidungsknoten
(auch mehr als 2 Fälle möglich)



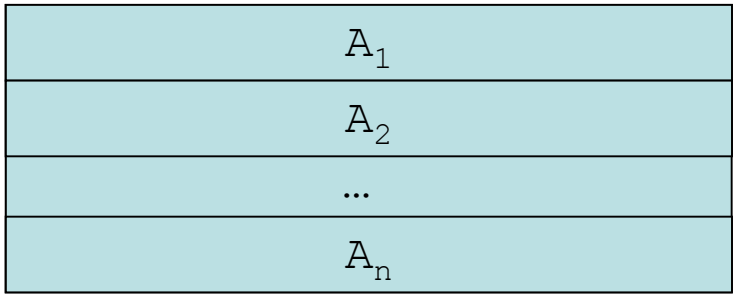
möglicher
Kontrollfluss

Struktogramm (1)

- Die Elemente eines Struktogramms entsprechen den identifizierten Programmiermustern
- Struktogramme spielen in der Praxis keine Rolle, helfen aber sehr gut beim Einstieg in Programmiermuster / Programmstrukturen
- Einzelanweisung `A`

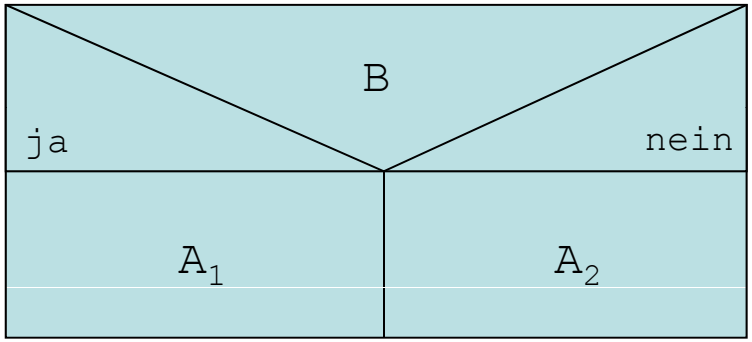


- Sequenz $A_1;\ldots;A_n$



- Selektion

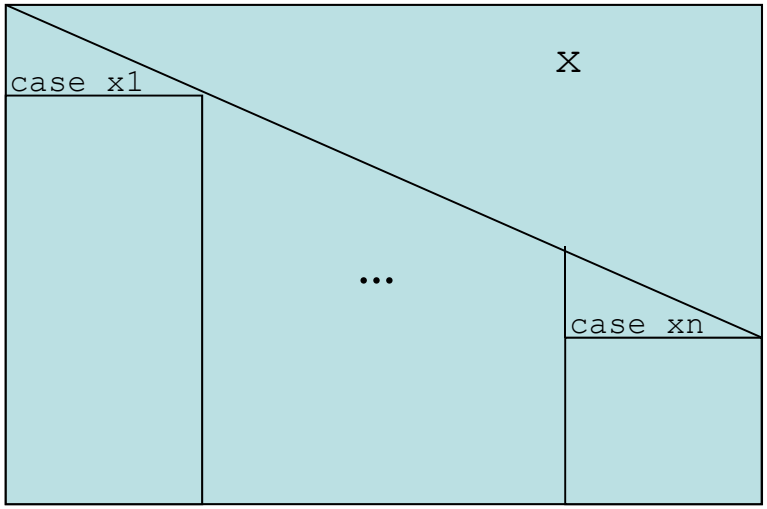
  `if` `B` **`then`** A_1 **`else`** A_2



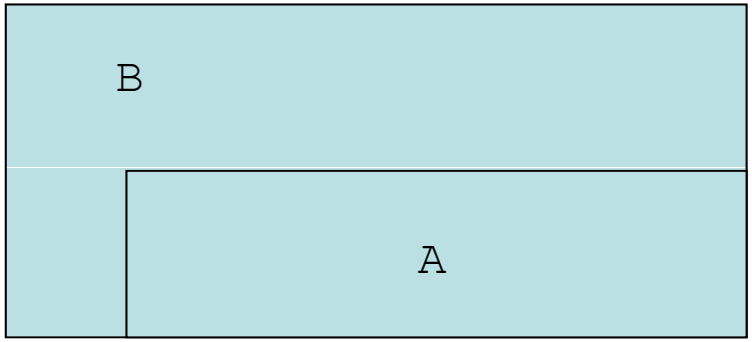
Struktogramm (2)

- Mehrfachselektion

  switch x: **case** x_1:A_1; **case** x_2:A_2; … **case** x_n:A_n; **end switch**

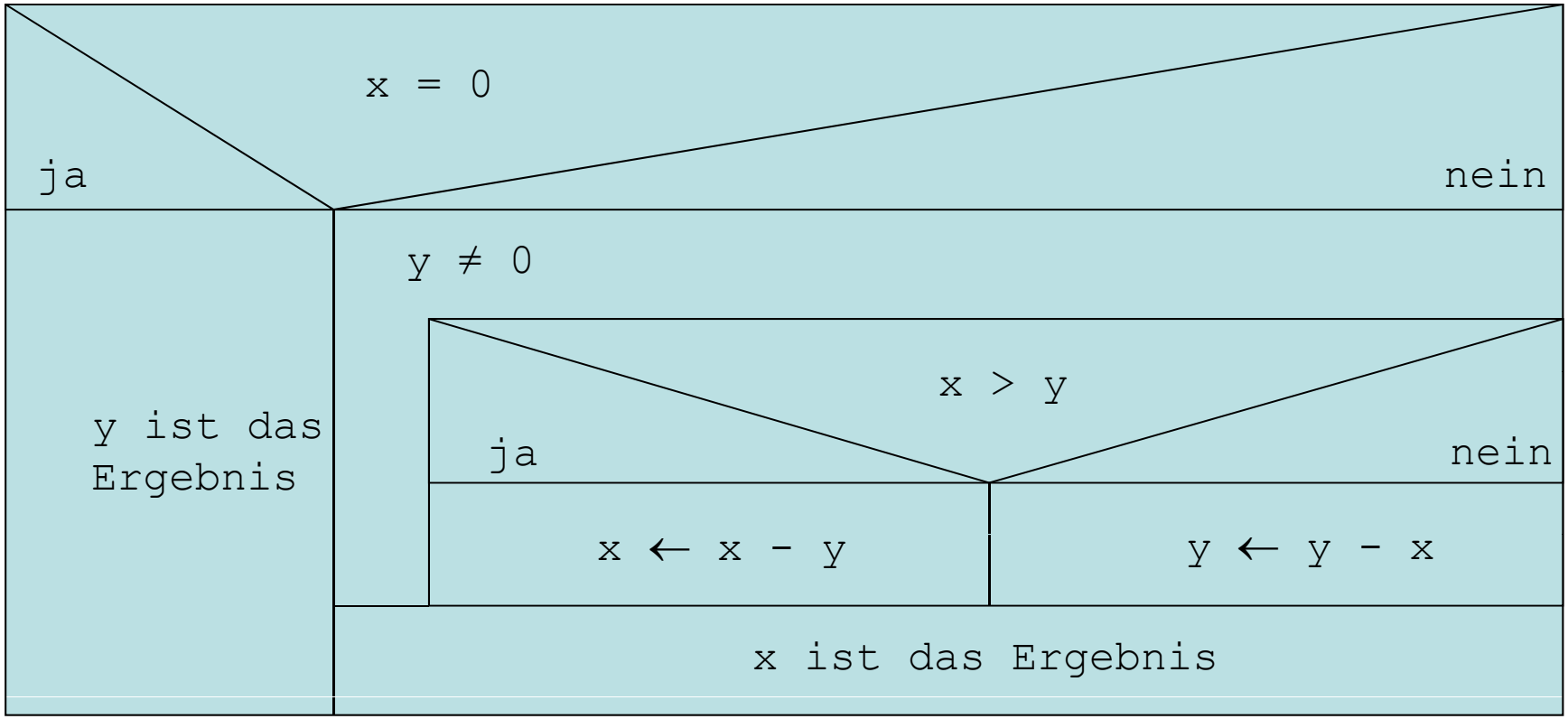


- Iteration **while** B **do** A



Beispiel

```
ggT(x,y):
Falls x = 0 ist, dann ist y das Ergebnis
ansonsten
    wiederhole, solange y ≠ 0 gilt
        falls x > y ersetze x durch x - y
        ansonsten   ersetze y durch y - x
    x ist das Ergebnis
```



Programm

- Aktivitätsdiagramm und Struktogramm sind als Dokumentationswerkzeuge für überschaubare Algorithmen im Wesentlichen für den Menschen gedacht
- Ein Rechner könnte diese Darstellungsmöglichkeiten (es sind ja eigentlich Zeichnungen) nur sehr schwer verstehen
- Zur Kommunikation mit dem Rechner gibt es besser geeignete Formulierungsmöglichkeiten für Algorithmen (Programmiersprachen), mit denen auch sehr umfangreiche und sehr komplexe Algorithmen formuliert werden können

- Wer den letzten Punkt anzweifelt: stellen Sie die derzeit ca. 11 Millionen Zeilen Programmcode für den Linux-Kernel oder die ca. 50 Millionen Zeilen für Microsoft Windows als Struktogramm oder Programmablaufplan (PAP; Flussdiagramm) dar

- Alle Details zu Programmen und Programmiersprachen folgen später

Beispiel

```
/**
 * groesster gemeinsamer Teiler ggT
 */
public class ggT  {
    public static void main (String[] args)  {
        // Beispielwerte
        int x = 43158;
        int y = 26364;

        // gebe die Werte von x und y aus
        System.out.print ("Der ggT von " + x + " und " + y + " ist ");

        if(x == 0) {
            // gebe das Ergebnis / den ggT aus
            System.out.println (y);
        } else {
            while (y != 0)  {
                if (x > y) {
                    x = x - y;
                } else {
                    y = y - x;
                }
            }
            // das Verfahren brach ab und in x steht das Resultat
            System.out.println (x);
        }
    }
}
```

Zwischenstand

- Algorithmen lassen auf verschiedene Weise angeben
- Die UML (Unified Modelling Language) besitzt Aktivitätsdiagramme, in denen im Wesentlichen die Reihenfolge von durchzuführenden Aktivitäten grafisch dargestellt wird
- Ein Struktogramm eignet sich sehr gut, um die geschachtelte Struktur von Anweisungen darzustellen (Programmiermuster)
- Ein Programm ist nötig, um einen Algorithmus auf einem Rechner ausführen zu können

Reflektion

Diskutieren Sie mit ihrem Nachbarn, wieso UML Aktivitätsdiagramme und Struktogramme Vorteile für einen menschlichen Betrachter haben (und falls Sie es schon wissen oder einschätzen können, weshalb diese Darstellungsformen eigentlich nicht so gut für einen Computer sind)!

Zusammenfassung

- Das große Bild der Software-Entwicklung:
  - Phasen- und Prozessmodelle sind für größere Projekte unentbehrlich (→ nachfolgende Veranstaltungen zum Software-Engineering)
  - In dieser Veranstaltung beschäftigen wir uns aber ausschließlich mit überschaubaren Problemstellungen, bei denen diese Instrumente zu komplex sind

- Algorithmusbegriff
- Häufig wiederkehrende Entwurfsmuster in Algorithmen
- Darstellungsmöglichkeiten: Aktivitätsdiagramm, Struktogramm, Programm