# Grundlagen der Informatik

## - Compiler, Linker, Interpreter und Debugger -

Prof. Dr. Klaus Volbert


Hochschule für angewandte Wissenschaften
Fakultät Informatik und Mathematik

Wintersemester 2010/11
Regensburg, 09. November 2010

# *Typische Phasen der Software-Entwicklung*

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

1. **Auftragsphase**
   - Anforderungen (teilweise unpräzise)
   - Vorstellungen/Wünsche über das zu entwickelnde System
   - Erstellung von Lastenheften
2. **Analysephase**
   - Anforderungen (möglichst präzise)
   - Konkretisierung der Problemstellung aus den Vorstellungen
   - Erstellung von Pflichtenheften
3. **Entwurfsphase**
   - Strukturierung und genaue Beschreibung der Funktionalitäten durch die Darstellung der Struktur und insbesondere des Verhaltens
   - Erstellung einer System- und Programmspezifikation
4. **Implementierungsphase**
   - Programmierung und Test der spezifizierten Komponenten
5. **Betriebsphase**
   - Pflege der erstellten Software, ggf. Beginn wieder bei 1 (SW-Zyklus)

# *Vorgehensmodelle*

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- …sind Leitfäden für die Software-Entwicklung, die
  - grundlegende Arbeitsschritte für den Entwickler festlegen
  - Hilfestellungen für die Arbeit eines Entwicklers geben
  - keine „Zwangsjacke“ darstellen sollten (Akzeptanz)
  - vorgeben sollten, in welcher Reihenfolge was zu tun ist (nicht wie!)
  - die Qualität einer Software nachweisbar erhöhen
- Beispiele
  - Trial-and-Error
  - Spiralmodell
  - Wasserfallmodell
  - V-Modell
  - Unified Software Development Process
  - Agile Softwareentwicklung
    - SCRUM
    - Extreme Programming (XP)

# Software-Architektur

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Wesentliche architektonische Richtlinien einer Software:
  - Aufteilung in einzelne Komponenten
  - Definition von schlanken, geeigneten und möglichst unabhängigen Schnittstellen zwischen den Komponenten
  - Entkopplung unterschiedlicher Aspekte, z.B. die Aspekte der Benutzeroberfläche von der eigentlichen Fachlogik des Programms und den Aspekten der Datenhaltung (Drei-Schichten-Architektur), zur Verbesserung der Erweiterbarkeit und Wartbarkeit
  - Verwendung von Entwurfsmustern (Design-Pattern) zwecks Wiederverwendbarkeit von Komponenten und Vermeidung von Fehlern, die schon in ähnlichen Programmen gemacht wurden
- Software wird heutzutage idealerweise objekt-orientiert in der Unified Modeling Language (UML) modelliert
  - Implementierungen der Modellierung in unterschiedlichsten, auch nicht objekt-orientierten Programmiersprachen sind möglich

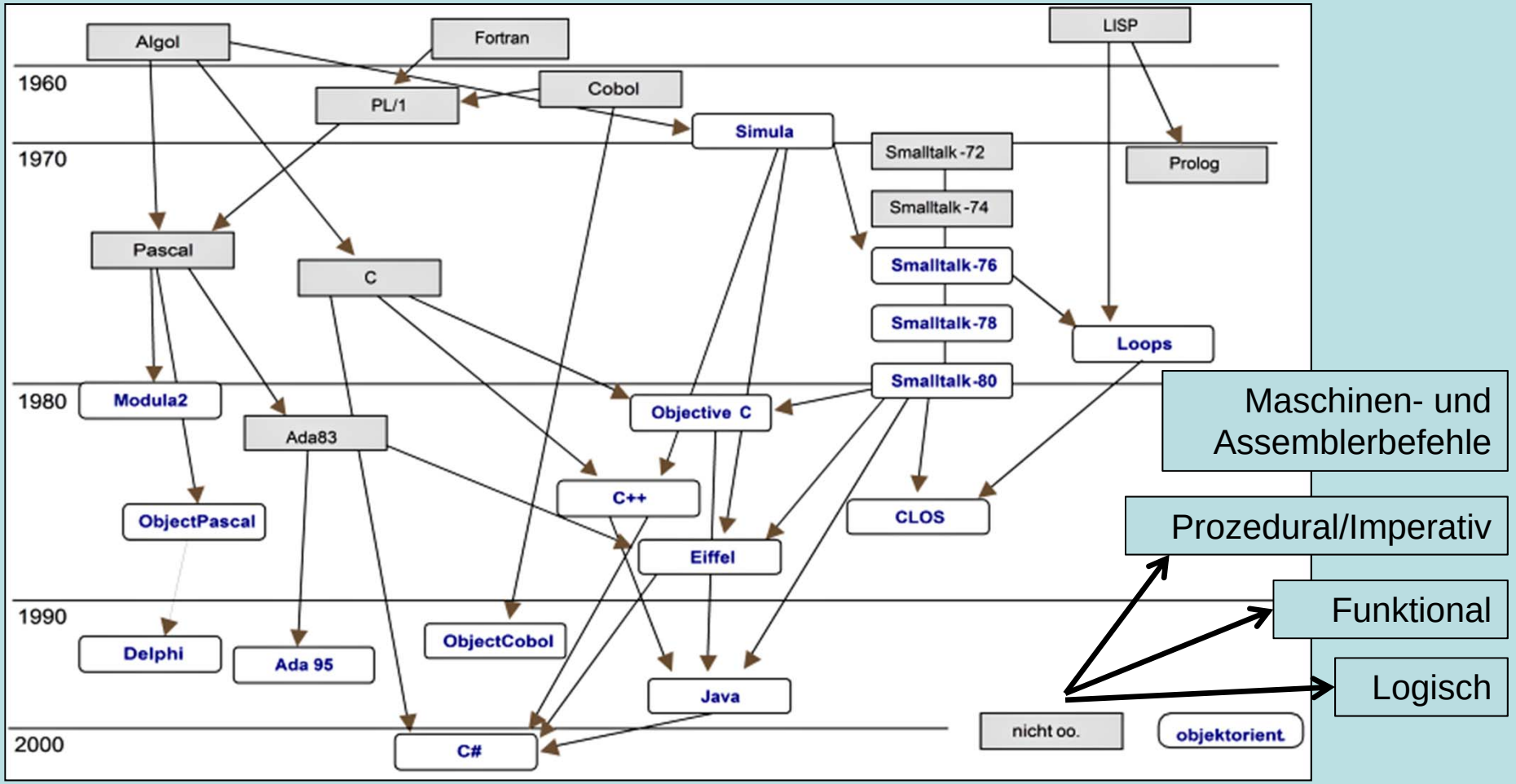
# *Programmieren im Kleinen/Großen*

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Die Entwicklung von kleinen, aber durchaus (mathematisch) komplexen Algorithmen und Kontrollstrukturen in einer Software wird auch als Programmieren im Kleinen bezeichnet
  - Im Mittelpunkt stehen die Algorithmen (monolithisch)
- Die Entwicklung einer Software-Architektur, d.h. die Modellierung einer Software hinsichtlich Struktur und Verhalten auf abstrakter Ebene mit vielen Komponenten wird auch als Programmieren im Großen bezeichnet
  - Im Mittelpunkt steht die Aufteilung eines Programms in viele kleinere Programmeinheiten (viele verschiedene Quelltext-Module)
  - Voraussetzung: Die kleineren Programmeinheiten sollten ihre Existenzberechtigung haben und nicht „zu klein werden“

# *Entwicklung der Programmiersprachen*

- Wesentliche Entwicklungsidee:
  - Vereinfachung der Software-Entwicklung durch Abstraktion („möglichst weit weg“ von der Maschinensprache ohne Verluste!)



# *Werkzeuge der Software-Entwicklung*

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- **Modellieren**
  - Abbildung eines Realitätsausschnitts, um eine Aufgabe mit Hilfe eines Computerprogramms zu lösen; z.B. mit grafischen Hilfsmitteln in Form von Diagrammen oder mathematisch exakt zur detaillierten Beschreibung von strukturellen und/oder dynamischen Eigenschaften
- **Programmieren**
  - Umsetzung von spezifizierten Abläufen in einer konkreten Programmiersprache (Erstellung einer Software)
- **Compilieren**
  - Übersetzung jedes Befehls eines Programms in einer höheren Programmiersprache in eine entsprechende Folge von Maschinenbefehlen für eine konkrete Hardware-Architektur
  - Ergebnis ist ein schwer lesbares, sehr spezielles Maschinenprogramm
- **Interpretieren**
  - Nicht alle Befehle des in der höheren Programmiersprache geschriebenen Programms werden in Maschinensprache übersetzt, sondern erst bei Ausführung zur Laufzeit interpretiert

# *Modellierung*

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Modellierung umgangssprachlich formulierter Sachverhalte
  - Mathematische Darstellung (Funktionale Spezifikation)
  - Repräsentation durch formale Sprachen
- Modellierung der zu erstellenden Software
  - Darstellung der Struktur und des Verhaltens durch Diagramme
  - Verwendung von Architekturmodellen und Ablaufplänen
- Modellierung durch die Verwendung von Datenstrukturen
  - Stapel, Liste, Baum, Graph
- Beispiel:
  - Eingabe: Modellierung umgangssprachlich formulierter Anforderungen
  - Ausgabe: Generierung von Quelltext-Rahmen ggf. inklusive der modellierten Abläufe in einer konkreten Programmiersprache (z.B. C)
- Quelltext-Generator
  (hilfreich zur Vermeidung einer manuellen Übersetzung)

# *Compiler*

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

**Eingabe**:
Quellprogramm

↓

Scanner: Lexikalische Analyse
(liefert eine Folge von Grundsymbolen)

↓

Parser: Syntaktische Analyse
(liefert einen Struktur-/Syntaxbaum)

↓

Semantische Analyse (liefert attributierten Syntaxbaum) und maschinenabhängige Optimierung

↓

Codeerzeugung und -optimierung

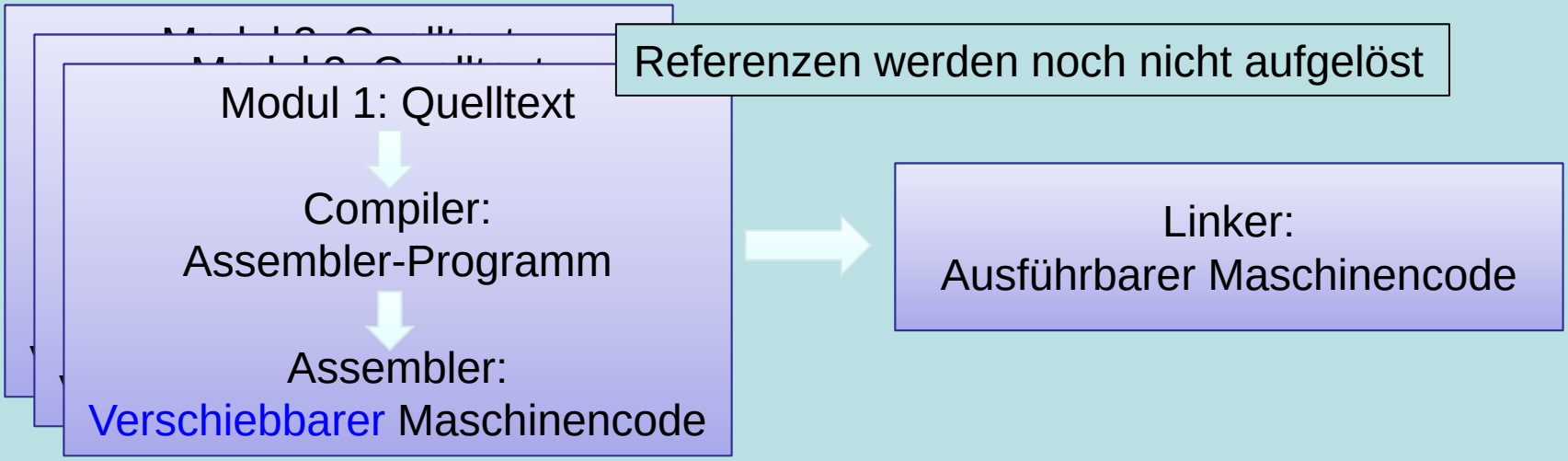
↓

**Ausgabe**:
Programm in Maschinensprache

# Linker

- Architektonischer Grundgedanke:
  Aufteilung in einzelne Komponenten (Module)

  ⇒ Getrennte Compilierung, d.h. unterschiedliche Module (verschiedene Quelltext-Dateien) können zu unterschiedlichen Zeitpunkten von verschiedenen SW-Entwicklern unabhängig voneinander compiliert werden
- Compilierung erzeugt zunächst Objektdateien
- Linker baut die Objektdateien zu einem ablauffähigen Programm durch verbinden („linken“) zusammen:



# *Arten des Linkens*

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Statisches Linken
  - Alle benötigten Funktionen werden zusammengebunden (insbesondere Bibliotheksfunktionen)
  - Vorteil: Beim statischen Linken ist sichergestellt, dass die benötigten Funktionen auch vorhanden sind (ansonsten führt das Linken zu einem Fehler)
  - Nachteil: Die Zieldatei kann sehr groß werden und widerspricht der architektonischen Komponentenrichtlinie (Monolithische Struktur)
- Dynamisches Linken
  - Nicht alle Funktionen werden zusammengebunden, sondern das Identifizieren der notwendigen Funktion erfolgt erst bei Verwendung, d.h. sobald die Funktion zur Laufzeit des Programms aufgerufen wird
  - Nachteil: Sollte eine benötigte Funktion zur Laufzeit nicht vorhanden sein, so muss der Programmablauf ggf. unterbrochen werden
  - Vorteile: Gemeinsame Speichernutzung, Austauschbarkeit
- Beispiele
  - Dynamically Linked Libraries (DLLs), Shared Libraries

# Lader

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Linker produziert Maschinencode
- Maschinencode kann prinzipiell an jede Stelle in den Arbeitsspeicher geladen werden
- Bei Verwendung kann der Maschinencode z.B. an Adresse 0x4000 geladen werden
- Der Lader (engl. loader) sorgt nun dafür, dass zu allen Adressen im Programm der Wert 0x4000 addiert wird:
  - Befehl für Befehl wird eingelesen und der Maschinencode jeweils einschließlich des Offsets an die Stelle im Hauptspeicher eingetragen, die als Speicheradresse vor jedem Befehl bzw. beim Programmstart angegeben wurde
  - Anschließend kann das Programm ausgeführt werden
- Nach dem Rechnerstart wird der Lader meist auf absolut vorgegebenen Speicheradressen geladen (Bootstrap)

# *Debugger*

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Ein wesentliches Hilfsmittel/Programmierwerkzeug zur Überprüfung der Programmierung (Qualitätssicherung)
- Die Übersetzung gibt die ursprüngliche Bedeutung wieder: Entwanzer (zur Zeit der Röhrenrechner traten Fehler durch Kurzschlüsse auf, die angeblich von Käfern erzeugt wurden)
- Ein Debugger erlaubt das schrittweise Durchlaufen eines Programms inklusive der Überprüfung des aktuellen Zustands (Arbeitsspeicher, Register, Variablen, Quelltext, Assemblercode, etc.)
- Wesentliche Funktionen eines Debuggers:
  - Run — Ausführung starten
  - Step — Nächste Anweisung im Assemblerprogramm ausführen
  - Next — Nächste Anweisung im Programm ausführen
  - Cont — Ausführung bis zum nächsten Haltepunkt ausführen
  - Undo — Letzte Debugger-Steueranweisung wieder rückgängig