Es gibt mehr als C und Java – Programmierpradigmen im Vergleich

Univ.-Prof. Dr. Dr. Günther Specht

Datenbanken und Informationssysteme
Institut für Informatik
Universität Innsbruck

Talk-IT, 26. November 2oo8

1

Prof. Dr. Dr.
Günther Specht

guenther.specht@uibk.ac.at

- 1999 - 2000 Prof. an der TU München
- 2000 - 2001 Prof. an der TU Ilmenau
- 2001 - 2006 Prof. an der Uni Ulm
- seit 2006: Prof. an der Uni Innsbruck

Hauptarbeitsgebiete:

- Datenbanken und Informationssysteme
- Multimedia DBS
- Mobile DBS
- Genom Datenbanken

2

Lehrstuhl Datenbanken und Informationssysteme
Institut für Informatik
Universität Innsbruck

http://dbis-informatik.uibk.ac.at

3

Es gibt mehr als C und Java …

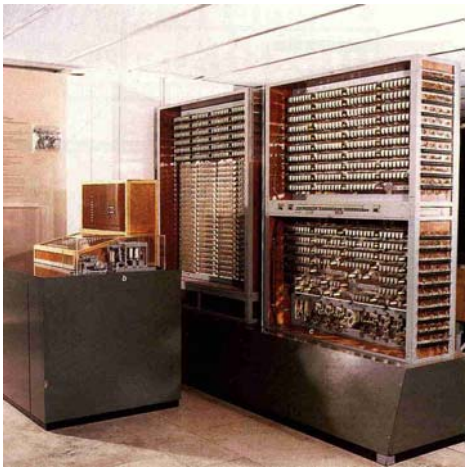
(Neuere) Programmierparadigmen im Vergleich

4

Im Anfang war das Wort

5

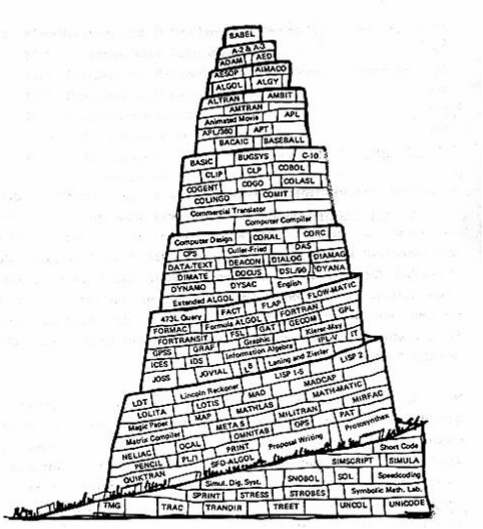
… und das Wort hatte 2 Byte



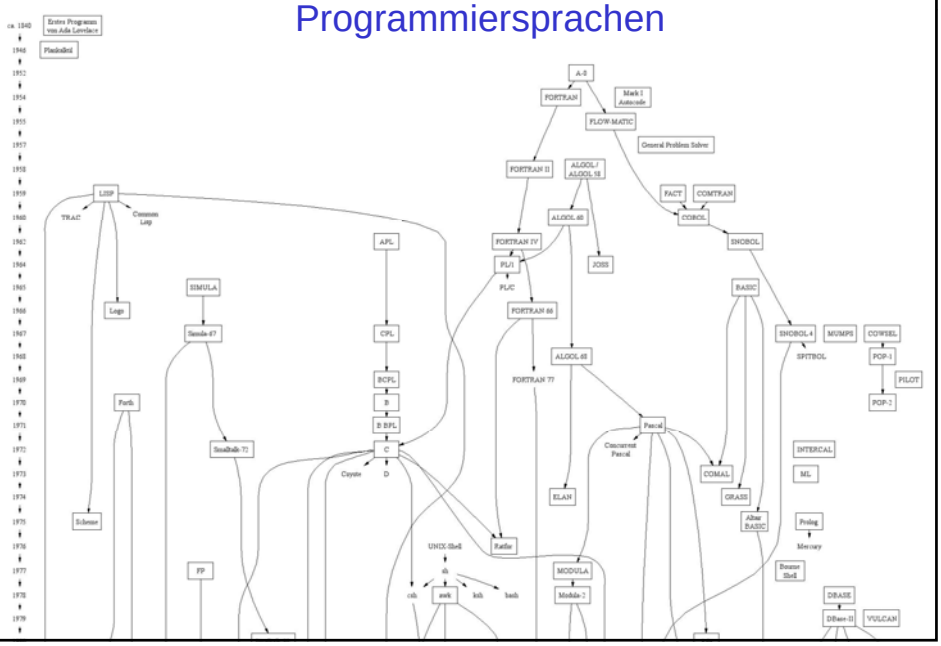
6

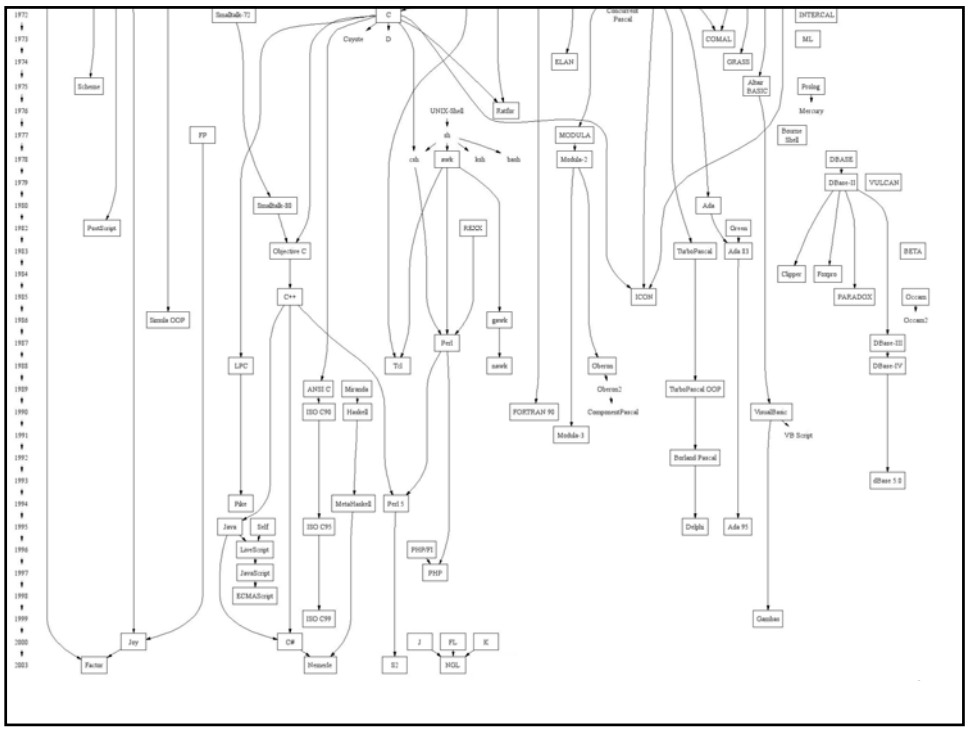
Zuse Z3, 1945

Babylonisches Sprachengewirr jetzt auch in der Informatik ?

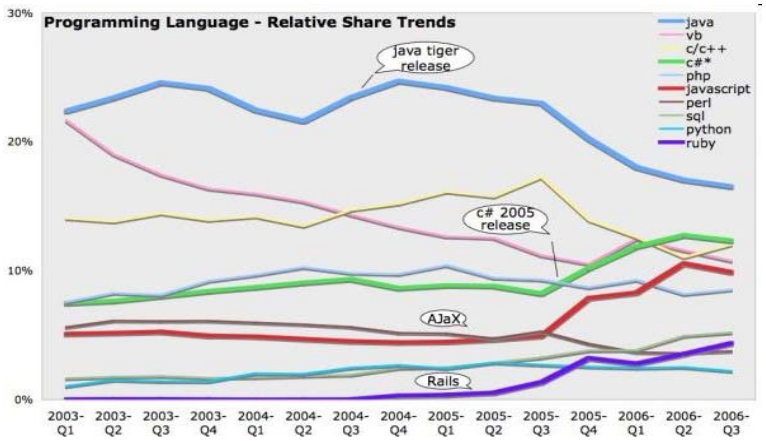


Geschichtliche Entwicklung der Programmiersprachen





Programmiersprachen Trends



10

11

Besser: bringen wir etwas Ordnung ins System

Geht das auch mit Programmiersprachen?

12

Mittels

Programmiersprachen-Paradigmen

13

Maschinenorientierte Sprachen

Bsp.: Assembler, Intel 8086

```
.MODEL Smal l
.STACK 100h
.DATA
HW DB ' He l l o World ! $ '
.CODE
s t a r t :
MOV AX, @data
MOV DS,AX
MOV DX, OFFSET HW
MOV AH, 09H
INT 21H
MOV AH, 4Ch
INT 21H
end s t a r t

// gibt Hello World aus
```

14

Maschinenorientierte Programmierung

- **Sprachen:**
  - **Assembler**
- **Stärken:**
  - **effizient**
  - **maschinennah**
- **Schwächen:**
  - **keine problemorientierte Programmierung**
  - **schwer lesbar**
  - **nicht wartbar**
- **Einsatz:**
  - **Zielcode der Compiler höherer Sprachen**
  - **Microprogrammierung, Firmwareprogrammierung**

15

Prozedurale Sprachen

Bsp.: C

```
static int tm_diff(struct tm *a, struct tm *b)
    {
       int ay = a->tm_year + (TM_YEAR_BASE - 1);
       int by = b->tm_year + (TM_YEAR_BASE - 1);
       int intervening_leap_days = (ay/4 - by/4) - (ay/100 - by/100) + (ay/400 - by/400);
       int years = ay - by;
       int days = 365*years + intervening_leap_days + (a->tm_yday - b->tm_yday);
       int hours = 24*days + (a->tm_hour - b->tm_hour);
       int minutes = 60*hours + (a->tm_min - b->tm_min);
       int seconds = 60*minutes + (a->tm_sec - b->tm_sec);

       return seconds;
    }
```

16

Prozedurale Programmierung

- **Sprachen:**
  - **Cobol, Fortran, Algol**
  - **Pascal, C**
- **Konzept / Stärken:**
  - **Programmieren durch sequentielle Befehlsfolgen, insbesondere durch Variablenzuweisungen**
  - **strukturiertes Programmieren**
  - **Typ-Konzept**
  - **schnelle Numerik**
  - **meist iterativer Programmierstil**
- **Schwächen:**
  - **“Ärger” mit Pointer (selbst referenzierend, Pointer zeigt ins Leere)**
  - **Seiteneffekte möglich (z.B. auf globale Variable)**
  - **kein rapid prototyping**
- **Einsatz:**
  - **Numerik, Systemprogrammierung**
  - **algorithmische Probleme**
  - **konventionelle IT**

17

Funktionale Sprachen

Bsp.: OCaml

```
let rec fak n =
        if n=0 then 1
        else n*fak (n-1);;

let laenge liste = List.fold_left (fun n x -> n+1) 0 liste;;

laenge [4;6;7];;

int = 3
```

18

Funktionale Programmierung

- **Sprachen:**
  - **Lisp, Scheme, OCaml**
- **Theoretische Fundierung:**
  - **Lambda-Kalkül**
- **Konzept / Stärken:**
  - **Programmieren durch Funktionsaufrufe und Parameterübergabe**
  - **keine Prozeduren und damit keine Seiteneffekte**
  - **Keine Variablen**
  - **rekursiver Programmierstil**
  - **Datenfluß gut sichtbar**
  - **uniforme Behandlung von Daten und Programmen**
  - **Funktionen höherer Ordnung: Funktionen können als Parameter übergeben werden**
  - **Semantik: call-by-value, left-most-inner-most Auswertung**
  - **Korrektheitsbeweise leicht möglich**
- **Schwächen:**
  - **Klammergebirge (--> Verwendung von Struktur-Editoren)**
  - **keine Variable, sondern nur Parameter**
- **Einsatz:**
  - **Symbol- und Listenverarbeitung**
  - **KI**

19

Logische Sprachen

Bsp.: Prolog

```
sterblich(X) :- mensch(X).
mensch(sokrates).

parent(john,paul).
parent(paul,tom).
parent(tom,mary).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).

append( [ ], L, L).
append( [ K | R ], L2, [ K | L3 ]) :- append( R, L2, L3).
```

20

Logische (regelorientierte) Programmierung

- **Sprachen:**
  - **Prolog, Datalog**
- **Theoretische Fundierung:**
  - **Prädikatenlogik 1. Stufe**
  - **Fixpunktsemantik (kleinstes Herbrandmodell)**
- **Konzept / Stärken:**
  - **deklarativ**
  - **Programm besteht aus Regeln und Fakten, ("was" soll berechnet werden, nicht "wie")**
  - **einfache Syntax**
  - **keine Kontollstrukturen**
  - **gute Verständlichkeit, da Trennung von Ablaufkontrolle und Problemwissen**
  - **rapid prototyping**
  - **Theorembeweise**
- **Schwächen:**
  - **keine Kontollstrukturen**
  - **Auswertung kann ohne Optimierungen ineffizient werden**
  - **(oft ineffiziente Arithmetik)**
  - **prozedurales Wissen schwer darstellbar**
- **Einsatz:**
  - **Analyse und Diagnosesysteme**
  - **Theorembeweiser**
  - **KI**

21

Objektorientierte Sprachen

Bsp.: Java

```
import com.sun.image.codec.jpeg.*;
import java.awt.image.*;
import java.io.*;

/* create BufferedImage with data to save */
Dimension size = getSize();
BufferedImage image = (BufferedImage)someJComponent.createImage(size.width, size.height);

 Graphics g = image.getGraphics();

someJComponent.paint(g);

try {  /* open a file */
       FileOutputStream out = new FileOutputStream(someFile);

       /* create JPEG encoder and use it to encode & write to the file */
       JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
       encoder.encode(image);

       out.flush(); out.close();
}
catch (IOException ioe) {
       /* handle exception */
}
catch(RasterFormatException rfe) {
 /* handle exception */
}
```

22

Objektorientierte Programmierung

- **Sprachen:**
  - **Smalltalk, Simula, C++, Java**
- **Theoretische Fundierung:**
  - **Abstrakte Datentypen,**
  - **Polymorphismus**
  - **Vererbung**
- **Stärken:**
  - **hierarchisches Klassenkonzept mit Vererbung**
  - **gute Strukturierung und Modularisierung**
  - **Overriding, Overloading,**
  - **late binding**
  - **Objektidentität statt Werte-Identität**
- **Schwächen:**
  - **Navigieren im Objektnetz nötig**
  - **Lange Einarbeitung in umfangreiche Klassenbibliotheken**
- **Einsatz:**
  - **Heute übliche Programmierung**

23

Skript-Sprachen

Bsp.: PHP

```
<?php

    $conn = mysql_connect("mysql_host", "mysql_user", "mysql_password");
    mysql_select_db("Meine_Datenbank",$conn);

    $query = "SELECT * FROM Benutzer";
    $result = mysql_query($query);

    echo "<table>";
    while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
            echo "<tr><td>".$row["name"]."</td></tr>";
    }
    echo "</table>";
    ?>
```

24

Skript-Programmierung

- **Sprachen:**
  - **Shell Skripte**
  - **Ruby, PHP, Python**
- **Stärken:**
  - **Keine oder sehr kurze Compile-Debug-Zyklen**
  - **Meist aufbauend auf OO**
  - **Nicht dogmatisch, eher pragmatisch**
  - **Einfache APIs**
  - **Anwendungsprogrammierung (für einfache GUIs)**
- **Schwächen:**
  - **Keine Compile-Checks**
  - **Performanz (daher manche: JIT-Compiler)**
- **Einsatz:**
  - **Skripte, Workfows**
  - **GUIs**
  - **Web-Programmierung**

25

Aspektorientierte Sprachen

Bsp.: AspectJ

```
aspect SimpleTracing
{
        pointcut tracedCall():
            call(void FigureElement.draw(GraphicsContext));

        before(): tracedCall()
        {
            System.out.println("Entering: " + thisJoinPoint);
        }
}

// ein Trace der sich in alle Aufrufe von draw hängt
// und zwar vor den Aufruf von draw
```

26

Aspektorientierte Programmierung

- **Sprachen:**
  - **AspectJ, MS Research: Aspect.NET**
  - **(in vielen Sprachen, die nicht statisch typisiert sind, mittels Bibliotheken realisierbar)**
- **Konzept / Stärken:**
  - **Hinzufügen von Aspekten (2 Teile: Advice (was?) und Pointcut (wo?))**
  - **Diese sind quer (horizontal) zur Klassenhierarchie**
    Bsp.: Transaktionen, Logging, Security, Trace
  - **Der Advice Aspekt bekommt den Methodenaufruf als Argument**
  - **"cross cutting concern"**
- **Schwächen:**
  - **kann zu schwer nachvollziehbarem Verhalten führen**
- **Einsatz:**
  - **z.B. Transaktionssteuerung, Logging, Security**

27

Meta-Programmierung

Bsp.: Ruby

```
class Recorder
      def initialize
        @messages = [ ]
      end

      def method_missing(method, *args, &block)
        @messages << [method, args, block]
      end

      def play_back_to(obj)
        @messages.each do |method, args, block|
          obj.send(method, *args, &block)
        end
      end
    end
```

Aufruf:

```
r = Recorder.new
r.sub!(/Java/) { "Ruby" }
r.upcase!
r[11, 5] = "Universe"
r << "!"

s = "Hello Java World"
r.play_back_to(s)
assert_equal "HELLO RUBY Universe!", s
```

28

Meta-Programmierung

- **Sprachen:**
  - **Ruby, Java, Smalltalk,**
    (im Prinzip alle Sprachen die ein *eval* beinhalten)
- **Konzepte:**
  - **Reflective Programming**
    (kann Variable fragen was für methoden sie besitzen, welche Interfaces sie Implementieren, zu welcher Klassen sie gehören)
    [Selbstaussagen werden möglich -> generische Aufrufe]
  - **Generative Programming**
    (erzeugen von adaptivem Code zur Lauf- oder Compilezeit
    (Erweiterungen, Entfernen von Codebestandteilen)
- **Stärken:**
  - **Gesteigerte Komplexität und**
  - **Abstraktionsniveau**
  - **Erhöhte Lesbarkeit**
- **Schwächen:**
  - **Gesteigerte Komplexität und**
  - **Abstraktionsniveau**
  - **Verminderte Lesbarkeit**

29

Model Driven Programming

- **Konzept:**
  - Generative Programmierung
  - Codegeneratoren,
    die das Meta-Model in die eigentliche Programmierung überführen
- **2 Unterarten:**
  - Graphische Programmierung
    - UML, …
  - Domain-spezifische Sprachen
    - Language Oriented Programming
      MS Oslo, JetBrains: Metaprogramming System

30

Multiparadigmen Programmierung

- **Fast alle heutigen Sprachen sind multi-paradigmen fähig**
- **Trend:**
  - Funktionale Sprachen um OO-Features erweitert (Bsp: OCaml, Loops, Clojure)
  - OO-Sprachen werden um funktionale Bestandteile erweitert (z.B. um Closures) (Bsp. Scala)
  - OO-Sprachen werden aspectorientiert erweitert (Bsp.: AspectJ, Aspect.NET)
  - Metaprogrammier-Erweiterungen (hatte schon Lisp und Prolog)
  - …

31

Quer dazu:

- **Einteilung entsprechend Typisierung:**
  - untypisiert
  - dynamisch
  - statisch
- **Einteilung entsprechend Laufzeitumgebung**
  - interpretiert
  - compiliert
- **Einteilung entsprechend Deklarativität**
  - deklarativ
  - imperativ
- **Entwickungsplattformen**
- **Interoparbilität und Integrierbarkeit mit Vorhandenem**
  - (klassisches Smalltalk-Problem)

32

Esoterische Programmierung

- **Sprachen:**
  - **Whitespace, Brainfuck, Chef, Shakespear**

33

Esoterische Sprachen

Bsp.: Whitespace

34

Esoterische Sprachen

Bsp.: Brainfuck

```
++++++++++
[
  >+++++++>++++++++++>+++>+<<<<-
]                       // Schleife zur Vorbereitung der Textausgabe
>++.                    // Ausgabe von 'H'
>+.                     // Ausgabe von 'e'
+++++++.                // 'l'
.                       // 'l'
+++.                    // 'o'
>++.                    // Leerzeichen
<<++++++++++++++.       // 'W'
>.                      // 'o'
+++.                    // 'r'
------.                 // 'l'
--------.               // 'd'
>+.                     // '!'
>.                      // Zeilenumbruch
```

35

Esoterische Sprachen

Bsp.: Chef

Hallo-Welt-Soufflee

Ingredients.
72 g haricot beans
101 eggs
108 g lard
111 cups oil
32 zucchinis
119 ml water
…

Method.
Put lard into the mixing bowl.
Put oil into the mixing bowl.
Put water into the mixing bowl.
Put zucchinis into the mixing bowl.
Put oil into the mixing bowl.
Put lard into the mixing bowl.
Put haricot beans into the mixing bowl.
Liquefy contents of the mixing bowl.
Pour contents of the mixing bowl into the baking dish.

Serves 1.

36

Esoterische Sprachen

Bsp.: Shakespear

The Romeo without Julia Program

Romeo, a young man with a remarkable patience.
Juliet, a likewise young woman of remarkable grace.
Ophelia, a remarkable woman much in dispute with Hamlet.
Hamlet, the flatterer of Andersen Insulting A/S.

Act I: Hamlet's insults and flattery.
Scene I: The insulting of Romeo.

[Enter Hamlet and Romeo]

Hamlet: You lying stupid fatherless big smelly half-witted coward!
You are as stupid as the difference between a handsome rich brave hero and thyself! Speak your mind!

[Exeunt Hamlet and Romeo]

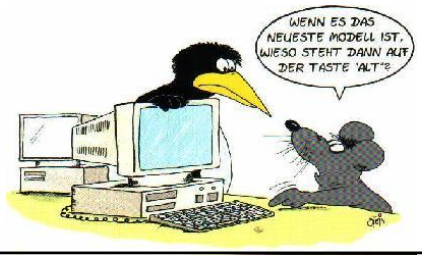
37

Real programmers code in binary.

38

Zusammenfassung

- Getrieben wird die Weiterentwicklung der Programmiersprachen und –paradigmen heute durch 3 Ziele:
  - Hohe Produktivität (rapid development)
  - Gute Wartbarkeit (Lesbarkeit durch Andere)
  - Gute Parallelisierbarkeit (heutige Mehrkern-Prozessorarchitektur)



39

40

Kontaktieren Sie uns:

Prof. Dr. Günther Specht
Universität Innsbruck
Institut für Informatik
Technikerstr. 21a
A-6020 Innsbruck

Tel: ++43 (0)512 507 96820
email: guenther.specht@uibk.ac.at

41