

PG

Autoren: Ulrike Böttcher, Christian Münster

Inhaltliches Lektorat: Andrea Schwarz

Überarbeitete Ausgabe vom 19. April 2006

© by HERDT-Verlag für Bildungsmedien GmbH,
Bodenheim

Internet: www.herdt4you.de/ .at/ .ch
www.herdt4business.de/ .at/ .ch
www.herdt4vhs.de/ .at

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Microfilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Herausgebers reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Diese Unterlage wurde mit großer Sorgfalt erstellt und geprüft. Trotzdem können Fehler nicht vollkommen ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Die Bildungsmedien des HERDT-Verlags enthalten Links bzw. Verweise auf Internetseiten anderer Anbieter. Auf Inhalt und Gestaltung dieser Angebote hat der HERDT-Verlag keinerlei Einfluss. Hierfür sind alleine die jeweiligen Anbieter verantwortlich.

Programmierung

Grundlagen

PG

1 Einstieg in die Programmierung	4	6.3	Dualsystem.....	55	
1.1	Was Sie wissen sollten	4	6.4	Oktalsystem.....	57
1.2	Grundbegriffe.....	5	6.5	Hexadezimalsystem.....	58
1.3	Warum programmieren?	6	6.6	Programme basieren auf Daten.....	60
1.4	Übung	7	6.7	Digitales Rechnen.....	62
2 Grundlagen der Software-Entwicklung	8	6.8	Zeichencodes	64	
2.1	Prinzipien und Methoden des Software-Entwurfs.....	8	6.9	Übung.....	66
2.2	Phasenmodell des Software-Lebenszyklus....	10	7 Grundlegende Sprachelemente	68	
2.3	Computergestützte Software-Entwicklung (CASE)	16	7.1	Merkmale und Darstellungsmittel einer Sprache	68
2.4	Vorgehensmodelle.....	16	7.2	Allgemein gültige syntaktische Regeln.....	70
2.5	Qualitätskriterien	20	7.3	Variablen	72
2.6	Hinweise zur Software-Entwicklung	24	7.4	Konstanten	74
2.7	Schnellübersicht	25	7.5	Standarddatentypen (elementare Datentypen).....	75
2.8	Übung	25	7.6	Operatoren	78
3 Programmiersprachen	26	7.7	Ausdrücke.....	82	
3.1	Die Klassifizierung nach Generationen	26	7.8	Übung.....	83
3.2	Die Klassifizierung nach Sprachtypen.....	29	8 Kontrollstrukturen	84	
3.3	Prozedurale Programmiersprachen	29	8.1	Grundlagen.....	84
3.4	Logische Programmiersprachen	32	8.2	Anweisungen und Anweisungsfolgen	84
3.5	Funktionale Programmiersprachen	33	8.3	Bedingungen	86
3.6	Objektorientierte Programmiersprachen	34	8.4	Kontrollstrukturen einsetzen	87
3.7	Übung	35	8.5	Mit Verzweigungen arbeiten	89
4 Darstellungsmittel für Programmabläufe	36	8.6	Einseitige Verzweigung	90	
4.1	Einleitung.....	36	8.7	Zweiseitige Verzweigung	90
4.2	Programmablaufplan.....	36	8.8	Mehrstufige (verschachtelte) Verzweigung	92
4.3	Datenflussdiagramm	39	8.9	Mehrseitige Verzweigung (Fallauswahl)	94
4.4	Struktogramme	40	8.10	Schleifen.....	98
4.5	Pseudocode.....	42	8.11	Zählergesteuerte Schleife (Iteration).....	99
4.6	Weitere Darstellungsformen.....	42	8.12	Kopfgesteuerte bedingte Schleife	101
4.7	Entscheidungstabellen.....	44	8.13	Fußgesteuerte bedingte Schleife	102
4.8	Übung	45	8.14	Schnellübersicht	104
5 Werkzeuge der Software-Entwicklung	46	8.15	Übung	105	
5.1	Programme erstellen.....	46	9 Elementare Datenstrukturen	106	
5.2	Übersetzer	47	9.1	Warum werden Datenstrukturen benötigt?...	106
5.3	Debugger	50	9.2	Arrays	106
5.4	Entwicklungsumgebungen	50	9.3	Eindimensionale Arrays	107
5.5	Standardbibliotheken	52	9.4	Zwei- und mehrdimensionale Arrays.....	110
5.6	Grundaufbau eines Programms	52	9.5	Zeichenketten	111
6 Zahlensysteme und Zeichencodes	54	9.6	Records	112	
6.1	Zahlensysteme unterscheiden	54	9.7	Zeiger	114
6.2	Dezimalsystem	54	9.8	Stapel	116
			9.9	Schlangen.....	118
			9.10	Listen	119
			9.11	Einfach verkettete Listen	119
			9.12	Doppelt verkettete Liste	123
			9.13	Übung	125

10 Prozeduren und Funktionen 126

10.1 Was sind Unterprogramme? 126
 10.2 Prozeduren 127
 10.3 Funktionen 128
 10.4 Parameterübergabe 129
 10.5 Parameterübergabe als Wert 130
 10.6 Parameterübergabe über Referenzen 133
 10.7 Rückgabewerte von Funktionen 133
 10.8 Übung 135

11 Algorithmen 136

11.1 Was ist ein Algorithmus? 136
 11.2 Iterativer Algorithmus 137
 11.3 Rekursiver Algorithmus 139
 11.4 Iterativ oder rekursiv? 141
 11.5 Generischer Algorithmus 142
 11.6 Schnellübersicht 142
 11.7 Übung 142

12 Spezielle Algorithmen 144

12.1 Suchalgorithmen 144
 12.2 Lineare Suche 144
 12.3 Binäre Suche 146
 12.4 Sortieralgorithmen 148
 12.5 Bubble-Sort 148

12.6 Insertion-Sort 150
 12.7 Shell-Sort 152
 12.8 Quick-Sort 154
 12.9 Vergleich der Sortierverfahren 157
 12.10 Mit Daten in Dateien arbeiten 157
 12.11 Übung 159

13 Einführung in die objektorientierte Programmierung (OOP) 160

13.1 Kennzeichen der objektorientierten Programmierung 160
 13.2 Stufen der OOP 161
 13.3 Prinzipien der OOP 162
 13.4 Klassen und Objekte 164
 13.5 Klassen 164
 13.6 Objekte, Attribute, Methoden 165
 13.7 Datenabstraktion und Datenkapselung 167
 13.8 Konstruktoren und Destruktoren 168
 13.9 Vererbung 168
 13.10 Polymorphie 170
 13.11 Schnellübersicht 172
 13.12 Übung 172

Stichwortverzeichnis 174

3 Programmiersprachen

In diesem Kapitel erfahren Sie

- wie sich die Programmiersprachen geschichtlich entwickelt haben
- nach welchen Kriterien Programmiersprachen eingeteilt werden

3.1 Die Klassifizierung nach Generationen

Es existieren heutzutage mehrere hundert Programmiersprachen, die sich nach ihrer historischen Entwicklung und nach ihren Anwendungsgebieten klassifizieren lassen. Die Einteilung erfolgt nach sachlich-logischen Überlegungen in so genannte Programmiersprachengenerationen.

Erste Generation: Maschinensprachen (Maschinencode)

Damit ein Computer Probleme lösen kann, muss ihm der Lösungsweg in einer ihm verständlichen Art und Weise mitgeteilt werden. Eine Maschinensprache erfüllt genau diese Bedingung.

Beschreibung	Die Abfolgen von Anweisungen werden in binärer Form eingegeben, das heißt, sowohl die Operationen als auch die Daten werden vom Programmierer ausschließlich als Bitfolge (0 oder 1) eingegeben.
Nachteile	Für jeden Computer müssen die Maschinenbefehle neu entwickelt werden, da diese Sprache von den Eigenschaften der Hardware (Prozessoren) abhängig ist. Algorithmen in Maschinensprache sind schwer lesbar, unübersichtlich und mit einem hohen Programmieraufwand verbunden. Deshalb wird diese Sprache heute kaum mehr verwendet.
Besonderheit	Der Programmierer hat die Aufgabe, die Grundoperationen und die Speichereinheiten, unter denen die zu verarbeitenden Daten im Arbeitsspeicher gespeichert sind, selbst zu verwalten.
Beispiel (3 + 4)	00011010 0011 0100

Zweite Generation: Assembler-Sprachen

Beschreibung	Assembler-Sprachen sind wie Maschinensprachen an bestimmte Prozessoren gebunden. Übersetzungsprogramme, die Assembler-Programme in Maschinencode umwandeln, werden als Assembler bezeichnet. Bei der Übersetzung entsteht ein Programm, das von einem bestimmten Prozessor ausgeführt werden kann.
Einsatz	Assembler-Sprachen werden überwiegend zur Programmierung der Hardware oder für schnelle, zeitkritische Programme und Programmteile eingesetzt.
Vorteile	Im Vergleich zur Maschinensprache bieten Assembler-Sprachen durch Operationskürzel dem Programmierer wesentliche Erleichterungen, wie beispielsweise durch Operationen wie ADD (addiere) oder MOV (bewege - move). Da Assembler-Sprachen auf die maschinenspezifischen Besonderheiten des jeweiligen Computers abgestimmt und (durch den Programmierer) optimiert sind, verbrauchen die Programme im Allgemeinen weniger Speicherplatz und sind meist auch schneller als ein entsprechendes Programm in einer anderen Programmiersprache. Die Effizienz von Assembler-Programmen hängt aber auch von den Fähigkeiten des Programmierers ab.

Vierte Generation: Deklarative Sprachen

Während die ersten drei Generationen noch relativ klar voneinander getrennt werden können, fehlen bei den folgenden Generationen eindeutige Kriterien.

Beschreibung	Bei deklarativen Programmiersprachen beschreibt der Programmierer lediglich, was das Programm leisten soll, ohne den genauen algorithmischen Weg anzugeben. Eine einzelne Anweisung löst eine ganze Folge von internen Einzelschritten aus. Sie werden deshalb auch als nicht prozedurale Sprachen bezeichnet.
Einsatz	Nicht prozedurale Programmiersprachen sind auf spezielle Anwendungsgebiete ausgerichtet. So gibt es Sprachen zum Bearbeiten und Auswerten von Dateien, Datenbanken, Tabellenkalkulationen oder zum Erstellen von Bildschirmformularen.
Vorteile	Zum Programmieren stehen fortschrittliche, einfach zu bedienende und leistungsfähige Entwicklungssysteme zur Verfügung. Die Erstellung eines Programms wird dadurch wesentlich erleichtert (quantitativ und qualitativ).
Nachteile	Die größtmögliche Effizienz der Programme ist nicht gegeben. Außerdem sind sie nur für spezielle Einsatzgebiete verwendbar. Da bestimmte Folgen von Arbeitsschritten innerhalb einer Anweisung automatisch ausgeführt werden, hat der Programmierer kaum einen Einfluss auf die internen Abläufe. Die Ausführungsgeschwindigkeit dieser Programme ist aufgrund der mächtigeren Sprache langsamer als bei problemorientierten Sprachen.
Programmiersprachen	SQL, Natural, Symphony, Open Access
Beispiel	CREATE Adresskartei SELECT Kunde FROM Tabelle WHERE KdNr = 10

Fünfte Generation: Künstliche Intelligenz

Beschreibung	Der Grundgedanke des Forschungsgebietes der künstlichen Intelligenz ist es, zu untersuchen, unter welchen Bedingungen Computer menschliche Verhaltensweisen, die auf Intelligenz beruhen, nachvollziehen können. Für diese Forschungszwecke sind einige spezielle Programmiersprachen entwickelt worden.
Einsatz	Inzwischen umfasst dieses Gebiet mehrere Fachbereiche, beispielsweise die Robotik, die zur Entwicklung von Robotern und deren komplizierten Bewegungsabläufen geführt hat, die Wissensverarbeitung und die Spracherkennung.
Vorteile	Da die zu verarbeitenden Daten im Wesentlichen aus einer Folge von Wörtern bestehen, liegt die besondere Stärke dieser Sprachen in der Verarbeitung von Zeichenketten.
Programmiersprachen	Prolog, Lisp, Smalltalk
Beispiel	Links_von(Apfel, Birne). Factum Links_von(Pflaume, Apfel). Factum Links_von(X, Y) :- Rechts_von(Y, X). Regel ... Frage: ?-Links_von(Rechts, Links). Frage Antwort: Links = Birne, Rechts = Apfel Antwort

Auswahl prozeduraler Programmiersprachen

Die Programmiersprache FORTRAN

Bedeutung	FORTRAN steht für Formula Translator .
Entwicklung	FORTRAN wurde 1954 von IBM entwickelt und gilt als die erste Sprache der dritten Generation.
Einsatz	FORTRAN ist wissenschaftlich orientiert und wird hauptsächlich für mathematische Anwendungen eingesetzt.
Nachteil	Für das Arbeiten mit Text und umfangreichen Datenbeständen ist die Sprache ungeeignet.

Die Programmiersprache COBOL

Bedeutung	COBOL steht für Common Business Oriented Language .
Entwicklung	COBOL wurde speziell für betriebswirtschaftliche Anwendungen entwickelt und 1959 eingeführt. 1968 wurde COBOL durch das American National Standard Institute (ANSI) genormt. Seitdem wurde es kontinuierlich erweitert und modifiziert. Der letzte gültige Stand ist ANSI-COBOL-85.
Einsatz	Betriebswirtschaftliche Anwendungen, die sich durch relativ einfache Algorithmen sowie umfangreiche Datenmengen und Dateiverarbeitungen auszeichnen COBOL ist eine noch immer weit verbreitete Programmiersprache. Im Bereich der kaufmännischen Großrechner sind weit über die Hälfte aller Anwendungen in COBOL geschrieben. Selbst auf dem PC werden betriebswirtschaftliche Anwendungen, die auf Großrechnerdaten zugreifen, in COBOL entwickelt.
Vorteil	COBOL ist stark an der gesprochenen englischen Sprache orientiert und daher leichter zu lernen und zu lesen als andere Programmiersprachen wie z. B. C++.
Nachteil	Gerade durch die große Anlehnung an eine gesprochene Sprache ist die Programmierung in COBOL mit einem gewissen Aufwand bei der Eingabe des Programmquelltextes verbunden.

Die Programmiersprache BASIC

Bedeutung	BASIC steht für Beginners All-Purpose Symbolic Instruction Code .
Entwicklung	BASIC wurde 1965 für Schulungszwecke entwickelt. Die Weiterentwicklung von BASIC führten viele Hersteller allerdings im Alleingang durch, weshalb mittlerweile für fast jeden Rechnertyp eine eigene BASIC-Version existiert.
Einsatz	Haupteinsatzbereich für BASIC sind Mikro-Computer.
Vorteil	BASIC ist relativ leicht zu erlernen.
Nachteil	Für größere Anwendungen ist BASIC eher ungeeignet. Die Sprache bietet nur unzureichende fortschrittliche Strukturierungsmöglichkeiten. Die Programme werden unübersichtlich und sind dadurch relativ schwer nachzuvollziehen. Visual Basic.NET stellt eine objektorientierte Weiterentwicklung von BASIC dar und bleibt kaum hinter den Möglichkeiten anderer objektorientierter Programmiersprachen zurück.

Die Programmiersprache ADA

Bedeutung	Diese Sprache entwickelte der Franzose Jean Ichbiah 1979 für das Militär. Zum Gedenken benannte er sie nach der ersten "Programmiererin" der Welt, Augusta Ada Byron, Countess of Lovelace.
Entwicklung	1983 wurde ADA als kommerzieller und internationaler Standard übernommen.
Einsatz	ADA war ursprünglich für militärische Anwendungen ausgelegt, ist aber eine allgemeine Sprache, die zur Lösung der meisten Programmierprobleme benutzt werden kann. Sie hat eine Blockstruktur und einen Datentypmechanismus, ähnlich wie PASCAL, weist aber auch Erweiterungen für Echtzeitanwendungen auf.
Vorteil	Sie besitzt Merkmale, die andere Programmiersprachen nicht vorweisen können: Verwaltung und Echtzeitsteuerung von Prozessen, gleichzeitiges Ausführen von Programmen, Ausnahmebehandlungen und abstrakte Datentypen.
Nachteil	ADA verbirgt, wie sich ein Programm intern bei der Ausführung verhält. Das heißt, für den Programmierer ist es schwer nachvollziehbar, wie das Programm letztlich ausgeführt wird. Außerdem wurde ADA durch die rasante Verwendung der Sprachen C und C++ relativ schnell als Programmiersprache verdrängt.

3.4 Logische Programmiersprachen

Diese Sprachen sind auf die Lösung von bestimmten Problemen abgestimmt (Datenbankabfragen, mathematische Beweise). Es wird kein Algorithmus zum Lösen eines Problems angegeben, sondern es werden lediglich die Bedingungen für eine korrekte Lösung bestimmt. Bei logischen Programmiersprachen berechnet das Programm eine Beziehung zwischen den einzelnen Daten.

Beispiel

Sie geben eine Menge von Fakten und Regeln ein, die eine Wissensbasis bilden. Stellen Sie dann eine Frage, erhalten Sie aufgrund dieser Wissensbasis eine entsprechende Antwort.

(1) <i>Männlich</i> (Steffen)	
(2) <i>Männlich</i> (Sebastian)	
(3) <i>Weiblich</i> (Ina)	
(4) <i>Ist_Vater_von</i> (Bernd, Ina)	<i>bedeutet: Bernd ist Vater von Ina</i>
(5) <i>Ist_Vater_von</i> (Bernd, Sebastian)	<i>bedeutet: Bernd ist Vater von Sebastian</i>
(6) <i>Ist_Vater_von</i> (Bernd, Steffen)	<i>bedeutet: Bernd ist Vater von Steffen</i>
(7) <i>Ist_Sohn_von</i> (X,Y):- <i>Ist_Vater_von</i> (Y,X), <i>Männlich</i> (X)	<i>bedeutet: X ist Sohn von Y, wenn Y</i>
(8) <i>Ist_Tochter_von</i> (X,Y):- <i>Ist_Vater_von</i> (Y,X), <i>Weiblich</i> (X)	<i>Vater von X ist und X männlich ist</i>
(9) <i>?-Ist_Sohn_von</i> (X,Y)	<i>bedeutet: X ist Tochter von Y, wenn Y</i>
X = Sebastian	<i>Vater von X ist und X weiblich ist</i>
Y = Bernd	<i>Frage, wer (X) ist Sohn von wem (Y)</i>
	<i>Antworten:</i>
X = Steffen	<i>Sebastian ist Sohn von Bernd</i>
Y = Bernd	<i>und</i>
(10) <i>?-Ist_Tochter_von</i> (X,Y)	<i>Steffen ist Sohn von Bernd</i>
X = Ina	<i>Frage, wer (X) ist Tochter von wem (Y)</i>
Y = Bernd	<i>Antworten:</i>
	<i>Ina ist Tochter von Bernd</i>

In den ersten sechs Zeilen werden Fakten gesammelt. Die Zeilen 7 und 8 sind Regeln. Aufgrund der Regeln und Fakten werden die Fragen 9 und 10 beantwortet.

Die Programmiersprache PROLOG

Bedeutung	Der Name PROLOG ist eine Abkürzung für "PROgramming in LOGic".
Entwicklung	Die Programmiersprache PROLOG wurde 1970 von Alain Colmerauer und Philippe Roussel entwickelt. PROLOG hat sich in den 80er-Jahren verbreitet, jedoch wurde durch die fehlende Entwicklung von Anwendungen die weitere Verbreitung behindert.
Einsatz	Dient als experimentelles Werkzeug für künstliche Intelligenz, wird jedoch hauptsächlich an Universitäten benutzt

3.5 Funktionale Programmiersprachen

Die prozeduralen Programmiersprachen sind durch die Abfolge von Befehlen gekennzeichnet. Im Gegensatz dazu zeichnet sich eine funktionale Programmiersprache durch den Gebrauch von Ausdrücken und Funktionen aus.

Bei funktionalen Programmiersprachen ist das Programm eine Funktion, die sich typischerweise auf einfachere Funktionen stützt, daher auch der Name "funktionale Programmiersprache". Die Beziehungen zwischen den Funktionen sind einfach: Eine Funktion kann eine andere aufrufen, oder das Ergebnis einer Funktion kann als Parameter für eine andere Funktion genutzt werden.

Programme werden wie mathematische Funktionen geschrieben. Eine Funktion hat einen Definitions- und einen Wertebereich. Die Funktion erhält einen Eingabewert und berechnet, mathematisch gesehen, den Wert der Funktion.

Beispiel

Das folgende Beispiel zeigt eine Funktion zur Fakultätsberechnung in der Sprache Lisp.

<pre>> (defun fakultaet (x) (if (x > 0) (* x (fakultaet (- x 1))) 1)) FAKULTAET > (fakultaet 5) 120</pre>	<pre>} Funktionsdefinition → Antwort: definierter Funktionsname → Funktionsaufruf mit dem Wert 5 → Antwort: errechneter Wert</pre>
-------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------

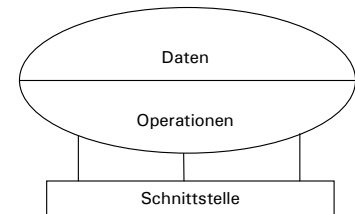
Die Programmiersprache LISP

Entwicklung	Die Programmiersprache LISP wurde von John McCarthy und einer Arbeitsgruppe am Massachusetts Institute of Technology Anfang der 60er-Jahre entwickelt. Es war die erste implementierte funktionale Sprache. Heute gibt es viele Dialekte von LISP, wie z. B. die Sprache Scheme.
Einsatz	LISP wird in der Informatik im Forschungsbereich eingesetzt und dort vorwiegend im Bereich der künstlichen Intelligenz und zur Lösung mathematischer Probleme.
Vorteil	Die Sprache ermöglicht es, komplexe Programme, die Zeichenketten bearbeiten, leichter als in anderen Programmiersprachen zu schreiben.
Nachteil	Sie ist nicht sehr weit verbreitet.

3.6 Objektorientierte Programmiersprachen

Die Weiterentwicklung der prozeduralen Programmierung führte zu objektorientierten Programmiersprachen. Das Problem der prozeduralen Programmierung ist, dass globale Daten in jedem Teil des Programms manipuliert und überschrieben werden können. Es fehlt eine Verbindung zwischen den Daten eines Programms und den sie manipulierenden Funktionen. Dies führt dazu, dass große Programme sehr leicht unübersichtlich werden und sich schwerer testen lassen.

1970 erkannte David Parnas das Problem und hatte die Idee, jedes einzelne Datenelement in einem Modul zu kapseln. Der direkte Zugriff auf diese Daten wurde nur über eine bestimmte Schnittstelle mit einem Satz von Operationen, wie z. B. über Prozeduren oder Funktionen, erlaubt. Sollen andere Module ebenfalls auf die Variable zugreifen, können sie dies nur indirekt über die Schnittstelle des Moduls tun.



Auswahl objektorientierter Programmiersprachen

Die Programmiersprache C++

Entwicklung	1982 begann Bjarne Stroustrup eine Erweiterung der prozeduralen Programmiersprache C zu entwickeln. 1989 wurde die Basissprache definiert, 1996 der internationale Standard verabschiedet. C++ stellt für den Anwender einen Übergang von der prozeduralen zur objektorientierten Programmierweise dar.
Einsatz	C++ eignet sich für die Entwicklung von systemnahen Programmen und von komplexen Anwendungen.
Vorteil	Es gibt zur objektorientierten Programmierung mit C++ mächtige Programmierwerkzeuge. C++-Compiler stehen praktisch auf jeder Rechnerplattform zur Verfügung. C++ besitzt mächtige Sprachmittel, um komplexe Anwendungen zu erzeugen. Außerdem stehen umfangreiche Klassenbibliotheken zur Verfügung.
Nachteil	Durch die systemnahen Anwendungsmöglichkeiten kann die falsche Anwendung von Sprachmitteln, z. B. Zeigern, sehr schnell zu Programmfehlern führen. Mögliche Fehlerquellen müssen manuell überwacht werden, sodass die Entwicklung von Programmen mit C++ nur sehr fortgeschrittenen Programmierern zu empfehlen ist.

Die Programmiersprache SMALLTALK

Entwicklung	Anfang der 70er-Jahre wurde SMALLTALK von Alan Kay am Xerox Palo Alto Research Center (PARC) entwickelt. Die Sprache wurde in sich selbst programmiert und sollte auch Nichtinformatikern die Benutzung erleichtern.
Einsatz	Wird vorwiegend in der Forschung und in der Lehre benutzt
Vorteil	SMALLTALK ist eine rein objektorientierte Sprache. Alle Werte sind Objekte, sogar Objektklassen sind Objekte. Kontrollstrukturen sind lediglich Operationen entsprechender Klassen. Deshalb kommt SMALLTALK mit wenigen Konzepten aus.
Nachteil	Programme in SMALLTALK können nicht ohne weiteres auf ein anderes System übertragen werden. Da diese Sprache interpretiert wird, sind die Programme sehr langsam.

Die Programmiersprache EIFFEL

Entwicklung	Die Sprache EIFFEL wurde Mitte der 80er-Jahre von Bertrand Meyer entwickelt.
Einsatz	Wird vorwiegend in der Forschung und in der Lehre benutzt
Vorteil	EIFFEL ist, wie SMALLTALK, eine objektorientierte Sprache, in der alle Typen Klassen und alle Werte Objekte sind und in der objektorientierte Konzepte angewendet werden können.
Nachteil	Die Sprache ist wenig verbreitet, und es gibt somit kaum vorgefertigte Bibliotheken.

