

# HANSER



## Leseprobe

zu

## „Technische Probleme lösen mit C/C++“

von Norbert Heiderich und Wolfgang Meyer

Print-ISBN: 978-3-446-46308-0

E-Book-ISBN: 978-3-446-46358-5

Weitere Informationen und Bestellungen unter  
<http://www.hanser-fachbuch.de/978-3-446-46308-0>

sowie im Buchhandel

© Carl Hanser Verlag, München

# Vorwort des Herausgebers

## Was können Sie mit diesem Buch lernen?

Wenn Sie mit diesem Lernbuch arbeiten, dann erwerben Sie umfassende Erkenntnisse, die Sie zur Problemlösungsfähigkeit beim Programmieren mit der Hochsprache C/C++ führen.

Der Umfang dessen, was wir Ihnen anbieten, orientiert sich an

- den Studienplänen der Fachhochschulen für technische Studiengänge,
- den Lehrplänen der Fachschulen für Technik,
- den Anforderungen der Programmierpraxis,
- dem Stand der einschlägigen, professionellen Softwareentwicklung.

Sie werden systematisch, schrittweise und an ausgewählten Beispielen mit der Entwicklungsumgebung Visual C++ (VC++) von Microsoft vertraut gemacht.

Dabei gehen Sie folgenden Strukturelementen und Verfahrensweisen nach:

- Wie stellt sich die Entwicklungsumgebung dar?
- Welche grundlegenden Sprach- und Steuerungswerkzeuge gilt es kennenzulernen und an einfachen Beispielen anzuwenden?
- Wie wird ein Problem strukturiert programmiert?
- Wie muss die Software dokumentiert und getestet werden?
- Was meint objektorientierte Programmierung?

## Wer kann mit diesem Buch lernen?

Jeder, der

- sich weiterbilden möchte,
- die Grundlagen der elektronischen Datenverarbeitung beherrscht,
- Kenntnisse in den Grundlagen der elementaren Mathematik besitzt,
- bereit ist, sich mit technischen, mathematischen und kommerziellen Fragestellungen auseinanderzusetzen.

Das können sein:

- Studenten an Fachhochschulen und Berufsakademien,
- Studenten an Fachschulen für Technik,
- Schüler an beruflichen Gymnasien und Berufsoberschulen,
- Schüler in der Assistentenausbildung,
- Meister, Facharbeiter und Gesellen während und nach der Ausbildung,
- Umschüler und Rehabilitanden,
- Teilnehmer an Fort- und Weiterbildungskursen,
- Autodidakten.

## Wie können Sie mit diesem Buch lernen?

Ganz gleich, ob Sie mit diesem Buch in Hochschule, Schule, Betrieb, Lehrgang oder zu Hause lernen, es wird Ihnen Freude machen!

*Warum?*

Ganz einfach, **weil wir Ihnen ein Buch empfehlen, das in seiner Gestaltung die Grundgesetze des menschlichen Lernens beachtet.**

*- Ein Lernbuch also! -*

Sie setzen sich kapitelweise mit den Lehr-, Lerninhalten auseinander. Diese sind in überschaubaren Lernsequenzen schrittweise dargestellt. Die zunächst verbal formulierten Lehr-, Lerninhalte werden danach in die softwarespezifische Darstellung umgesetzt. An ausgewählten Beispielen konkretisiert und veranschaulichen die Autoren diese Lehr- bzw. Lerninhalte.

*- Also auch ein unterrichtsbegleitendes Lehr-/Lernbuch mit Beispielen! -*

Für das Suchen bestimmter Inhalte steht Ihnen das Inhaltsverzeichnis am Anfang des Buches zur Verfügung. Sachwörter finden Sie am Ende des Buches. Bücher zur vertiefenden und erweiterten Anwendung sind im Literaturverzeichnis zusammengestellt.

*- Selbstverständlich mit Sachwortregister, Inhalts- und Literaturverzeichnis! -*

Sicherlich werden Sie durch intensives Arbeiten mit diesem Buch Ihre „Bemerkungen zur Sache“ unterbringen und es so zu Ihrem individuellen Arbeitsmittel ausweiten:

*- So wird am Ende Ihr Buch entstanden sein! -*

Möglich wurde dieses Buch für Sie durch die Bereitschaft der Autoren und die intensive Unterstützung des Verlages mit seinen Mitarbeitern. Ihnen sollten wir herzlich danken.

Beim Lernen wünsche ich Ihnen viel Freude und Erfolg!

Ihr Herausgeber

*Manfred Mettke*

# Inhalt

|  |           |
|--|-----------|
| <b>Einleitung</b>                                    | <b>13</b> |
| <b>1 Systematik der Problemlösung</b>                | <b>17</b> |
| 1.1 Phasen der Programmentwicklung                   | 17        |
| 1.2 Software-Lebenszyklus                            | 19        |
| 1.3 Software-Entwicklungsverfahren                   | 21        |
| <b>2 Erste Gehversuche mit C/C++</b>                 | <b>26</b> |
| 2.1 Warum gerade C/C++?                              | 26        |
| 2.2 Compiler und Interpreter                         | 28        |
| 2.3 Übersetzen eines C/C++-Programms                 | 30        |
| 2.4 Programmstart                                    | 31        |
| <b>3 Die Entwicklungsumgebung Visual C++</b>         | <b>32</b> |
| 3.1 Installation von VC++                            | 32        |
| 3.2 Starten von VC++                                 | 34        |
| 3.3 Erstellen eines neuen Projektes                  | 36        |
| 3.3.1 Win32-Projekte                                 | 37        |
| 3.3.1.1 Variante 1 – VC++ leistet Vorarbeit          | 38        |
| 3.3.1.2 Variante 2 – leeres Projekt                  | 39        |
| 3.3.2 CLR-Projekte                                   | 42        |
| 3.4 Übersetzen eines eigenen Programms               | 44        |
| 3.5 Ausführen eines eigenen Programms                | 47        |
| 3.6 Paradigmen der Projektorganisation               | 47        |
| <b>4 Grundlegende Sprach- und Steuerungselemente</b> | <b>50</b> |
| 4.1 Kommentare                                       | 50        |
| 4.2 Datentypen und Variablen                         | 51        |
| 4.2.1 Variablennamen                                 | 52        |
| 4.2.2 Ganzzahlige Variablen                          | 52        |
| 4.2.3 Fließkommazahlen                               | 54        |
| 4.2.4 Zeichen  | 55        |
| 4.2.5 Felder   | 56        |

|         |   |    |
|---------|---|----|
| 4.2.5.1 | Eindimensionale Felder .....                                  | 56 |
| 4.2.5.2 | Mehrdimensionale Felder .....                                 | 57 |
| 4.2.5.3 | Zugriff auf die Elemente eines Feldes .....                   | 59 |
| 4.2.5.4 | Startwertzuweisung für ein- und mehrdimensionale Arrays ..... | 61 |
| 4.2.6   | Zeichenketten .....   | 63 |
| 4.3     | Konstanten .....  | 64 |
| 4.4     | Operatoren .....  | 65 |
| 4.4.1   | Vorzeichenoperatoren .....                                    | 65 |
| 4.4.2   | Arithmetische Operatoren .....                                | 65 |
| 4.4.2.1 | Addition + .....  | 65 |
| 4.4.2.2 | Subtraktion - .....   | 65 |
| 4.4.2.3 | Multiplikation * .....  | 66 |
| 4.4.2.4 | Division / .....  | 66 |
| 4.4.2.5 | Modulo % .....  | 66 |
| 4.4.2.6 | Zuweisung = .....   | 66 |
| 4.4.2.7 | Kombinierte Zuweisungen .....                                 | 67 |
| 4.4.2.8 | Inkrementierung ++ .....                                      | 67 |
| 4.4.2.9 | Dekrementierung - .....                                       | 68 |
| 4.4.3   | Vergleichsoperatoren .....                                    | 68 |
| 4.4.3.1 | Gleichheit == .....   | 68 |
| 4.4.3.2 | Ungleichheit != .....   | 68 |
| 4.4.3.3 | Kleiner < .....   | 69 |
| 4.4.3.4 | Größer > .....  | 69 |
| 4.4.3.5 | Kleiner gleich <= .....                                       | 69 |
| 4.4.3.6 | Größer gleich >= .....  | 70 |
| 4.4.4   | Logische Operatoren .....                                     | 70 |
| 4.4.4.1 | Logisches NICHT ! .....                                       | 70 |
| 4.4.4.2 | Logisches UND && .....  | 70 |
| 4.4.4.3 | Logisches ODER    .....                                       | 70 |
| 4.4.5   | Typumwandlungsoperator .....                                  | 71 |
| 4.4.6   | Speicherberechnungsoperator .....                             | 71 |
| 4.4.7   | Bedingungsoperator .....                                      | 72 |
| 4.4.8   | Indizierungsoperator .....                                    | 73 |
| 4.4.9   | Klammerungsoperator .....                                     | 73 |
| 4.5     | Anweisungen und Blöcke .....                                  | 75 |
| 4.6     | Alternationen .....   | 75 |
| 4.6.1   | Einfache Abfragen (if - else) .....                           | 75 |
| 4.6.2   | Mehrfachabfragen (else - if) .....                            | 76 |
| 4.6.3   | Die switch-case-Anweisung .....                               | 77 |
| 4.7     | Iterationen .....   | 79 |
| 4.7.1   | Zählergesteuerte Schleifen (for) .....                        | 79 |
| 4.7.2   | Kopfgesteuerte Schleifen (while) .....                        | 83 |
| 4.7.3   | Fußgesteuerte Schleifen (do - while) .....                    | 84 |
| 4.7.4   | Schleifenabbruch (continue) .....                             | 85 |
| 4.7.5   | Schleifenabbruch (break) .....                                | 86 |
| 4.7.6   | Schleifenumwandlungen .....                                   | 88 |

|          |   |            |
|----------|---|------------|
| 4.8      | Funktionen  | 88         |
| 4.8.1    | Formaler Aufbau einer Funktion                        | 89         |
| 4.8.1.1  | Der Funktionskopf                                     | 90         |
| 4.8.1.2  | Der Funktionsrumpf                                    | 91         |
| 4.8.2    | Datentyp und Deklaration einer Funktion – Prototyping | 92         |
| 4.8.3    | Das Prinzip der Parameterübergabe                     | 97         |
| 4.8.3.1  | Aufrufverfahren call by value                         | 97         |
| 4.8.3.2  | Aufrufverfahren call by reference                     | 99         |
| 4.8.3.3  | Adressoperator, Zeiger und Dereferenzierung           | 102        |
| 4.8.4    | Regeln für ein erfolgreiches Prototyping              | 103        |
| 4.8.5    | Die exit()-Funktion                                   | 104        |
| 4.8.6    | Rekursive Funktionen                                  | 104        |
| 4.9      | Ein- und Ausgabe                                      | 107        |
| 4.9.1    | Formatierte Eingabe mit scanf()                       | 107        |
| 4.9.2    | Formatierte Ausgabe mit printf()                      | 108        |
| 4.9.3    | Arbeiten mit Dateien                                  | 109        |
| 4.9.3.1  | Öffnen der Datei                                      | 110        |
| 4.9.3.2  | Verarbeiten der Datensätze                            | 110        |
| 4.9.3.3  | Schließen der Datei                                   | 111        |
| 4.9.3.4  | stdio.h   | 111        |
| 4.9.3.5  | fflush() und stdin                                    | 113        |
| <b>5</b> | <b>Strukturierte Programmierung</b>                   | <b>114</b> |
| 5.1      | Problemstellung                                       | 115        |
| 5.2      | Problemanalyse  | 116        |
| 5.3      | Struktogramm nach Nassi-Shneiderman                   | 119        |
| 5.3.1    | Sequenz   | 121        |
| 5.3.2    | Alternation   | 123        |
| 5.3.3    | Verschachtelung                                       | 124        |
| 5.3.4    | Verzweigung   | 125        |
| 5.3.5    | Schleifen   | 127        |
| 5.3.5.1  | Zählergesteuerte Schleife                             | 127        |
| 5.3.5.2  | Kopfgesteuerte Schleife                               | 131        |
| 5.3.5.3  | Fußgesteuerte Schleifen                               | 133        |
| 5.3.5.4  | Endlosschleifen                                       | 134        |
| 5.3.5.5  | Kriterien zur Schleifenauswahl                        | 134        |
| 5.3.6    | Programm- oder Funktionsaufruf                        | 134        |
| 5.3.7    | Aussprung   | 135        |
| 5.3.8    | Rechnergestützte Erstellung von Struktogrammen        | 136        |
| 5.3.8.1  | StruktEd  | 136        |
| 5.3.8.2  | hus-Struktogrammer                                    | 143        |
| 5.4      | Flussdiagramm nach DIN 66001                          | 151        |
| 5.5      | Programmerstellung                                    | 153        |
| 5.6      | Programmtest  | 153        |
| 5.7      | Programmlauf  | 154        |
| 5.8      | Dokumentation nach DIN 66230                          | 155        |

|          |  |            |
|----------|--|------------|
| 5.8.1    | Funktion und Aufbau des Programms  | 155        |
| 5.8.2    | Programmkenndaten  | 156        |
| 5.8.3    | Betrieb des Programms  | 157        |
| 5.8.4    | Ergänzungen  | 157        |
| 5.9      | Aspekte des Qualitätsmanagements EN-ISO 9000   | 158        |
| 5.10     | Algorithmus – was ist das?   | 159        |
| 5.11     | EVA-Prinzip  | 165        |
| 5.12     | Programmierung von Formelwerken  | 166        |
| <b>6</b> | <b>Lösung einfacher Probleme</b>   | <b>171</b> |
| 6.1      | Umrechnung von Temperatursystemen  | 171        |
| 6.2      | Flächenberechnung geradlinig begrenzter Flächen (Polygone)   | 177        |
| 6.2.1    | Erste Problemvariation: Berechnung der Schwerpunktkoordinaten $S(x_S; y_S)$ von polygonförmig begrenzten Flächen | 184        |
| 6.2.2    | Zweite Problemvariation: Suche nach einem „günstigen“ Treffpunkt   | 185        |
| 6.3      | Berechnung einer Brückenkonstruktion   | 186        |
| 6.4      | Schaltjahrüberprüfung  | 190        |
| 6.5      | Ein Problem aus der Energiewirtschaft  | 196        |
| 6.6      | Logarithmische Achsenteilung   | 206        |
| <b>7</b> | <b>Objektorientierte Programmierung (OOP)</b>  | <b>214</b> |
| 7.1      | Modellbildung mittels Abstraktion  | 214        |
| 7.2      | Klassen und Objekte  | 215        |
| 7.3      | Attribute und Methoden einer Klasse  | 218        |
| 7.4      | Bruchrechnung mit OOP  | 219        |
| 7.5      | Vererbung  | 228        |
| 7.6      | Strings  | 234        |
| 7.7      | Typumwandlungen  | 236        |
| 7.8      | Strukturierte Programmierung vs. OOP   | 239        |
| <b>8</b> | <b>Lösung fortgeschrittener Probleme</b>   | <b>241</b> |
| 8.1      | Grafische Darstellung funktionaler Abhängigkeiten  | 241        |
| 8.1.1    | Welt- und Screenkoordinaten  | 243        |
| 8.1.2    | Koordinatentransformationen  | 245        |
| 8.1.3    | Darstellung der Sinusfunktion  | 251        |
| 8.1.4    | Darstellung quadratischer Parabeln   | 255        |
| 8.1.5    | Spannungsteilerkennlinien  | 258        |
| 8.2      | Lösung technisch-wissenschaftlicher Probleme   | 260        |
| 8.2.1    | Widerstandsreihen E6 bis E96   | 260        |
| 8.2.2    | Farbcodierung von Widerständen nach DIN 41429  | 263        |
| 8.2.3    | Fourier-Synthese periodischer empirischer Funktionen   | 266        |
| 8.2.4    | Fourier-Analyse empirischer Funktionen   | 274        |
| 8.3      | Nullstellenbestimmung von Funktionen   | 279        |
| 8.3.1    | Inkrementverfahren und Intervallhalbierung   | 279        |
| 8.3.2    | Die regula falsi   | 284        |
| 8.3.3    | Das Newton-Verfahren   | 286        |

|           |   |            |
|-----------|---|------------|
| 8.4       | Numerische Integration .....  | 289        |
| 8.4.1     | Riemannsche Unter- und Obersummen .....   | 289        |
| 8.4.2     | Trapezregel .....   | 293        |
| 8.4.3     | Simpsonsche Regel .....   | 298        |
| 8.4.4     | Effektivwertberechnungen .....  | 303        |
| 8.5       | Einbindung eigener Klassen .....  | 305        |
| 8.5.1     | Das „Platinenproblem“ als objektorientierte Konsolenanwendung ..                                    | 305        |
| 8.5.2     | Das „Platinenproblem“ in der Erweiterung mit grafischer<br>Benutzeroberfläche .....                 | 310        |
| <b>9</b>  | <b>Lösung komplexer Probleme .....</b>  | <b>314</b> |
| 9.1       | Kurvendiskussion und Funktionsplotter am Beispiel<br>ganzrationaler Funktionen bis 3. Ordnung ..... | 314        |
| 9.2       | Ausgleichsrechnung – Bestimmung der „besten“ Geraden<br>in einer Messreihe .....                    | 317        |
| 9.3       | Digitaltechnik .....  | 327        |
| <b>10</b> | <b>Tabellen und Übersichten .....</b>   | <b>341</b> |
| 10.1      | Datentypen und ihre Wertebereiche .....   | 341        |
| 10.2      | Vergleich der Symbole nach DIN 66 001 und der<br>Nassi-Shneiderman-Darstellung .....                | 342        |
| 10.3      | Schlüsselwörter ANSI C .....  | 343        |
| 10.4      | Erweiterte Schlüsselwörter C++ .....  | 345        |
| 10.5      | ASCII-Tabelle .....   | 348        |
| 10.6      | Standardfunktionen und ihre Zuordnung zu den Header-Dateien (Include)                               | 350        |
|           | <b>Literatur .....</b>  | <b>354</b> |
|           | <b>Index .....</b>  | <b>355</b> |



# 1

# Systematik der Problemlösung

Einst löste Alexander der Große den Gordischen Knoten sehr unkonventionell mit dem Schlag seines Schwertes. An den kunstvoll geknoteten Stricken, die einen Streitwagen untrennbar mit seinem Zugjoch verbinden sollten, waren zuvor die Gelehrten gescheitert. Sie versuchten, ihn ohne Beschädigung zu entfernen, quasi die Verknotungen umzukehren. Dies zeigt deutlich, dass ein Problem komplex und damit sogar unlösbar werden kann, wenn man nicht fähig ist, es unvoreingenommen zu betrachten, wenn man sich nicht von unvermeidbar erscheinenden Lösungswegen trennen kann. Die Lösung des Problems soll das Ziel sein – aber auch der Weg dorthin!

Zur Lösung eines Problems mit Hilfe eines Rechners geht man üblicherweise in mehreren Einzelschritten vor. Diese Vorgehensweise ist sinnvoll, weil die in jedem Schritt anfallenden Probleme häufig so speziell sind, dass Fachleute des jeweiligen Gebietes sie lösen müssen. So muss z.B. ein Betriebsführer, der eine Problemstellung sehr genau aus der Sicht des Betriebsablaufes beschreiben und sicherlich aus dieser Sicht auch erste Strategien entwickeln kann, nicht notwendigerweise auch derjenige sein, der mögliche Auswirkungen auf die Buchführung und Abrechnung des Unternehmens beurteilen, oder zur Auswahl geeigneter Programmierelemente und einzusetzender Hardware einen Beitrag leisten kann.

## ■ 1.1 Phasen der Programmentwicklung

In den Anfängen der Datenverarbeitung waren Systemanalyse und methodisches Vorgehen bei der Entwicklung von Software beinahe bedeutungslos und der heute gebräuchliche Begriff **Softwareengineering** war noch nicht geprägt. Die erste Phase des Softwareerstellungsprozesses ist die Systemanalyse. Der Systemanalytiker beschreibt hier die für seine Fragestellung relevanten Elemente und deren Beziehungen zueinander.

Die ersten Rechner waren von den Abmessungen her groß und von der Leistungsfähigkeit aus heutiger Sicht sehr bescheiden. Hardware war so teuer, dass kleinere Unternehmen in der Regel die Verarbeitung ihrer Daten Service-Rechenzentren übergaben. Diese Rechenzentren entwickelten und warteten auch die individuellen Programme ihrer Kunden. Die eigene Datenverarbeitung im Hause bedeutete immense Investitionen, und die Software wurde dann mehr oder weniger individuell um die vorhandene Hardware „gestrickt“.

Die steigende Leistungsfähigkeit und der Preisverfall mit jeder neuen Generation von Rechnern eröffneten nach und nach immer neue Einsatzgebiete. So konnte man zunehmend integrierte Systeme entwickeln. Allerdings wurden mit dem wachsenden Integrationsgrad der Software die Programme und Programmsysteme komplexer.

Betrachtet man zu den Anfängen der Datenverarbeitung in mittleren bis großen Unternehmen das Verhältnis der Kosten von Hard- zur Software, so lag die bei etwa 85:15. Die gleiche Bewertung liefert heute ein Verhältnis von 10:90. Vergleicht man das Kostenverhältnis der Hard- zur Software im PC-Bereich, so ergibt sich für einen normalen Anwender in einem kleinen bis mittleren Betrieb ein ganz anderes Bild. Hier liegt das Verhältnis nahezu bei 50:50.

Der Einsatz von Datenverarbeitung in neuen Anwendungsgebieten ist primär ein Problem der Qualität, Funktionalität und Verfügbarkeit der Software zum richtigen Zeitpunkt und zu einem vertretbaren Preis. Damit wird deutlich, dass die Entwicklung von Software ein hochkomplexes Unterfangen ist und ein abgestimmtes, methodisches Verfahren und organisatorisches Vorgehen verlangt. Zusammengefasst wird dies unter dem Begriff Softwareengineering.

Softwareengineering wurde als Vorgehensweise zur Verbesserung der bis dahin unbefriedigenden Situation bei der Softwareentwicklung und -wartung betrachtet. Software sollte produziert werden können wie Produkte aus der industriellen Fertigung: solide, zuverlässig und kontrollierbar. Aus diesen Anfängen entwickelte sich die heutige Definition:



Unter **Softwareengineering** versteht man die Anwendung von Strategien, Methoden, Werkzeugen und Kontrollinstrumenten im gesamten Prozess der Softwareentwicklung und -wartung einschließlich des Managements.

Die Beschäftigung mit Softwareengineering setzt nun einen gewissen Erfahrungsschatz in der Softwareentwicklung voraus. Bei der **Softwareentwicklung im Kleinen** geht es um die Umsetzung überschaubarer Problemstellungen in rechnergestützte Lösungen. Dem Anwender der fertigen Software sollen möglichst viele, von ihm bisher evtl. mit anderen Hilfsmitteln erledigte Arbeitsschritte durch einen Rechner abgenommen werden. Dabei stehen die Auswahl und das Design einzelner Konstrukte im Vordergrund, was für die korrekte Funktionsweise und das spätere Verständnis eines Bausteins absolut wesentlich ist. Bei der **Softwareentwicklung im Großen** geht es um die zweckmäßige, fast generalstabmäßige Organisation eines Arbeitsvolumens von vielen Mann-Jahren. (In der Informatik wird der Begriff Mann-Tage, Mann-Monate oder Mann-Jahre als Aufwandsmaß eines abstrakten Wesens verwendet, das während seiner Arbeitszeit weder männlich noch weiblich ist.)

In manchem ist das Softwareengineering mit der Arbeitsorganisation in herkömmlichen Produktions- und Konstruktionsprozessen vergleichbar. Softwareengineering beschäftigt sich mit Arbeitsabläufen in und um die Softwareentwicklung herum. Neben dem eigentlichen Entwicklungsprozess sind dies:

- Projektmanagement,
- Qualitätssicherung und
- Projektverwaltung.



Unter **Projektmanagement** versteht man die Gesamtheit von Führungsaufgaben bei der Abwicklung eines Projekts, z. B. Fragen der Projektorganisation.

Bei der **Qualitätssicherung** geht es einerseits um formelle, konstruktive und analytische Kontrollmaßnahmen während des gesamten Entwicklungsprozesses, andererseits um interpersonelle Techniken, also darum, dafür Sorge zu tragen, dass alle Aufgaben von möglichst geeigneten Mitarbeitern erledigt werden.

Die **Projektverwaltung** (auch: Konfigurationsmanagement) beschäftigt sich mit der Bereitstellung und Verwaltung aller Ressourcen für den Softwareentwicklungsprozess sowie mit allen nebengelagerten Prozessen. Dazu gehören u. a. die Organisation der Speicherung aller Programmvarianten einschließlich der Dokumentationen sowie die notwendigen Update-Dienste.

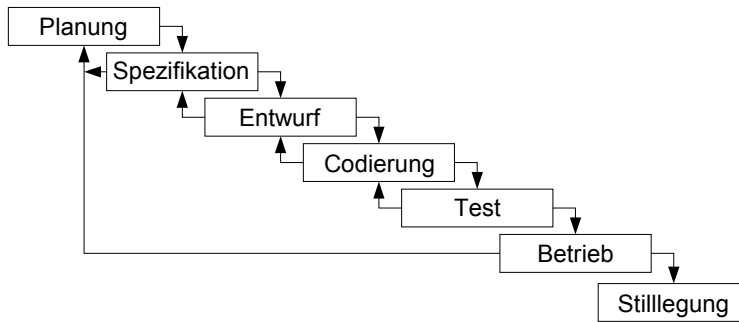
## ■ 1.2 Software-Lebenszyklus

Der Software-Lebenszyklus ist ein abstraktes Modell für den Lebenslauf einer jeden Software und die Grundlage für alle weiteren Betrachtungen zur Softwaretechnologie. Die meisten Aktivitäten, Methoden und Werkzeuge der Softwaretechnologie lassen sich anhand dieses Modells ein- und zuordnen. Für den konkreten Ablauf der Arbeit ist das Projektmanagement verantwortlich.

Der **Software-Lebenszyklus** stellt ein Modell für alle Aktivitäten während der Existenz einer Software dar. Man kann im Wesentlichen drei Teile unterscheiden:

- die eigentliche **Softwareentwicklung**, bei der das neue System aufgebaut wird;
- den **laufenden Betrieb**, währenddessen das System produktiv arbeitet, und
- die **Außerbetriebnahme** des Systems mit der Sicherstellung der Datenbestände für Nachfolgesysteme und der Entsorgung von Altdaten.

Während des laufenden Betriebs werden immer wieder ungeplante und geplante Unterbrechungen durch Wartung der eigentlich verschleißfreien Software erfolgen. Diese Wartungsarbeiten sind notwendig, um während des laufenden Betriebs festgestellte Fehler oder Effizienzverluste in den Programmen zu beheben oder die Software an geänderte Bedingungen des Umfeldes, in dem sie abläuft, anzupassen. Die Außerbetriebnahme einer Software erfolgt ebenso in der Regel aus dem laufenden Betrieb heraus. Schematisch lässt sich der **Software-Lebenszyklus** darstellen wie in Bild 1.1.



**Bild 1.1** Software-Lebenszyklus

Bei der Entwicklung eines Systems werden die Zyklen Planung bis Test als Abfolge von einzelnen Phasen durchlaufen. In jeder Phase können unterschiedliche Mitarbeiter an der Realisierung des Projektes beteiligt sein, die ihre Ergebnisse jeweils für die nächste Phase zur Verfügung stellen. Der Betriebszyklus umfasst während der gesamten Lebensdauer des Systems dessen Unterhalt und Weiterentwicklung bis zur Außerbetriebnahme des Systems. Betrachtet man nun die Kostenseite, so verursachen die ersten vier Zyklen etwa 40 % der Gesamtsystemkosten; die restlichen 60 % der Kosten entfallen auf den Betrieb des Systems.

Die einzelnen Zyklen lassen sich inhaltlich folgendermaßen beschreiben:

- Die **Planung** umfasst eine Voruntersuchung des künftigen Systems mit den entsprechenden Wirtschaftlichkeitsberechnungen und bildet die Entscheidungsgrundlage die Rechtfertigung und somit die Freigabe zur Entwicklung des neuen Systems. In der Praxis wird dazu zunächst eine Studie beauftragt, über deren Ergebnis ein sog. Lenkungsausschuss befindet.
- Bei der **Spezifikation** werden die wesentlichen Anforderungen und Leistungsparameter des neuen Systems festgelegt. Dies ist gleichzeitig der Zeitpunkt der Erstellung eines sog. Pflichtenheftes, das eine exakte Beschreibung des zu erstellenden Systems liefert und die Basis bildet für die Programmdokumentation und das Anwenderhandbuch.
- Der **Entwurf** des Systems schlüsselt die Anforderungen und Leistungsparameter schrittweise auf bis ein Detaillierungsgrad erreicht ist, bei dem die fachlichen Anforderungen und der fachliche Lösungsweg in Form von Elementarprozessen umfassend beschrieben sind. Am Ende müssen alle fachlichen und datenverarbeitungstechnischen (kurz: DV-technischen) Anforderungen festgelegt sein. Zu diesem Zeitpunkt ist eine umfassende Problemanalyse abgeschlossen, das Pflichtenheft liegt in seiner endgültigen Form vor und alle an der Erstellung der neuen Software beteiligten Personen verfügen über ausreichende Fachkenntnis, um den nächsten Schritt angehen zu können.
- Die **Codierung** umfasst die eigentliche Programmkonstruktion mit der Programmierung der neu zu erstellenden Software.
- Der **Test** dient der Aufdeckung von Entwurfs- und Codierungsfehlern. Werden Fehler entdeckt, so wird die Software zur Korrektur an die Codierungsphase zurückgewiesen. Lassen sich Fehler nicht lokal beheben, z. B. weil ihre Ursache bereits im Entwurf liegt, so wird die Software bis in die Entwurfsphase zurückverwiesen. Diese Testphase blockiert die weitere Entwicklung, bis eine sachlich und fachlich richtige Ausführung der einzelnen Programmkomponenten sowie des Gesamtsystems gesichert werden kann. Dabei

sollten Testhilfen eingesetzt werden, die sicherstellen, dass alle möglichen Fälle, die auftreten können, auch tatsächlich einmal durchlaufen worden sind.

- Der **Betrieb** einer Software wird bis zur Außerbetriebnahme immer wieder durch korrigierende oder geplante Wartung der Software unterbrochen. Das reicht von Eingriffen in die Konfigurationsdateien über das selektive Einspielen neuer Systemkomponenten (sog. Patches) bis hin zur Modifikation oder Neuentwicklung ganzer Systemteile. Besonders kritisch wird der Betrieb, wenn aus Sicherheitsgründen eine alte und eine neue Softwareversion parallel gefahren werden müssen.
- Bei der **Stilllegung** einer Software kommt es schließlich darauf an, wesentliche Nutzdaten sicherzustellen, die für die Konfiguration und Initialisierung von Nachfolgesystemen sonst erst aufwendig akquiriert werden müssten, möglicherweise datenschutzrelevante Daten zuverlässig aus dem System zu entfernen und alle Arten von Datenmüll zu beseitigen. Dies ist nicht nur eine Frage der vorbeugenden Hygiene im Rechnersystem, sondern wegen möglicher Fernwirkungen auf später zu installierende Software dringend notwendig.

## ■ 1.3 Software-Entwicklungsverfahren

Alle EDV-Projekte (EDV = elektronische Datenverarbeitung) haben einen typischen und gleichartigen im Software-Lebenszyklus bezeichneten Ablauf, der in einzelne Abschnitte unterteilt werden kann. Diese einzelnen Abschnitte oder Phasen lassen sich in einer sehr stark standardisierten Form darstellen und führen zu den Phasenmodellen. Prinzipiell kann jedes EDV-Projekt in zwei große Bearbeitungsbereiche, Entwurf und Realisierung, zerlegt werden. Jeder dieser beiden Blöcke muss für die weitere Bearbeitung in einzelne Abschnitte aufgesplittet werden. Ein Phasenmodell entsteht im Prinzip durch genaue Definition und Abgrenzung der einzelnen Abschnitte des Software-Lebenszyklus.

Eine zu grobe Unterteilung der einzelnen Phasen lässt einen großen Spielraum innerhalb der einzelnen Phase zu und erhöht damit die Fehlerwahrscheinlichkeit. Eine zu feine Unterteilung der Phasen verzögert die Bearbeitung wegen der häufigen Unterbrechungen durch externe Entscheidungen. Sinnvolle Phasenmodelle unterscheiden zwischen drei und sechs Phasen, in Abhängigkeit vom Projektumfang. Hier soll von einem 6-Phasenmodell wie in Bild 1.2 ausgegangen werden.

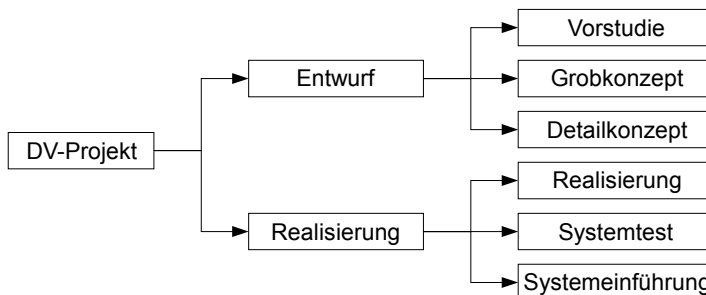


Bild 1.2 Das 6-Phasenmodell

Die einzelnen Phasen lassen sich wie folgt beschreiben:

- Die **Vorstudie** ist ein Abklärungsprozess, dem unmittelbar eine Entscheidung bezüglich der möglichen Lösungsvarianten folgt. Dabei wird die Zielrichtung für die Gestaltung des neuen Projektes festgelegt. Folgende Punkte müssen in einer Vorstudie enthalten sein:
  - Beschreibung der Ausgangslage und Begründung für die Entwicklung einer neuen Lösung
  - Konkrete Zielvorstellung
  - Vollständige Beschreibung des Ist-Zustandes und Schwachstellenanalyse
  - Vor- und Nachteile der heutigen Lösung mit Schwachstellenbeschreibung
  - Gestellte Anforderungen und Wünsche an die neue Lösung
  - Beschreibung der Lösung mit möglichen Alternativen
  - Bewertung der Lösung und der möglichen Alternativen
  - Wirtschaftlichkeitsüberlegungen
  - Planung und Freigabe der nächsten Phase
- Auf der Basis der in der Vorstudie favorisierten Lösungsmöglichkeit muss eine generelle Lösung mit den möglichen Varianten in einem betrieblichen und DV-technischen **Grobkonzept** erarbeitet werden. Die Lösung muss hier so detailliert sein, dass eine fachliche und sachliche Beurteilung und Bewertung möglich ist. Inhalt dieser Phase ist:
  - EDV-technische Konzeption der Funktionen, Abläufe, Transaktionen, Datenstrukturen, Festlegung der Verarbeitungsmodalitäten und des weiteren Vorgehens
  - Betriebliche Konzeption der Funktionen, Abläufe, Transaktionen, Layouts, Ausfallverfahren, Verarbeitungsmodalitäten und weiteres Vorgehen
  - Definition der betrieblichen Einführungsstufen
  - Test- und Einführungskonzeption
  - Überprüfung der Lösung
  - Wirtschaftlichkeitsberechnungen
  - Planung und Freigabe für die nächste Phase
- In der Phase **Detailkonzept** ist das komplette fachliche und technische Systemdesign definitiv und abschließend zu erarbeiten. Ungelöste Probleme sind in dieser Phase nicht mehr zulässig. Die EDV-technische und betriebliche Machbarkeit muss sichergestellt sein.
  - Detaillierung und Komplettierung der EDV-technischen Konzeption
  - Detaillierung und Komplettierung der betrieblichen Konzeption
  - Detaillierung und Komplettierung der betrieblichen Einführungsstufen
  - Detaillierung und Komplettierung der Test- und Einführungskonzeption
  - Überprüfung aller Konzeptionen
  - Wirtschaftlichkeitsberechnungen
  - Planung und Freigabe für die nächste Phase
- Die Phase **Realisierung** stellt die reine Umsetzung der erstellten Konzeption in Programme dar. Zu diesem Zeitpunkt muss die komplette Dokumentation, wie Benutzerhandbücher und Operatorhandbuch, vorliegen.

# Index

## A

- Abfrage 75
- abgeleitete Klasse 228
- Ablauflinie 152
- abstrakte Klasse 217
- Abstraktion 214
- abweisende Schleife 132
- Achsenbeschriftung 259
- Addition 65
- Adressoperator 102, 103
- Aggregation 228
- Aggregatobjekt 228
- Algorithmus 159, 178, 218
- Alternation 75, 120, 123, 124, 136, 141, 150
- Anpassungshinweise 157
- Anweisungen 75
- arithmetischer Operator 65
- Array 56, 63
- ASCII-Tabelle 348
- ASCII-Zeichensatz 55
- Assoziation 228
- Assoziativität 74
- Attribut 216, 218, 302
- Attributwert 217, 253
- Aufgabe 156
- Aufgabenlösung 155
- Aufgabenstellung 155
- Ausgabesymbol 152
- Ausgleichsrechnung 317
- Ausnahmefehler 253
- Außerbetriebnahme 19
- Aussprung 121, 135
- Auswahl 114, 138
- auto 343

## B

- Basisklasse 228
- Batchdatei 104
- Bedienung 157
- Bedingungsoperator 72
- Bemerkung 152
- Betrieb 19, 21
- Bezeichnung 156
- Bibliotheksfunktionen 96
- Bildpunkt 244
- binäre Operatoren 65
- Blöcke 75
- bool 346
- Boole 327
- Boole, Georg 327
- break 78, 86, 127, 343
- Brückenkonstruktion 186

## C

- call by reference 99
- call by value 97
- case 343
- case-sensitiv 52
- Cast 236
- catch 346
- char 52, 63, 343
- class 346
- Codierung 20, 25
- const 343
- const\_cast 346
- Container-Klasse 234
- continue 85, 343
- Copy-and-paste 89

**D**

Datentyp 51  
 Datentypen und ihre Wertebereiche 341  
 Debugger 154  
 default 126, 343  
 Default-Werte 259  
 Definition der Funktion 92  
 Dekrementierung 68  
 delete 346  
 Dereferenzierung 102, 104  
 Designfehler 31  
 Deskriptoren 157  
 destruktives Schreiben 110  
 Destruktor 222  
 Detailkonzept 22, 24  
 deterministische Verfahren 160  
 Differenzialrechnung 314  
 Digitalschaltung 329  
 Digitaltechnik 327  
 DIN 66001 151  
 DIN 66230 155  
 Division 66  
 do 343  
 Dokumentation 115, 138, 155, 221  
 double 54, 343  
 do - while 84  
 Dualzahlen 330, 331  
 dynamic\_cast 346

**E**

Effektivwertberechnungen 303  
 eindimensionale Felder 56  
 einfache Abfrage 75  
 Eingabesymbol 152  
 Elemente eines Feldes 59  
 Elementverweis-Operator 235  
 else 344  
 else - if 76  
 Endlosschleife 81, 127, 134  
 Energiewirtschaft 196  
 EN-ISO 9001 158  
 Entität 216  
 Entwurf 20, 21, 24  
 enum 344  
 E-Reihen 260  
 erweiterte Schlüsselwörter C++ 345  
 euklidischer Algorithmus 164  
 EVA-Prinzip 165  
 Exception 253  
 Exemplar 216  
 explicit 346  
 explizite Typumwandlungen 236

extern 344, 346  
 externe Operation 219  
 Extremstellen 314

**F**

Fakultätsberechnung 160  
 false 346  
 Farbcodierung nach DIN 41429 263  
 fclose() 111  
 Fehler 31  
 Fehlerbehandlung 157  
 Fehlerquadratsumme 318  
 Feld 56, 57, 58  
 fflush() 113  
 fgetc() 110  
 fgets() 110  
 FILE 109  
 Fließkommazahl 54  
 float 54, 60, 344  
 Flussdiagramm 151, 153  
 Font 257  
 fopen() 110  
 for 79, 344  
 formatierte Ausgabe 108  
 formatierte Eingabe 107  
 Formelwerk 166, 167  
 Fourier  
 - Analyse 274  
 - Koeffizient 274  
 - Reihen 267  
 - Synthese 266  
 fprintf() 111  
 fputc() 110  
 fputs() 110  
 fread() 111  
 friend 346  
 fscanf() 111  
 Funktion 88  
 Funktionen  
 - aufruf 120, 134, 135, 250  
 - graph 255  
 - kopf 89, 90  
 - rumpf 89, 91  
 fußgesteuerte Schleife 84, 127, 133, 134  
 fwrite() 111

**G**

ganzzahlige Variablen 52  
 Geheimnisprinzip 217  
 Gerätebedarf 156  
 get-Methode 222



Gleichheit 68  
goto 344  
Graphical User Interface (GUI) 234  
Grenzstelle 152  
Grobkonzept 22  
größer 69  
größer gleich 70  
GUI 36, 214, 234

## H

Hauptprojektdatei 94  
Header-Datei 94, 350  
Heißeleiter 206, 213

## I

if 344  
if - else 75  
Implementierung 153  
Implementierungsaufwand 89  
implizite Typumwandlung 236  
Include 350  
Indizierungsoperator 73  
Initialisierung 79  
Inkludierung 96  
Inkrementierung 67  
Inkrementverfahren 279  
inline 346  
Instanz 216, 234  
int 52, 53, 56, 57, 344  
IntelliSense 237, 238  
Intervallhalbierung 280  
Iteration 79, 128, 150

## K

Kapselung 216, 221  
Kennlinie 258  
Kennlinienfeld 258  
Klammerungsoperator 73  
Klassen 215, 217, 314  
- beschreibung 217  
- diagramm 217  
- hierarchie 228  
- operation 219  
kleiner 69  
kleiner gleich 69  
kombinierte Zuweisung 67  
Kommentar 50  
Komponente 32, 228  
Komposition 228  
Konstante 52, 57, 64

Konstruktor 219, 221  
Kontrollstruktur 75  
Koordinatentransformation 245  
kopfgesteuerte Schleife 82, 83, 132, 134  
Kosinus-Koeffizient 274

## L

least square fit 318  
Leibniz 330  
logarithmische Achsenteilung 206, 207  
logischer Fehler 153  
logisches NICHT 70  
logisches ODER 70  
logisches UND 70  
long 52, 53, 108, 344  
long int 52

## M

malloc() 181  
mehrdimensionale Felder 57  
Mehrfachabfrage 76  
Mehrfachauswahl 345  
Message-Box 253  
Messreihe 317  
Methode 216, 218, 257, 259  
Methode überladen 219  
mode 110  
Modellbildung 214  
Modul 89  
Modulo-Operator 66  
Multiplikation 66  
mutable 347

## N

nachprüfende Schleife 133  
namespace 347  
Nassi-Shneidermann 119, 153  
new 347  
Newton-Verfahren 286  
nicht-abweisende Schleife 133  
NTC 206  
Nullstellenbestimmung 279  
numerische Integration 289  
Nutzungsvereinbarungen 157

## O

Obersumme 291  
Objekt 215, 216, 253  
Objektoperation 219

objektorientierte Programmierung (OOP) 114,  
152, 214  
OOP 112  
Operation 218  
Operator 65, 347  
Ordinalwert 55, 56  
Overloading 219

## P

Parabel 255  
Paradigma 114  
Parameterübergabe 97  
Planung 20  
Pointer 102  
Postfix 67  
Potenzfunktion 125  
pow( ) 125  
Präfix 67  
Präprozessoranweisung 96  
printf() 107, 130  
Priorität 74  
private 218, 221, 229, 347  
Problemanalyse 115, 116, 166, 171  
Problemstellung 115, 171  
Programm 171  
- ablauf 154, 156, 236  
- ablaufplan 151  
- aufbau 156  
- aufruf 120  
- bedarf 156  
- erstellung 115, 153  
- lauf 115, 142, 154, 171  
- test 171, 181  
Programmierparadigma 114  
Programmiersprache 157  
Projektmanagement 18, 19  
Projektverwaltung 18  
protected 218, 222, 347  
Protokoll einer Klasse 218  
Prototyp 93  
Prozeduren 89, 114  
public 218, 221, 229, 347

## Q

Qualitätsmanagement 158  
Qualitätssicherung 18, 159

## R

Realisierung 21, 22, 24  
Rechteckfunktion 268, 276

Referenz 99, 101  
register 344  
regula falsi 284  
Reinitialisierung 80  
reinterpret\_cast 347  
Rekursion 104  
return 344  
Riemannsche Unter- und Obersummen 289  
Rundungsfehler 55

## S

scanf( ) 107, 132  
Schaltjahrüberprüfung 190  
Schaltnetz 328  
Schleifen 114, 120, 127, 144, 149, 192  
- abbruch 85, 86  
- bedingung 82  
- begrenzungssymbol 152  
- kopf 82  
- rumpf 82, 83, 84  
- steuerungsvariable 81  
- umwandlung 88  
Schlüsselwörter ANSI C 343  
Schnittstelle 156, 218  
Schriftart 257  
Schrittweitenwert 255  
Schwerpunktkoordinaten 184  
Screenkoordinaten 243  
Sequenz 114, 120, 135, 139  
set-Methode 222  
Shannon, Claude E. 327  
short 52, 53, 108, 344  
short int 52, 54, 328  
signed 53, 54, 344, 345  
Signifikanz 54  
Simpsonsche Regel 298  
Sinusfunktion 241, 243, 251  
Sinus-Koeffizient 274  
sizeof 344  
Skalierung 255  
Softwareengineering 17  
Softwarelebenszyklus 19  
SolidBrush 257  
Spannungsteiler 258  
Spannungsteilerkennlinie 258  
Spannungsteilerwiderstand 258  
Speicherbedarf 156  
Speicherberechnungsoperator 71  
Spezifikation 20  
sprachbedingte Fehler 31  
sqrt( ) 125, 296  
Standardeingabe 83

- Standardfunktionen 350
- Stapelverarbeitungsdatei 104
- Startbedingung 81
- Startwertzuweisung 61
- static 344
- static\_cast 347
- stdin 83
- stdio.h 109
- Stilllegung 21
- STL 234, 235
- Stream 109
- string 194, 235
- String 234
- struct 344
- Struktogramm 119, 151, 171
- Struktur 109, 218, 320
- strukturierte Programmierung , 14
- Subtraktion 65
- switch 345
- switch-case 77
- syntaktischen Fehler 153
- Syntax 50, 61
- Systemeinführung 23
- Systemtests 23

## T

- Tagesbelastungskurve 196
- Temperatursysteme 171
- template 347
- Template-Klasse 234
- Test 20, 115, 155, 157
- Test des Programms 115
- TextBox 235, 302
- Textstrom 109
- this 347
- throw 347
- Top-down-Verfahren 119
- Trapezregel 293
- Treffpunktkoordinaten 185
- true 348
- try 348
- try-catch 235
- typecast 71
- Typecasting 54
- typedef 345
- typeid 348
- typename 348
- Typkonvertierung 236
- Typmodifizierer 53
- Typumwandlung 236, 257
- Typumwandlungsoperator 71

## U

- Übergangsstelle 152
- überladene Methode 219, 238
- UML 217, 222
- unäre Operatoren 65
- Ungleichheit 68
- union 345
- unsigned 52, 54, 108
- Untersumme 290
- using 348

## V

- Variable 51, 52, 63
- Vererbung 228, 240
- Vergleichsoperatoren 68
- Verhalten einer Klasse 217
- Verschachtelung 120, 124, 139
- Verzweigung 114, 120, 125, 126
- Verzweigungssymbol 152
- virtual 348
- Visual C++ 13
- void 90, 345
- volatile 345
- Volladdierer 334
- vorprüfende Schleife 132
- Vorstudie 22
- Vorzeichenoperator 65

## W

- Wahrheitstabelle 333, 339
- Wartbarkeit 89
- wchar\_t 348
- Weltkoordinaten 243
- Wendestellen 314
- while 83, 345
- Whitespace 108
- Widerstandsreihe 260
- Wiederanlaufverfahren 157
- Wiederholung 79, 114, 129
- Wiederverwendbarkeit 89

## Z

- zählergesteuerte Schleife 79, 82, 127, 145
- Zeichen 55
- Zeiger 102
- Zuweisung 66
- Zwei-Punkte-Form 247