norbert HEIDERICH
wolfgang MEYER

3., neu bearbeitete Auflage

Technische Probleme lösen mit C/C++

VON DER ANALYSE BIS ZUR DOKUMENTATION

Im Internet: Alle Codebeispiele zum Buch

HANSER

Heiderich/Meyer

Technische Probleme lösen mit C/C++

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter

www.hanser-fachbuch.de/newsletter

Hanser Update ist der IT-Blog des Hanser Verlags mit Beiträgen und Praxistipps von unseren Autoren rund um die Themen Online Marketing, Webentwicklung, Programmierung, Softwareentwicklung sowie IT- und Projektmanagement. Lesen Sie mit und abonnieren Sie unsere News unter

www.hanser-fachbuch.de/update

Lernbücher der Technik

herausgegeben von Dipl.-Gewerbelehrer Manfred Mettke, Oberstudiendirektor a. D.

Bisher liegen vor:
Bauckholt, Grundlagen und Bauelemente der Elektrotechnik

Felderhoff/Busch, Leistungselektronik

Felderhoff/Freyer, Elektrische und elektronische Messtechnik

Fischer/Hofmann/Spindler, Werkstoffe in der Elektrotechnik

Freyer, Nachrichten-Übertragungstechnik

Knies/Schierack, Elektrische Anlagentechnik

Schaaf/Böcker, Mikrocomputertechnik

Seidel/Hahn, Werkstofftechnik

HANSER

Norbert Heiderich • Wolfgang Meyer

Technische Probleme lösen mit C/C++

Von der Analyse bis zur Dokumentation

3., neu bearbeitete Auflage

Mit 248 Bildern, 76 Beispielen und zahlreichen Listings

HANSER

Autoren

Dipl.-Math. Norbert Heiderich, Berufskolleg des Kreises Kleve, Kleve

Dipl.-Ing. Dipl-Ing. Wolfgang Meyer, Heinz-Nixdorf-Berufskolleg, Essen

Alle in diesem Buch enthaltenen Programme, Verfahren und elektronischen Schaltungen wurden nach bestem Wissen erstellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund ist das im vorliegenden Buch enthaltene Programm-Material mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials oder Teilen davon entsteht.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung – mit Ausnahme der in den §§ 53, 54 URG genannten Sonderfälle –, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2016 Carl Hanser Verlag München

Lektorat: Mirja Werner
Herstellung: Franziska Kaufmann
Satz: Kösel Media GmbH, Krugzell
Coverconcept: Marc Müller-Bremer, www.rebranding.de, München
Coverrealisierung: Stephan Rönigk
Druck und Bindung: Kösel, Krugzell
Printed in Germany

ISBN: 978-3-446-44784-4
E-Book-ISBN: 978-3-446-44905-3

www.hanser-fachbuch.de/computer

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

Einleitung

Bücher, die sich mit Entwicklungsumgebungen beschäftigen, gibt es viele. Ebenso gibt es unzählige Werke, die die Programmiersprachen C und C++ beschreiben und sich mit den Vorteilen der einen gegenüber der anderen auseinandersetzen. Lehrbücher, die dem Leser den Weg vom konkreten Problem über Lösungsstrategien und Dokumentationshilfen bis hin zur rechnerunterstützten fertigen Lösung aufzeigen, sind hingegen Mangelware.

Niemand fängt bei dem Versuch, mit einer neuen Programmiersprache zu arbeiten, gerne ganz von vorne an. Daher werden Sie in diesem Buch Codebeispiele und exemplarische Vorgehensweisen finden, die Ihnen den Einstieg in die C/C++-Programmierung erleichtern. Das Buch möchte Sie als Leser bei der Lösung und Bearbeitung Ihrer Probleme unterstützen, Ihnen anhand vieler Beispiele unterschiedliche Einsatzmöglichkeiten von Programmier- und Dokumentationstechniken näherbringen, um Ihnen Schritt für Schritt den Weg in die professionelle Softwareentwicklung aufzuzeigen.

Zur Realisierung der Beispiele haben wir uns entschieden, mit Visual C++ (kurz: VC++) der Firma Microsoft zu arbeiten. Zum einen liegt mit VC++ ein hochmodernes Werkzeug zur effizienten Erstellung von Software jeden Anwendungs- und Komplexitätsgrades vor. Zum anderen ist die Verbreitung von VC++ so groß, dass die Wahrscheinlichkeit, dass Sie im professionellen Einsatz mit dieser Entwicklungsumgebung umgehen werden, sehr hoch ist. Darüber hinaus ist die Verfügbarkeit von VC++ als kostenloser Download der Firma Microsoft (siehe https://www.visualstudio.com/downloads) ein nicht zu unterschätzender Vorteil.

Grundsätzlich ist es natürlich auch möglich, eine andere Entwicklungsumgebung zu nutzen (z. B.: Code::Blocks, Eclipse, NetBeans IDE u. a.), allerdings können dann die von uns bereit gestellten Beispiele nicht ohne Anpassungen genutzt werden. Sie müssen in solchen Fällen die jeweiligen Quellcodes in Ihre Entwicklungsumgebung übertragen. Die Nutzung unserer Beispiele mit grafischen Oberflächen entfällt vollständig. Die entsprechenden Layouts müssen dann in Ihrer Entwicklungsumgebung nachempfunden werden. Alle relevanten Ideen können natürlich aus unseren Quellcodes ausgelesen werden.

Aufbau des Buches

Das Buch ist in zehn Kapitel gegliedert, die jeweils ein spezielles Thema zum Inhalt haben. Bei der Auswahl der umgesetzten Beispiele haben wir versucht, sowohl technisch-wissenschaftliche, mathematische als auch kommerzielle Probleme zu thematisieren, um ein mög-

lichst großes Spektrum von Fragestellungen und damit verbundenen Problemlösungen aufzuzeigen.

Die Kapitel werden im Folgenden kurz skizziert:

Kapitel 1
Im ersten Kapitel stehen Gesichtspunkte, die in einer hochvernetzten und globalisierten Arbeitswelt den Umgang mit zu lösenden Problemen diktieren, im Fokus der Betrachtung. Längst sind die Zeiten vorbei, in denen sich ein Softwareentwickler so intensiv mit seinem Problem auseinandersetzen konnte, bis er sicher war, eine optimale und anwendungsbreite Lösung gefunden zu haben. Die Maxime heute lautet: in kürzester Zeit ein brauchbares Ergebnis zu liefern.

Kapitel 2
Dieses Kapitel beschäftigt sich mit einigen grundlegenden Aspekten der Programmiersprache C/C++. Neben der Frage, warum es sinnvoll ist, gerade mit C/C++ zu arbeiten, werden Funktionsweisen der Komponenten der Entwicklungsumgebung betrachtet und erläutert.

Kapitel 3
Im dritten Kapitel wird die Arbeitsweise von VC++ und die Arbeit mit dieser Entwicklungsumgebung beschrieben. Der Leser erfährt, wie ein komplexeres Projekt organisiert wird.

Auf Installationhinweise auf zur jeweils aktuellen Version von VC++ haben wir absichtlich verzichtet, da Microsoft in sehr kurzen Zyklen neue Versionen zur Verfügung stellt, deren Installationen keinerlei Probleme darstellen. Die Kompatibilität von eingesetztem Betriebssystem und verwendeter Version der Entwicklungsumgebung von VC++ wird auf den Internet-Seiten von Microsoft dargestellt (s. a. https://msdn.microsoft.com/de-de).

Kapitel 4
In diesem Kapitel werden die grundlegenden Sprach- und Steuerungselemente der Syntax der Programmiersprache C/C++ an einfachen Beispielen dargestellt.

Kapitel 5
Dieses Kapitel enthält eine Einführung in die strukturierte Programmierung und ihre Darstellungsformen. Der Leser lernt rechnergestützte Systeme zur Erstellung von Struktogrammen kennen, die an Beispielen zunehmender Komplexität beschrieben werden. Außerdem werden die Bestandteile einer Software-Dokumentation beschrieben und die Frage beantwortet, was Software mit Qualität zu tun haben sollte.

Kapitel 6
Im sechsten Kapitel werden die Kenntnisse der strukturierten Programmierung zunächst an einfachen Problemen angewendet. Von der Problemanalyse bis zur Ergebnisausgabe der Programme sind die Beispiele durchgängig dokumentiert.

Kapitel 7
Dieses Kapitel gibt eine Einführung in die objektorientierte Programmierung. Der Leser lernt das erweiterte Vokabular und die Techniken der OOP kennen.

Kapitel 8
Mit den Grundkenntnissen der OOP können in diesem Kapitel fortgeschrittene Probleme unter Zuhilfenahme von grafischen Oberflächen gelöst werden. Die Darstellung der Ergebnisse geschieht hier teilweise auch in zeichnerischer Form.

Kapitel 9
Im neunten Kapitel werden komplexe Fragestellungen durch konsequente Anwendung aller erlernten Techniken bearbeitet. Die erzielten Lösungen sind dabei nach entsprechend gründlicher Problemanalyse verblüffend einfach.

Kapitel 10
In Kapitel 10 finden Sie Tabellen und Übersichten, es stellt eine hilfreiche Zusammenfassung der für die Problemlösung erforderlichen Teilgebiete dar. Neben Datentypen, Symbolen für die Erstellung von Struktogrammen und Programmablaufplänen enthält es ein Glossar für den Sprachumfang von C und C++ und eine Tabelle oft genutzter Standardfunktionen.

Außerdem finden Sie im Internet alle Code-Beispiele der Kapitel 6 bis 9 unter: *http://www.hanser-fachbuch.de/buch/Technische+Probleme+loesen+mit+C+C/9783446447844* auf der Homepage des Hanser Verlags. Die Beispiele sind hier sowohl als PDF-Dateien mit den durchnummerierten Zeilen, auf die in den Erläuterungen verwiesen ist, als auch als Quelltexte, die Sie ohne mühsames Abtippen sofort in eigene Programme integrieren können, vorhanden. So lassen sich alle Beispiele unmittelbar erzeugen und ausführen. Einige Beispiele der Kapitel 8 und 9 finden Sie aufgrund ihrer Länge nur im Internet.

Zielgruppe des Buchs:
Das Buch wendet sich in erster Linie an Studierende an Fachschulen, Studenten technischer Studiengänge sowie an Auszubildende in den IT-Berufen. Aber auch Schülerinnen und Schüler in technisch orientierten Gymnasien und Fachoberschulen werden an der Vielfalt der Problemstellungen und der Herangehensweise an die Lösung – vom Problem über die Problemanalyse mit Struktogramm bis zum Testing und zur Dokumentation – sicherlich Gefallen finden.

Es liegt ein zeitgemäßes Buch vor. Zeitgemäß deshalb, weil wir den wichtigsten Schritt in der Programmentwicklung, die Problemanalyse, in den Mittelpunkt der Entwicklungsarbeit stellen und erst danach die Umsetzung mit VC++ besprechen. Denn ohne eine gründliche Problemanalyse haben Sie später keine Chance, logische Programmfehler zu finden. Übrigens verdient man in der Wirtschaft in diesem Bereich das meiste Geld.

Zeitgemäß ist das Buch auch dadurch, dass Sie ellenlange Quellcodes nicht mehr abtippen müssen, sondern einfach aus dem Internet herunterladen können. Das vereinfacht den Programmtest und die individuelle Fortentwicklung der Programme ganz gewaltig.

Um Ihnen die Orientierung im Buch zu erleichtern, haben wir Icons verwendet, die folgende Bedeutung haben:

	Bei den **Beispielen** im Buch finden Sie dieses Symbol.
	Wichtige **Hinweise** werden durch dieses Icon kenntlich gemacht.
	Dieses Symbol kennzeichnet **Aufgaben**.
	Die Darstellungen von **Lösungen** sind an diesem Icon zu erkennen.
	Hinweis auf einen Teil im **Internet**.

Zum Abschluss bleibt uns noch all den Personen zu danken, die uns bei der Arbeit an diesem Buch unterstützt haben. Ihnen als Leserin und Leser wünschen wir viel Erfolg bei der Arbeit mit diesem Buch und dem Lernen und Ausprobieren mit und von Visual C++.

Norbert Heiderich und Wolfgang Meyer

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 generalstabsmäßige 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.

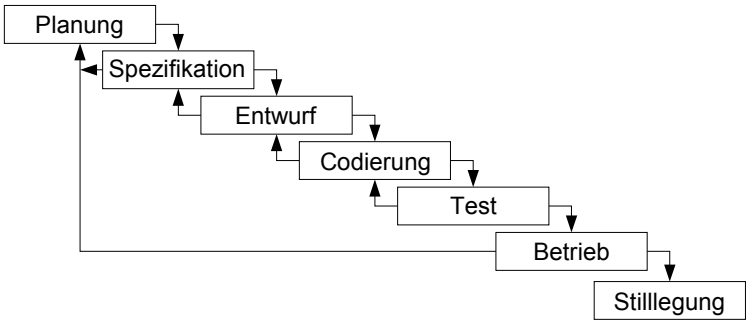

Bild 1.1 Software-Lebenszyklus

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.

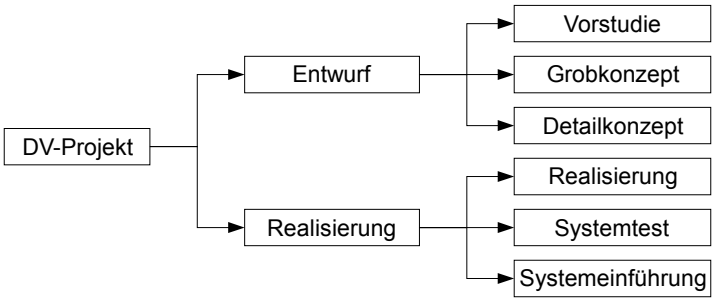

Bild 1.2 Das 6-Phasenmodell

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.

- Erstellung der Programme
- Erstellung der JOB-Control/Shell-Skripte
- Einzel- und Integrationstest der Programme
- Erstellung des Einführungsplans
- Planung und Freigabe für die nächste Phase

- Die realisierten Teile aus der vorhergegangenen Phase werden während des **Systemtests** auf ihre Richtigkeit und Vollständigkeit unter betrieblichen Gesichtspunkten überprüft. Es muss dabei vorausgesetzt werden, dass die einzelnen Komponenten bereits für sich alleine ausführlich getestet und abgenommen worden sind. Alle durchgeführten Tests müssen dokumentiert werden:
  - Dokumentation der Testergebnisse
  - Dokumentation des betrieblichen Tests
  - Einzel- und Integrationstest der Programme
  - Überarbeitung des Einführungsplans
  - Planung und Freigabe für die nächste Phase
- In der Phase **Systemeinführung** wird das fertiggestellte und komplett abgenommene System in die laufende EDV-Produktion integriert. Ein abschließender Funktionstest mit dem GO-Entscheid stellt allen involvierten Benutzern das neue System zur Verfügung.
  - Systemeinführung und Pilotbetrieb
  - Systemübergabe in die laufende Produktion
  - Konsolidierung und Optimierung des Systems
  - Überarbeiten des Einführungsplans
  - Planung und Freigabe für die nächste Phase

Den Ablauf des klassischen 6-Phasenmodells kann man wie in Bild 1.3 zeitlich darstellen:

Jede Phase schließt normalerweise mit einer Freigabe für die nächste Phase ab.

Wird die Freigabe nicht erteilt, sind solange Korrekturen innerhalb der aktuellen Phase notwendig, bis eine Freigabe dieser Phase erfolgt (Ablaufpfeile rechts).

Bei sehr gravierenden Fehlern muss im Extremfall mehrere Phasen zurückgesprungen werden, um diese Fehler zu bereinigen. Unter Umständen kann dies sogar zu einem Abbruch des Projektes führen (Ablaufpfeil links).

Theoretisch besteht in einem solchen Fall (Abbruch) natürlich die Möglichkeit, bis zum Projektstart zurückzuspringen, aber in der Praxis sind in aller Regel bis zu einem Abbruchzeitpunkt bereits erhebliche Kosten verursacht worden, ohne dass entsprechende Fortschritte erzielt wurden, so dass die Frage, ob ein solches Projekt noch wirtschaftlich sinnvoll weitergeführt werden kann eher zu verneinen sein wird.

Die Entscheidung über mögliche weitere Vorgehensweisen bei jedem Rücksprung trifft in jedem Fall der Lenkungsausschuss, dem von den beteiligten Partnern die Personen angehören, die die wirtschaftlichen Entscheidungen treffen können.

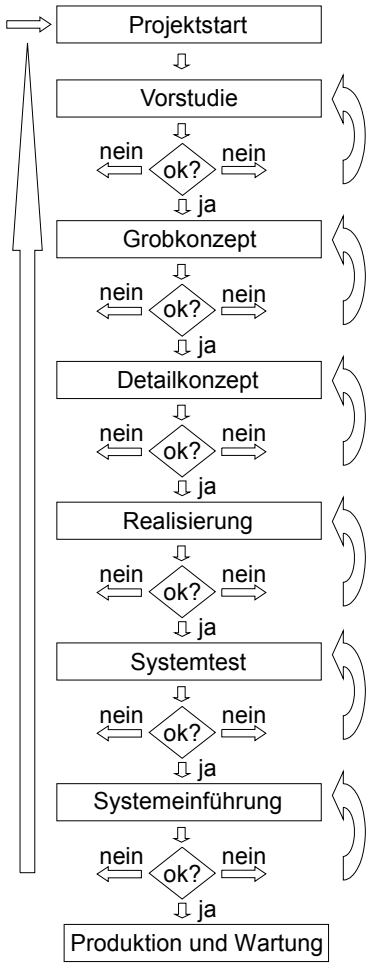

Bild 1.3
Ablauf des 6-Phasenmodells

Für die Bearbeitung der einzelnen Phasen sollte in etwa mit folgenden prozentualen Anteilen des Gesamtentwicklungsaufwands kalkuliert werden:

Entwurf		**Realisierung**	
Vorstudie	10 %	Realisierung	25 %
Grobkonzept	25 %	Systemtest	10 %
Detailkonzept	20 %	Systemeinführung	10 %
Summe Anteil	**55 %**	**Summe Anteil**	**45 %**

Aus dieser Aufstellung der Aufwände der einzelnen Phasen bei der Umsetzung eines Softwareprojektes wird sehr deutlich, dass die Vorarbeiten, die der eigentlichen Programmierung vorausgehen, den größten Teil (nämlich 55 %) des Aufwands des Gesamtprojekts einnehmen. Nur, wenn diese Arbeiten mit der gebotenen und erforderlichen Sorgfalt durch-

geführt werden, besteht die Aussicht, dass das Projekt in der Praxis auch Bestand haben wird. Und das ist ja schließlich das Ziel der Umsetzung eines jeden Ablaufes in ein „Stück“ Software!

Die eigentliche Codierung des Programms, also die Realisierung, nimmt gerade ein Viertel des zu kalkulierenden Aufwands ein. Dieser Ansatz kollidiert in der Praxis oftmals mit dem Anspruch der beteiligten Programmierer, eine „perfekte“ Lösung umzusetzen und an der einen oder/und (!) anderen Stelle über das in der Entwurfsphase festgelegte und mit dem Auftraggeber abgesprochene Leistungsmaß der zu entwickelnden Software hinauszuschießen. Hier besteht die Aufgabe des Projektmanagements darin, solche vermeintlichen Leistungssteigerungen frühzeitig zu erkennen und in angemessener Form zu reagieren. Ein erfahrener Projektmanager wird die beteiligten Programmierer so frühzeitig wie möglich an der Entwurfsphase teilhaben lassen, um ihren Erfahrungen und Bedenken schon in dieser Phase Rechnung tragen zu können. Andererseits kann ein detailverliebter Programmierer eine wichtige Besprechung zwischen Auftraggeber und -nehmer durch seine Rolle als Bedenkenträger natürlich auch völlig aus dem Ruder laufen lassen, indem er das Gesamtprojekt ausschließlich aus seinem Blickwinkel als der, der für eine lauffähige Umsetzung zu sorgen haben wird, betrachtet, ohne den nötigen und oftmals erforderlichen Abstand zu den technischen Anforderungen zu besitzen – also eine echte Gradwanderung für das Projektmanagement. Werden die Programmierer nämlich erst sehr spät mit dem Projekt vertraut gemacht, kann nur noch sehr schwer auf tatsächliche Probleme, die aus der technischen Umsetzung resultieren, reagiert werden. Und das kann natürlich eine reibungslose und termingerechte Umsetzung eines Datenverarbeitungsprojektes massiv gefährden.

2 Erste Gehversuche mit C/C++

Dieses Kapitel beschäftigt sich mit einigen grundlegenden Aspekten der Programmiersprache C/C++. Neben der Frage, warum es sinnvoll ist, gerade mit C/C++ zu arbeiten, werden Funktionsweisen der Komponenten der Entwicklungsumgebung betrachtet und erläutert. In den folgenden Kapiteln werden zunächst Beispiele in klassischem C als Konsolenanwendungen realisiert, bevor später objektorientiert mit C++ weitergearbeitet wird. Dann sind die Beispiele auch mit grafischen Oberflächen ausgestattet.

2.1 Warum gerade C/C++?

Wer C/C++ erlernen will, hat sich für eine Programmiersprache entschieden, die auf fast allen Rechnertypen und unter fast allen Betriebssystemen verfügbar ist. Es steht Ihnen, anders als bei vielen anderen Programmiersprachen, auf den verschiedensten Entwicklungsplattformen eine genormte Standard-Bibliothek zur Verfügung. Damit gelingt eine einheitliche Implementierung der mit dieser Programmiersprache erstellten Programme mit sehr hoher Geschwindigkeit.

C wird auch als Highlevel-Assembler bezeichnet, also als Programmiersprache, die sehr nah an der Maschinensprache ist. Dies beruht auf der Tatsache, dass der Kern (bzw. Kernel) aller gängigen Betriebssysteme in C geschrieben wurde. Damit eignet sich C/C++ auch in besonderem Maße für die Systemprogrammierung, also für Programme, die für den Betrieb von Rechenanlagen erforderlich sind.

Dank der relativ einfachen Struktur und dem geringen Umfang der eigentlichen Sprache, d. h. der verfügbaren Schlüsselworte der Programmiersprache, war es möglich, **C-Compiler**, also spezielle Programme zur Übersetzung des vom Programmierer erstellten Codes in eine maschinenverständliche Sprache, für alle Arten von Prozessorplattformen zu entwickeln, so dass die Programmiersprache C/C++ heute für die gesamte Leistungspalette vom Mikrocontroller bis zu High-End-Rechnern verfügbar ist. Für den Entwickler von Software bedeutet dies: Egal für welche Prozessorplattform programmiert wird, einen C-Compiler wird man für das relevante Zielsystem bekommen. Man braucht sich nicht um eine Programmierung zu kümmern, die spezifisch für den jeweiligen Zielprozessor ist. In den meis-

ten Fällen wird es möglich sein, die auf einer Plattform entwickelte Anwendung auf einer anderen Plattform auszuführen. Der erforderliche Anpassungsaufwand ist in aller Regel sehr überschaubar.

Das bedeutet nicht, dass man fertige Programme von einer Plattform auf eine andere übertragen kann (etwa von einem Windows-PC auf einen Linux-PC) und diese dann auf der neuen Plattform (also unter Linux) sofort wieder funktionieren. Vielmehr ist nur die problemlose Übertragung der Quelltexte auf ein neues System gemeint, auf dem diese dann mit dem entsprechenden Compiler und Linker (ein Linker oder Binder ist ein Programm, das einzelne Programmmodule zu einem ausführbaren Programm verbindet) in ein funktionierendes Programm umzuwandeln sind!

Die Tatsache, dass Programme, die in C/C++ geschrieben werden, sehr klein sind (nur in Assembler – also Maschinensprache – geschriebene Programme sind noch kleiner), macht C/C++ zu einer wichtigen Programmiersprache im Bereich Embedded Systems (also Systemen, die stark einschränkenden Randbedingungen unterliegen, wie geringe Kosten, Platz-, Energie- und Speicherverbrauch) und der Mikrocontroller-Programmierung, wo Speicherplatz ebenfalls sehr kostbar ist.

Ein C/C++-Programm wird mithilfe eines **Compilers** (dem Übersetzer des Quelltextes) aus einer oder mehreren einfachen Textdateien zu Objektcodedateien übersetzt. Diese Objektcodedateien werden anschließend von einem **Linker** (bzw. Linkage-Editor = Binder, Werkzeug für das Zusammenfügen übersetzter Programmteile) mit den erforderlichen Systembibliotheken zu einer ausführbaren Datei (der Executable – oder kurz EXE-Datei) zusammengebunden.

Jedes ausführbare C/C++-Programm besitzt eine Hauptfunktion. In C wird diese Hauptfunktion als `main` bezeichnet.

Damit das Betriebssystem erkennen kann, wo der Einstiegspunkt für den Ablauf eines C/C++-Programms zu finden ist, muss diese Namenskonvention unbedingt eingehalten werden. Auch wenn andere Entwicklungsumgebungen als das Visual Studio von Microsoft oder andere Compiler eingesetzt werden, ändert sich an diesem Einstiegspunkt nichts. Variieren kann allenfalls die Parametrisierung (also die Art, Anzahl oder der Datentyp der Übergabeparameter) dieser Hauptfunktion. Dieser Aspekt wird später in Kapitel 4.8, in dem es um Funktionen gehen wird, noch ausführlich erläutert.

Darüber hinaus ist es natürlich auch möglich, eigene Programme und/oder Funktionen in eigenen Bibliotheken zusammenzufassen, um diese später erneut benutzen zu können. Diese Bibliotheken können bei einem späteren Bindevorgang durch den Linker wieder verwendet werden, damit diese dann zu einem neuen Programm hinzugebunden werden.

2.2 Compiler und Interpreter

Die höheren Programmiersprachen (dazu zählen u. a. FORTRAN, ALGOL und C/C++) sind entwickelt worden, damit die Beschreibung eines Lösungsverfahrens, der Algorithmus, in einer für Menschen einfacher lesbaren Form erfolgen kann und nicht in Maschinencode vorliegen muss. Der Programmierer als der Anwender solcher Programmiersprachen soll sich also auf die Lösung seiner konkreten Aufgabenstellung konzentrieren können, ohne sich zu sehr mit den Interna des Rechners beschäftigen zu müssen. Ein weiterer, nicht zu vernachlässigender Vorteil ist, dass Programmiersprachen normalerweise unabhängig von der Maschine sind, auf der die Programme ausgeführt werden.

Die Tatsache, dass es zwei verschiedene Verfahren zur Erzeugung von Programmen gibt, deutet das Problem an, das sich hinter der Verwendung einer Programmiersprache verbirgt: die Programme können nicht mehr direkt und unmittelbar vom Computer gelesen und zur Ausführung gebracht werden, sondern sie müssen erst in eine vom Computer interpretierbare passende Darstellungsform gebracht werden.

Eine Möglichkeit Quellprogramme (also die vom Programmierer erstellten und für den Programmierer lesbaren Textdateien mit den Programmen) zu übersetzen, sind **Interpreter**. Bei ihnen wird das Programm Zeile für Zeile gelesen und bewertet, wobei Schreibfehler oder Verstöße gegen die Regeln der Programmiersprache, Syntaxfehler, festgestellt werden. Danach führt der Interpreter die mit den Anweisungen verbundenen Befehle und Aktionen aus. Die Arbeitsweise solcher Interpreter kann wie in Bild 2.1 dargestellt werden.

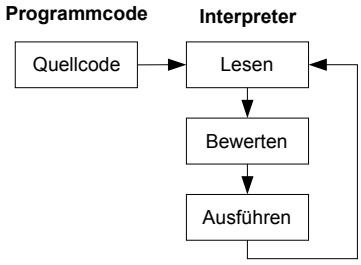

Bild 2.1
Arbeitsweise eines Interpreters

Der große Vorteil interpretierter Programme besteht darin, dass es möglich ist, schnell einmal etwas auszuprobieren oder während des Programmablaufs einzugreifen und Werte von Variablen zu betrachten oder zu verändern. Für professionelle Projekte dagegen sind Interpreter eher ungeeignet. Zuerst einmal ist da das Problem der mangelnden Geschwindigkeit zu nennen. Denn egal, ob ein Programm zum ersten oder tausendsten Mal ausgeführt wird, alle zugehörigen Programmzeilen durchlaufen immer wieder den Zyklus: *Lesen – Bewerten – Ausführen.*

Außerdem ist nur ein geringer Schutz des Quellcodes vor Eingriffen von außen gegeben. Der Anwender hat jederzeit die Möglichkeit den Code zu manipulieren, ohne sich mit dem Programmierer abzusprechen. Es entstehen so in der Praxis ziemlich schnell alternative Programmversionen, die bei einem neuen Release zu großen Problemen führen.

Der andere Lösungsansatz, um ein ausführbares Programm zu erzeugen, führt über den Einsatz eines Compilers. Ein **Compiler** ist ein Programm, das einen Quellcode einliest und

ihn zunächst auf syntaktische Fehler untersucht. Werden keine wirklichen Fehler festgestellt, so wird der Quellcode in eine maschinenlesbare Form übersetzt.

Nach dem abschließenden Schritt des Bindens (Linken) liegt das Programm in einer Form vor, die auf dem Rechner, auf dem es verarbeitet wurde, lauffähig (also ausführbar) ist (engl. executable, daher die Extension (Programmendung) EXE). Dieses Programm kann nun unabhängig von Quellcode und Compiler ausgeführt werden. Diese Arbeitsweise lässt sich wie in Bild 2.2 darstellen.

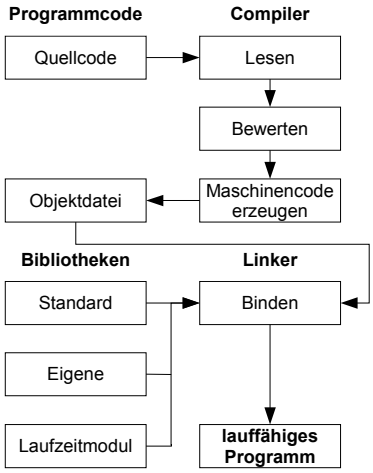

Bild 2.2
Arbeitsweise eines Compilers und Linkers

Der große Vorteil dieser Vorgehensweise liegt darin, dass die Programme ohne Preisgabe des Quellcodes lauffähig sind. So können z. B. auch im Quellcode verwendete Betriebsgeheimnisse von keinem Anwender mehr eingesehen werden. Vor allen Dingen aber laufen solche Programme sehr viel schneller ab als interpretierte Programme, da ja die Interpretation jedes Programmstatements (also jeder Programmzeile) entfällt. Die Überprüfung hat bereits bei der Kompilierung (also der Übersetzung in die Object-Datei) stattgefunden. Ein gewisser Nachteil besteht zweifellos darin, dass nun bei jeder Programmänderung der komplette Zyklus *Editieren – Übersetzen – Starten* durchlaufen werden muss, bevor man etwas in einem Programm ausprobieren kann. Das kostet in der Testphase natürlich etwas mehr Zeit als bei interpretativen Programmen. Denn auch beim Auftreten eines syntaktischen Fehlers muss man immer erst wieder das Programm in den Editor laden, den Fehler beheben und die Übersetzung (Kompilierung) von neuem starten.

Deutlich entschärft wird dieses Problem dadurch, dass die meisten Compiler heutzutage mit einer sogenannten integrierten Entwicklungsumgebung ausgestattet sind, in der der Editor und der Compiler eine Einheit bilden. Damit verkürzen sich die Bearbeitungsschritte erheblich.

2.3 Übersetzen eines C/C++-Programms

Nachdem der Quellcode eingegeben ist, muss dieser in eine für den Computer verständliche Sprache übersetzt werden. An dieser Stelle wird ein **Object** erzeugt. Dieser Prozess stellt einen Zwischenschritt bei der Herstellung eines ausführbaren Programms dar. Das Werkzeug, mit dem hier gearbeitet wird, ist ein Compiler. Das ist ein eigenständiges Programm zur Übersetzung des Quellcodes in ein Object.

1. Der erste Schritt ist der **Präprozessorlauf.** Bei dem Programm, das hier zum Einsatz kommt, handelt es sich um einen Textersetzer, der spezielle Dateien, die Headerdateien, in den Programmtext importiert. Zusätzlich vermag es der Präprozessor, Makros und symbolische Konstanten in die ihnen zugeordneten Anweisungsfolgen und Werte zu übersetzen und bestimmte Programmteile (Kommentare) vor der Übersetzung auszublenden. Vom Präprozessor wird eine temporäre Zwischendatei erzeugt, die aber nach dem Übersetzungsvorgang wieder zerstört wird und somit keinen dauerhaften zusätzlichen Speicherplatz auf der Festplatte beansprucht.
2. Im zweiten Schritt testet der Compiler, ob die Schreibweisen und der Aufbau des Programms den Regeln der Programmiersprache entsprechen. Hier spricht man von der **syntaktischen Analyse.** Ein dabei auftretender Fehler wird dementsprechend Syntaxfehler genannt. Moderne Compiler sammeln nun die Fehler und brechen die Übersetzung erst bei der Überschreitung einer gewissen Fehlerobergrenze ab. Anschließend kann der Anwender von der Fehlerliste aus zu den Programmstellen springen, die der Compiler moniert hat. Hier gilt es zu bedenken, dass es sich nicht immer nur um eine einzige Stelle handeln muss, an der der Fehler liegen kann. Evtl. ist schon einige Zeilen vorher ein Fehler programmiert worden, der erst jetzt zu einem Syntaxproblem führt und den Compiler zur Meldung veranlasst. Der Programmierer korrigiert die beanstandeten Zeilen und startet den Übersetzungslauf erneut. Je nach Komfort oder Aussagekraft der Fehlermeldungen ist die Analyse der Fehlerquellen aufwendig. In jedem Fall bedarf es einiger Übung, um aus den teilweise sehr kryptischen Fehlermeldungen die Fehlerquelle zu erkennen.
3. Nach einigen Durchgängen ist das Programm in der Regel frei von Syntaxfehlern. Der Compiler hat dann den **Objectcode** erzeugt. Dieser ist prinzipiell schon maschinenlesbar, kann allerdings noch nicht ausgeführt werden, da einerseits weitere Objectdateien evtl. hinzuzufügen sind, andererseits aber immer Bibliotheksdateien und das Laufzeitmodul hinzugebunden werden müssen.
4. Das **ausführbare Programm**, das dann auf der Plattform, auf der die Entwicklung durchgeführt wurde, lauffähig ist – und in aller Regel auch nur auf dieser Plattform, denn C/C++ ist eine plattformabhängige Programmiersprache –, erzeugt letztendlich der Linker. Hier werden nun alle Bestandteile, die zur Lauffähigkeit des Programms benötigt werden, also das Laufzeitmodul, die Bibliotheksfunktionen und Objectdateien, zu einer ausführbaren Programmdatei zusammengeführt (gebunden). Auch in diesem Schritt besteht noch die Möglichkeit, dass der Programmierer Fehler begeht. So sind Fehler, die in diesem Schritt entdeckt werden z. B. fehlende Funktionsdefinitionen, fehlende externe Variablen oder doppelte Funktionsdefinitionen. Auch in solchen Fällen muss der Programmierer wieder zurück bis in den Quellcode, um die Fehler zu korrigieren. Anschließend sind die aufgeführten Schritte erneut alle zu durchlaufen.

2.4 Programmstart

Nachdem der Quellcode fehlerfrei übersetzt und gebunden wurde, liegt eine Datei vor, die ein ausführbares Programm-Modul darstellt. Sie ist gekennzeichnet durch die **Extension** (Dateierweiterung) EXE. Dieses Programm kann nun, wie jedes andere Programm auch, gestartet werden. An dieser Stelle beginnt standardmäßig die Testphase, in der der Programmierer versucht herauszubekommen, ob die von ihm realisierte Lösung immer (d. h. mit unterschiedlichen Eingabedaten) zu einem vernünftigen und erwarteten Ergebnis führt. Dazu gehören selbstverständlich auch die Auswahl von geeigneten Testdaten sowie die Dokumentation von erwarteten und erzielten Resultaten.

Selten werden die Programme auf Anhieb die an sie gestellten Aufgaben mit den erwarteten Ergebnissen liefern. Es ist also nun wieder der Quellcode heranzuziehen, um ihn so zu berichtigen, dass die Ergebnisse den Erwartungen entsprechen.

Die häufigsten Fehler, die in diesem Zusammenhang gemacht werden, sind:

▪ Designfehler:	Die Lösung ist nicht angemessen. Zur Realisierung des Programms sind zu viele Annahmen getroffen worden, die nicht in jedem (Beispiel-) Fall tatsächlich auch vorliegen, oder es ist eine falsche Schrittfolge der einzelnen durchzuführenden Arbeitsschritte innerhalb des Quellcodes umgesetzt worden.
▪ Logische Fehler:	Z. B. falsche Abfragebedingungen oder Reihenfolge der Programmschritte falsch angeordnet.
▪ Sprachbedingte Fehler:	Da C/C++ eine äußerst knapp gehaltene Sprache ist, können Laufzeitfehler daraus resultieren, dass man sich schlicht vertippt hat. Es kann also passieren, dass man einen syntaktisch korrekten Programmteil erstellt, der aber doch ein logisch falsches Konstrukt enthält, das der Compiler allerdings nicht identifizieren kann. Man spricht hier von semantischen Fehlern.

Durch die sequentielle Arbeitsweise (Korrektur von Fehlern – Test der neuen Version – Korrektur von Fehlern – Test der neuen Version …) entstehen immer wieder neue Versionen des Programms, die man durch entsprechende Vergabe von Programmnamen oder Versionsnummern dokumentieren kann und sollte. Ist dann irgendwann der endgültige Programmstand erreicht, können diese Zwischenstände wieder entsorgt werden.

3 Die Entwicklungsumgebung Visual C++

Um die von Microsoft kostenlos zur Verfügung gestellte Version von Visual C++ nutzen zu können, muss das Programmpaket zunächst von der Homepage von Microsoft heruntergeladen werden. Dazu kann der Benutzer auf der Seite *https://www.visualstudio.com/de-de* zunächst die aktuelle Installationsdatei herunterladen, die für die eigentliche Installation anschließend auszuführen ist.

3.1 Installation von VC++

Nachdem die Installationsdatei gestartet wurde, wird die eigentliche Installation vorbereitet. Dazu wird vom Installationsprogramm ein Setup-Fenster angezeigt. Üblicherweise wird die Software direkt von der Microsoft-Homepage durch einen vollständigen Download installiert, d. h. es liegt kein Installationsdatenträger vor. Daher ist darauf zu achten, dass für den weiteren Fortgang der Installation zwingend eine Internetverbindung bestehen muss. Es wird dann zunächst eine Zusammenstellung der zu installierenden Komponenten angezeigt, bevor die eigentliche Installation durch Betätigung des Buttons *Installieren* gestartet wird.

Der Reihe nach werden nun die einzelnen Installationsschritte vom Installationsprogramm abgearbeitet. Visual C++ installiert dabei neben den Hauptkomponenten u. a. eine Hilfe-Referenz und möglicherweise den .NET-Framework in der erforderlichen Version, falls dieser nicht bereits auf dem Rechner installiert ist.

Die Software bietet Vorlagen für Apps unter Windows, Android, iOS und das Web mit voreingestellten Variablen und Argumentenlisten.

Das Installationsprogramm installiert Klassenbibliotheken, Befehlscode und Ressourcen-Dateien. Die zu installierenden Produkte können dabei durchaus variieren. Ihre exakte Zusammenstellung hängt davon ab, welche Komponenten möglicherweise bereits auf dem Zielsystem vorhanden sind. Die Installation dauert einige Minuten.

Nach erfolgreicher Installation wird das Setup durch einen entsprechenden Hinweis an den Benutzer abgeschlossen.

Im Anschluss kann VC++ gestartet werden und der Benutzer wird aufgefordert, sich bei Visual Studio anzumelden (siehe Bild 3.1

Bild 3.1 Anmeldung VC++ und Zustimmung Nutzungsbedingungen

Nach erfolgreicher Anmeldung wird die gewünschte Entwicklungsumgebung ausgewählt und der eigentlichen Programmierung steht nichts mehr im Wege (siehe Bild 3.2).

Bild 3.2 Auswahl der Entwicklungsumgebung

3.2 Starten von VC++

Die Entwicklungsumgebung kann aus der Programmübersicht durch den entsprechenden Aufruf gestartet werden:

Start → Alle Programme (bzw. Apps – je nach Betriebssystem) → Visual Studio (in der aktuellen Version)

Es erscheint das Startbild der Entwicklungsumgebung (siehe Bild 3.3).

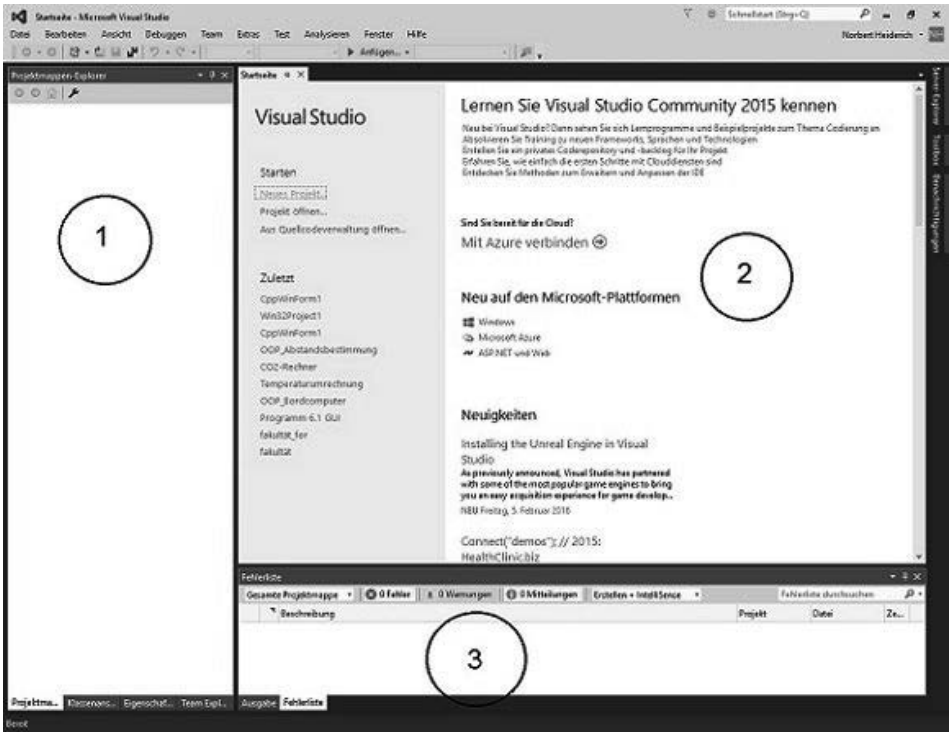

Bild 3.3 Der Startbildschirm von VC++

Der Bildschirm ist grob in drei Teile untergliedert:

1. **Projektmappen-Explorer, Klassenansicht und Eigenschaften-Manager**

   Über die drei Reiter am unteren Ende des ersten Fensterbereichs wird gesteuert, welche Informationen in dem Fenster angezeigt werden sollen.

   Im Projektmappen-Explorer werden die Projekte und die zugehörigen Dateien in einer Baumstruktur angezeigt. Zu dieser Ansicht gibt es eine spezielle Symbolleiste, auf der häufig verwendete Befehle für das in der aktuellen Liste markierte Element zu finden sind. Die Klassenansicht eines VC++-Projektes zeigt die Aufteilung des Projektes aus der Sicht der objektorientierten Programmierung (OOP).

   Wenn ein neues Projekt mit VC++ erstellt wurde, der Quellcode geschrieben ist, müssen ggfls. für das Projekt spezielle Eigenschaften definiert werden, damit z. B. der Compiler den Pfad zu den Include-Dateien und der Linker den Pfad zu den Bibliotheken finden kann. Diese Informationen können über den Eigenschaften-Manager editiert werden. Da es sich hier allerdings um sehr spezielle Informationen handelt, die in der Anfangsphase der Arbeit mit VC++ keine Rolle spielen, weil die Standardeinstellungen völlig ausreichend sind, kann diese Ansichtsauswahl getrost über den bekannten „Schließen“-Mechanismus entfernt werden. Dazu aktiviert man die entsprechende Ansicht und schließt dann mittel „X“ das aktuelle Fenster.

2. **News- und Projektlisten-Bereich**

   In diesem mit „Startseite“ überschriebenen Fenster werden News von Microsoft angezeigt. Die Ansicht wird bei jedem Start des Programms über das Internet aktualisiert. Im linken Teil dieses Fensters steht eine Liste der Projekte zur Verfügung, die zuletzt bearbeitet wurden. Soll an einem der aufgeführten Projekte weitergearbeitet werden, so kann dieses durch Doppelklick ausgewählt und in die Entwicklungsumgebung geladen werden.

3. **Codedefinitionsfenster, Aufrufbrowser und Ausgabe**

   Das Codedefinitionsfenster ist eine spezielle Editoransicht des Quelltextes, in der die Definition eines ausgewählten Symbols aus einer Codedatei angezeigt wird. Diese Ansicht ist zunächst nicht erforderlich und kann geschlossen werden.

   Der Aufrufbrowser ist dazu gedacht, in einem Projekt Funktionsaufrufe zu suchen. Anschließend kann über die Suchergebnisse einfach zu einem Funktionsverweis in den Quellcodedateien des Projektes navigiert werden. Da die ersten Projekte sicherlich nicht über die Komplexität verfügen, dass der Aufrufbrowser sinnvoll und gewinnbringend eingesetzt werden kann, kann auch diese Ansicht geschlossen werden.

   In der Ausgabe werden die aktuellen Protokolle (Compiler und Linker) der gerade ausgeführten Aktion aufgelistet.

Nachdem im unteren Bereich die Ansichten überarbeitet wurden, kann eine für die Entwicklungsphase sehr sinnvolle Ansicht in der Auswahl aufgenommen werden.

Über die Auswahl: *Ansicht → weitere Fenster → Fehlerliste* bekommt man eine tabellarische Auflistung aller möglicher Fehler, Warnungen und Meldungen all der Aktionen, die in der Entwicklungsumgebung ausgeführt werden. Dieses Fenster mit der übersichtlichen Auflistung aller Problemfälle des Quellcodes aus Sicht des Compilers und Linkers ist besonders in der Anfangsphase sehr hilfreich! So erspart die Auflistung die Arbeit mit den eher gewöhnungsbedürftigen Protokollen des Compilers und Linkers, da die relevanten Informationen bereits herausgefiltert werden.

3.3 Erstellen eines neuen Projektes

In der ersten Phase der Programmerstellung werden wir uns zunächst mit Konsolenanwendungen beschäftigen. Eine **Konsolenanwendung** ist ganz allgemein ein Computerprogramm ohne grafische Benutzeroberfläche (also ohne *Graphical User Interface*, oder auch kurz GUI), das ausschließlich über textbasierte Eingaben gesteuert wird. Das Gegenstück, also die GUI-Anwendungen, die über grafische Oberflächen verfügen, werden in Kapitel 8 behandelt.