HANSER

Leseprobe

zu

Technische Probleme lösen mit C/C++

von Norbert Heiderich und Wolfgang Meyer

Print-ISBN: 978-3-446-46823-8
E-Book-ISBN: 978-3-446-46896-2

Weitere Informationen und Bestellungen unter
https://www.hanser-kundencenter.de/fachbuch/artikel/9783446468238
sowie im Buchhandel

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

Vorwort der Autoren

Die vierte Auflage war schon nach kurzer Zeit vergriffen. Das ist ein deutliches Indiz für die zunehmende Digitalisierung unseres Alltags. Neben der alltäglichen digitalen Kommunikation und der beruflichen wie privaten Nutzung des Internets rückt die Lösung von Problemstellungen mit Hilfe der Programmierung immer mehr in den schulischen, studentischen und beruflichen Fokus. C/C++ ist eine in der Wirtschaft sehr weit verbreitete, problemorientierte Programmiersprache. Das Erlernen anhand konkreter Problemstellungen ist eine praxisorientierte Investition in die eigene berufliche Zukunft.

Der pädagogische Ansatz *so wenig Theorie wie nötig, so viel Praxis wie möglich* kommt bei den Leserinnen und Lesern offensichtlich gut an. Aber nichtsdestotrotz ist ein theoretischer Hintergrund für das Schreiben *guter* Programme weiterhin unverzichtbar.

Wie in den bisherigen Auflagen legen wir als Autoren besonderen Wert auf die Problemanalyse, also auf die theoretische Durchdringung der Aufgabenstellung, ohne die beispielsweise ein späterer Programmtest nicht möglich wäre. Syntaktische Fehler zeigt der Compiler an, für die logischen Fehler ist allein der Programmierer verantwortlich. Die Vorstufe zur Umsetzung der Problemanalyse in ein C/C++-Programm ist das Struktogramm nach Nassi-Shneiderman, das sich in der strukturierten Programmierung gegenüber dem Programmablaufplan durchgesetzt hat und auch in diesem Buch durchgängig Verwendung findet.

Zusätzlich zu den umfangreichen Aufgaben, Beispielen und den nach Schwierigkeitsgrad gestaffelten Problemstellungen haben wir in die fünfte Auflage die Monte-Carlo-Methode, die Volumenberechnung von Rotationskörpern und einen Primzahlalgorithmus aufgenommen, wie er in vielen Sicherungsverfahren digitaler Codierung verwendet wird. Und zur Steigerung der Motivation haben wir schon weit vorne „Ein erstes Programm in C“ eingebaut, denn wir wissen, dass nichts motivierender ist als der Erfolg.

Wir wünschen Ihnen also viel Freude und gute Ergebnisse beim Programmieren mit C/C++!

Norbert Heiderich

Wolfgang Meyer

Inhalt

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.

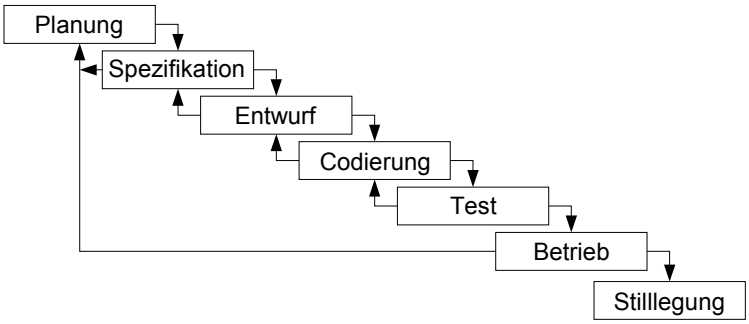

Bild 1.1 Software-Lebenszyklus

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.

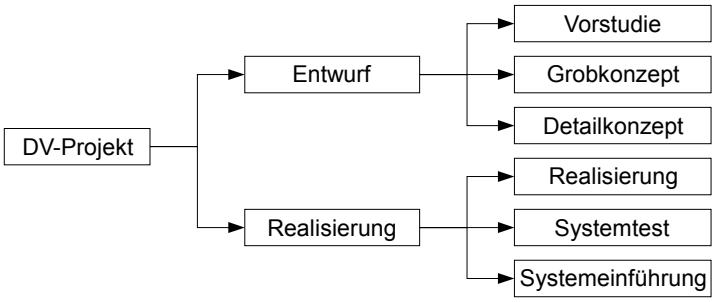

Bild 1.2 Das 6-Phasenmodell

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.

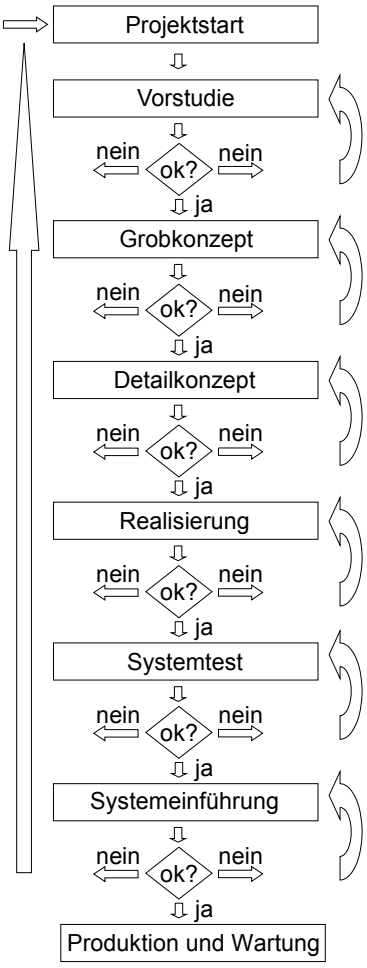

Bild 1.3
Ablauf des 6-Phasenmodells

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 mit Hilfe 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. Varieren kann allenfalls die Parametrisierung (also die Art, Anzahl oder der Datentyp der Übergabeparameter) dieser Hauptfunktion. Dieser Aspekt wird später in Abschnitt 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.

Index

D

E

F

T

U

V

W

Z