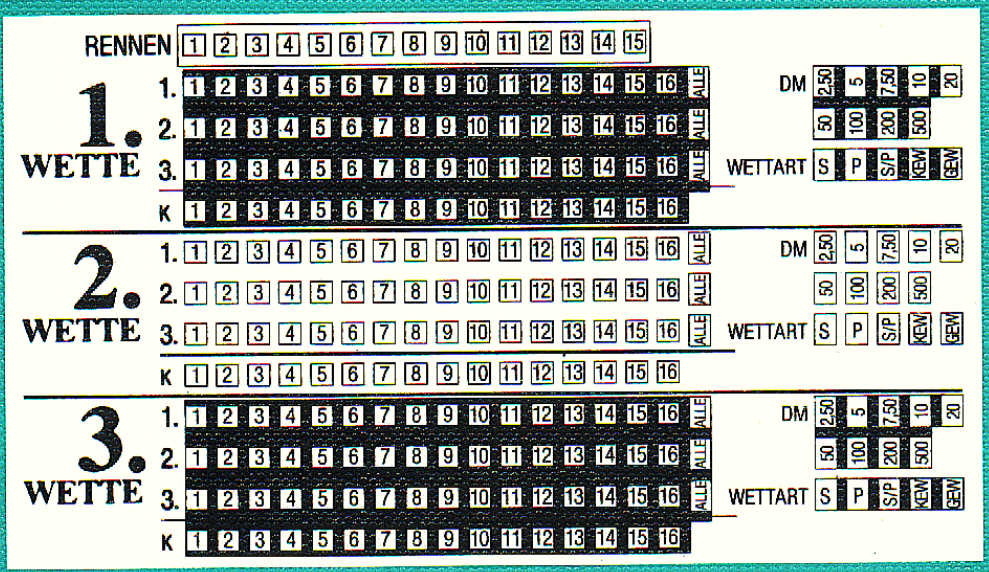
BAUSTEINE INFORMATIK

Eberhard Lehmann

Projekte im Informatik-Unterricht

- Software-Engineering -



- Analyse
- Konstruktion
- Wartung

komplexer Software

DÜMMLER

1995

Inhaltsverzeichnis

Vorwort

Die Arbeit an komplexen Systemen mit den Methoden der Projektarbeit gehört zu den Inhalten der Schulinformatik, die besonders praxisnah sind. Überall werden umfangreiche Softwaresysteme benutzt und mindestens soweit analysiert, daß man ihre Funktionen erfaßt und sinnvoll für die eigenen Probleme einsetzen kann. Doch wie entstehen derartige Systeme, welche Konstruktionsverfahren setzt man ein, welche Kenntnisse muß man haben, um komplexe Probleme soweit zu modellieren, daß sie einen brauchbaren Ausschnitt aus der vorliegenden Realität darstellen? Gerade hiermit beschäftigt sich das vorliegende Buch.
Das Buch ist als ein Schülerbuch konzipiert, das in die Methoden zur Bearbeitung von Softwareprojekten (Software-Engineering) einführt und zahlreiche Übungsaufgaben zur Verfügung stellt. Es ist unabhängig von den anderen schon erschienenen Bänden zur Reihe "Analyse, Konstruktion, Wartung komplexer Software", setzt diese Bände also nicht voraus!

Kapitel 1 beschäftigt sich mit dem Umfeld der Konstruktion komplexer Softwareprodukte, insbesondere mit der **Projektmethode** und dem Begriff des Software-Engineering. Dieses Hintergrundwissen ist auch für Schüler wichtig, soll diesen bewußt gemacht werden und trägt wesentlich zum Verständnis der Zusammenhänge bei.

Kapitel 2.1 zeigt **Wege zur erfolgreichen Konstruktion von Softwaresystemen** auf und betont dabei die Bedeutung des Projektmanagements und des Prototyping. Kapitel 2.2 beschreibt die **Konzepte**, die man beherrschen muß, um komplexe Datenverarbeitungsprobleme zu bewältigen. Programmiersprachen spielen hier nur eine untergeordnete Rolle, die Konzepte sind programmiersprachenunabhängig. Wegen der besonderen Bedeutung der **Projektdokumentation** für spätere Wartungsarbeiten an der Software erhält diese ein eigenes Kapitel (2.3). Dabei werden die Grundlagen einer Programmiersprache vorausgesetzt, hier PASCAL. Schul-Softwareprojekte scheitern nicht selten an diversen Unzulänglichkeiten. Die **Warnungen** in Kapitel 2.4 entspringen meiner längjährigen Erfahrung in Projektarbeit. Die angegebenen **Checklisten** können helfen, Projektarbeit erfolgreich abzuschließen!

Kapitel 3 bringt die **Beschreibung eines realen Projektablaufs in Form einer Entwicklungsdokumentation.** Das gewählte Beispiel "Simulation einer Trabrennbahn" wurde mit einer Schülergruppe in Klasse 12/13 durchgeführt. Es kann als Muster für andere Projekte dienen.

Grundzüge der Projektarbeit kann man auch bei der **Wartung von Software** (Reenginee-ring) lernen. Grundlegende Ausführungen hierzu bringt Kapitel 4, als

Beispiel wird erneut das Trabrennbahn-Programm herangezogen, von dem hier eine zweite Version entsteht. In dem Kapitel werden auch Hinweise auf weitere Softwaresysteme des Autors gegeben, die für Wartungsprojekte geeignet sind.
Es gibt in der Literatur kaum **Aufgabenstellungen zu Informatik-Projektarbeit**! Diese Lücke soll mit den **Übungsaufgaben** in Kapitel 5 geschlossen werden. Diese beschäftigen sich mit Fragen aus verschiedenen Bereichen des Software-Engineering. Es sind insbesondere Fragestellungen zu den Konzepten und dem Bewußtmachen von Zusammenhängen.

Kapitel 6 ist ein **Glossar**, in dem weitere Begriffe im Zusammenhang mit Software-Engineering erläutert werden, soweit sie nicht schon im Buchtext näher betrachtet wurden.

Die meisten Darstellungen im Buch sind programmiersprachenunabhängig. Die Program-miersprache PASCAL (Turbo-PASCAL 6) wird wichtig beim Angebot an Hilfspro-grammen und wiederverwendbaren Bausteinen (Kapitel 2.2.6, 2.2.7) und beim Trabrennbahnprojekt in Kapitel 3. Die MS-DOS-**Begleitdiskette** SOFTWARE-ENGINEER (Turbo-PASCAL 6) enthält nicht nur Quellprogramm und compiliertes Programm zum Trabrennbahnprojekt (das hiermit auch für Wartungsarbeiten zur Verfügung steht), sondern auch eine zweite Version des Trabrennbahrn-Programms nach einem Wartungsprojekt, siehe Kapitel 4.

Kapitel 7 verweist auf weitere Bücher und Disketten in Zusammenhang mit Projektarbeit.

Zum Adressatenkreis des Buches gehören Schüler, Lehrer, Studenten, Dozenten und alle, die sich mit der Konstruktion von Software beschäftigen.

An Schüler und Studenten: *Projektarbeit ist die Arbeitsform im Informatikunterricht, die wegen ihrer Praxisnähe besonders wichtig ist. In der beruflichen Praxis werden nicht nur im Bereich der Datenverarbeitung komplexe Probleme mit dieser Arbeitsform gelöst, so daß viele der hier vermittelten Methoden über die Informatik hinausgehen (z.B. Teamarbeit). Nach allen Erfahrungen macht diese Arbeitsform zudem viel Spaß. Ich hoffe, daß auch Sie diese Freude empfinden, wenn Sie Ihr Projektthema schrittweise einer Lösung zuführen und schließlich das fertige Produkt vor sich haben!*

An Lehrer und Dozenten: *Projektartiger Unterricht erfordert einige Übung. Sie finden in dem Buch viele Hinweise zur Projektmethode und zur Unterrichtspraxis.*

Für private Softwarekonstrukteure *bietet das Buch Grundlagen und Methoden, die zur erfolgreichen Konstruktion größerer Softwareprodukte führen.*

Berlin, Januar 1995 Eberhard Lehmann

1. Konstruktion komplexer Software

1.1 Komplexe Softwareprodukte in der Schule

Das vorliegende Buch befaßt sich mit der Herstellung komplexer Softwareprodukte im Informatikunterricht der Schule.

Was heißt Fehler! Verweisquelle konnte nicht gefunden werden.**"komplexe Software"?**

Abhängig vom Kenntnisstand des Benutzers wird die Beurteilung der Komplexität einer Software variieren. Einige Kriterien zur Beurteilung des Komplexitätsgrads von Softwareprodukten:

Komplexe Softwareprodukte zeichnen sich aus durch:

- ihren Umfang, meßbar u.a. an der Anzahl der Programmzeilen oder an dem benötigten Speicherplatz
- lange Entwicklungsdauer, an der in der Regel mehrere Personen beteiligt waren
- eine Vielzahl abrufbarer Funktionen, sichtbar u.a. an der Menge der Schnittstellen zwischen dem Benutzer und dem Computer
- vielfältige Verknüpfungen zwischen den Programmteilen
- umfangreichen Datenaustausch
- eine Vielzahl von Algorithmen

Beispiel 1

Im Projektunterricht werden häufiger Programmsysteme zur **Schülerbibliotheksverwaltung** einer Schule erstellt. Der Bestand an Schulbüchern muß aufgenommen und gespeichert werden, Neuzugänge und Abgänge von Büchern führen zu einer Aktualisierung der Bücherdatei. Der Leihverkehr arbeitet mit dieser Bücherdatei, aber auch mit der Schülerdatei und muß auf diverse Entleihsituationen reagieren, z.B. auch Mahnungen an Schüler aussprechen. - Hierbei handelt es sich offenbar um ein sehr komplexes Softwareprodukt.

Beispiel 2

Im Informatikunterricht wird ein **elementares Sortierverfahren** programmiert, z.B. "Sortieren in einem Feld mit Hilfe von Bubblesort". **Fehler! Verweisquelle konnte**

nicht gefunden werden. Diese Programmieraufgabe ist wenig komplex. Sie kann leicht von einem einzelnen Schüler bewältigt werden.

Schlüsselqualifikationen

Warum sollen Sie in der Schule an einem Softwareprojekt mitarbeiten?

Die Erfahrung zeigt, daß die Herstellung komplexer Software ein schwieriger Prozeß ist. Das gilt für die Herstellung bei Softwarefirmen, für Softwareprojekte an der Universität und erst recht für die in der allgemeinbildenden Schule. Dennoch gibt es gute Gründe, auch in der Schule komplexe Software zu entwerfen, insbesondere dann, wenn man nicht nur die Erstellung der Software zum Ziel hat, sondern den gesamten Herstellungsvorgang in die Betrachtung miteinbezieht. Hierbei werden nämlich Prozesse in Gang gesetzt, die später in der Berufspraxis immer wieder benötigt werden.

Damit erweisen sich die bei Projektarbeit im Projektteam erwerbbaren Kenntnisse und Fähigkeiten als wesentliche Schlüsselqualifikationen für die spätere Berufstätigkeit.

Schlüsselqualifikationen sind solche Kenntnisse, Fähigkeiten und Fertigkeiten, die geeignet sind für

- eine große Anzahl von Positionen, Funktionen und Tätigkeiten
- die Bewältigung geänderter, auch unvorhersehbarer Anforderungen im Verlaufe des Lebens.

Qualifikationen, die nur einen unmittelbaren und begrenzten Bezug zu bestimmten, nicht gleichen Tätigkeiten erbringen, sind also keine Schlüsselqualifikationen.

Heute werden Computerqualifikationen auf dem Arbeitsmarkt als eine zentrale Schlüsselqualifikation angesehen. Dabei ist nicht immer klar, welche Bereiche der Datenverarbeitung es eigentlich sind, die diese Qualifikation ausmachen; sicher nicht lediglich die Handhabung von verbreiteten Anwendersystemen, wie z.B. zur Textverarbeitung oder Tabellenkalkulation.

Vielmehr sind solche Informatikqualifikationen gemeint, die über diese Handhabungsfertigkeiten hinausgehen. Es sind übergeordnete strukturelle Gemeinsamkeiten; es sind Kenntnisse über die Nutzungsmöglichkeiten von Computersystemen überhaupt, es sind Fähigkeiten, die zunehmende Komplexität der Informationsverarbeitung zu bewältigen. Und dabei erweisen sich die grundlegenden Methoden und Arbeitsweisen als besonders wichtig. **Komplexe Systeme können nicht durch einzelne Personen beherrscht werden. Die Arbeit im Team ist unerläßlich und demzufolge weit verbreitet. Für diese in der Berufspraxis so wichtige Arbeitsweise ist der Informatikunterricht in besonderem Maße geeignet, denn:**

Der Informatikunterricht bietet schon vom fachlichen Ansatz her die Möglichkeit von Projektarbeit, z.B. zur Konstruktion komplexer Software und hierbei muß ebenfalls im Team gearbeitet werden.

In zahlreichen Projekten im Wirtschaftsleben werden Kenntnisse aus verschiedenen Gebieten benötigt. Das fächerorientierte Lernen in der Schule fördert jedoch gerade diesen Aspekt kaum. Für die Informatik jedoch gilt:

Softwareprojekte haben in der Regel einen fächerübergreifenden Charakter, und die Informatik ist wegen der Vielzahl ihrer Anwendungsbereiche besonders geeignet, um gerade diesen Aspekt erfolgreich zu berücksichtigen.

In der Schule herrscht in der Regel das sogenannte "Schubkastendenken". Jedes Fach arbeitet nur für sich selbst, Bezüge zu anderen Fächern werden kaum hergestellt, sondern eher vermieden. Das liegt meistens an der Ausbildung der Lehrer, aber auch an den Lehrplänen. So kommt es, daß sich auch die Schüler dieser Situation anpassen und immer nur das Schubfach öffnen, das die Inhalte des gerade aktuellen Faches enthält. Im Informatikunterricht kann das enge Fächerdenken überwunden werden.

Es ist weit interessanter, eine komplexe Anwendung zu programmieren, als ständig nur kleine Programme zu entwerfen, die keine praktische Bedeutung haben.

An der Herstellung solcher kleinen Programme ist der Schüler von seiner häuslichen Arbeit am Computer gewöhnt; doch wie verläuft der Herstellungsprozeß bei umfangreichen Problemstellungen?

Jedes Team sieht sich als Empfänger und Zulieferer zugleich und achtet deshalb nicht nur auf die Qualität der empfangenen Leistung, sondern auch auf die Qualität der weitergegebenen Leistung.

Teamarbeit bedeutet, Verantwortung zu übernehmen!

1.2 Was versteht man unter Software-Engineering?

"Anwendung von Prinzipien, Methoden und Techniken auf den Entwurf und die Implementierung von Programmen und Programmsystemen. Der Begriff "Software-Engineering" steht für die Auffassung, daß die Erstellung, Anpassung und Wartung von Programmsystemen kein "künstlerischer", sondern vorwiegend ein ingenieurmäßig ablaufender Prozeß ist. Der Begriff entstand in den 60er Jahren, als die Entwicklung immer größerer Softwaresysteme zunehmend problematischer wurde (sog. Softwarekrise). Er sollte einen Umdenkprozeß in der Softwareproduktion einleiten, Bewertungskriterien stärker berücksichtigen und zu qualitativ hochwertigen Produkten führen." (Duden Informatik, Dudenverlag 1993.)

Im Duden folgen dann längere Detailausführungen, die die Bedeutung von Software-Engineering unterstreichen.

Software-Engineering hat in der Datenverarbeitungspraxis eine immer größere Bedeutung gewonnen, schon allein deswegen, weil in zahlreichen Bereichen etwa der Wirtschaft und Verwaltung immer mehr Datenverarbeitungsanlagen installiert werden, die immer komplexere Aufgaben zu bearbeiten haben. Dazu mußte passende hochwertige Software entwickelt werden, die nach ihrer Implementierung auch eine ständige Wartung angesichts neuer Anforderungen ermöglichte.

Wir müssen also beim Software-Engineering unterscheiden zwischen
- Neuentwicklung von Software
- Wartung von Software (Reengineering), d.h. Anpassung an veränderte Gegebenheiten.

Der Begriff "Engineering" weist darauf hin, daß versucht wird, die komplexen Anforderungen an die Programmsysteme mit ingenieurmäßigen Methoden zu bewältigen. Ein Blick in Fachzeitschriften zur Datenverarbeitung zeigt, daß man sich immer wieder neue Gedanken macht, wie man qualitativ hochwertige große Softwaresysteme rationell und damit auch mit möglichst wenig Kosten herstellen kann. Dabei wird die Herstellung der Software teilweise in den eigenen Datenverarbeitungsabteilungen der Firmen vorgenommen, die die Software benötigen, oder die Aufträge werden an spezialisierte Firmen ("Softwarehäuser") vergeben.

Was ist mit "ingenieurmäßigen Methoden" gemeint? Wir versuchen uns die Vorgehensweise von Ingenieuren zu verdeutlichen, indem wir mit Vertretern anderer Berufszweige vergleichen.

Der ...	*Ausgewählte Tätigkeiten*	*Der Ingenieur*	*Beide*
Jurist in der Rechtsprechung	interpretiert Gesetze und wendet sie an	erzeugt eigenständige Entwicklungen	gehen systematisch vor
Betriebswirt-	sucht das wirtschaftlich sinnvolle: definiert die Durchführbarkeit als Wirtschaftlichkeitsproblem (Nutzen > Kosten?)	sucht das technisch Anspruchsvolle; definiert die Durchführbarkeit als technisches Problem (Werkzeuge, Technologie vorhanden?)	orientieren sich bei ihrer Arbeit an präzisen (aber unterschiedlichen) Zielvorstellungen
Handwerker bzw. der **Arbeiter** in der Fertigung	stellt her; erstellt das Produkt	entwickelt; erstellt den Plan für das Produkt bzw. für dessen Herstellung	haben die Entstehung eines (fertigen) Produkts im Auge
Physiker/Chemiker	entwickelt theoretische Erkenntnisse; entwickelt eine Versuchsumgebung bzw. einen Prototyp	wendet theoretische Erkenntnisse an; entwickelt ein Produkt	wenden mathematische Methoden an; sind im naturwissenschaftlichen Bereich tätig
Werbegrafiker	geht unkonventionell vor	geht systematisch, standardisiert vor	sind schöpferisch, kreativ und formgebend tätig, indem sie etwas erfinden und Ideen realisieren

Abb. 1.2.a: Arbeitsweise in verschiedenen Berufszweigen
(nach H.J.Ott: Das "ingenieurgemäße" am Software Engineering, in der Zeitschrift "Softwaretechnik-Trends", Februar 1994)

Die Tätigkeiten des Software-Ingenieurs orientieren sich an der letzten Spalte. Er muß etliche der genannten Eigenschaften in sich vereinen.

Ein guter Software-Ingenieur

- entwickelt neue oder verbessert alte Produkte, es sind Anwendungen mit Praxisbezug
- strukturiert seine Arbeitsaufgabe in Einzelaufgaben, seine Arbeitsabläufe in Phasen, er vergibt Prioritäten der Bearbeitung
- geht systematisch/methodisch vor, er entwickelt das Produkt mit Hilfe von Werkzeugen "auf Papier" oder am Computer (Tools)
- prüft die Ergebnisse aller Arbeitsphasen auf Qualität

- denkt strukturell, d.h. er zerlegt die zu entwickelnde Software in Bausteine bzw. Objekte, die Bausteine werden getrennt bearbeitet und danach zum Gesamtsystem integriert.
- verwendet bestehende Bausteine wieder
- hält sich soweit wie möglich an Standards (Normen)
- plant (denkt und arbeitet übergreifend), erstellt Programmieraufträge
- arbeitet zielorientiert und vorausschauend
- wagt auch Neues, Unkonventionelles

SCHWIERIGKEITEN BEI DER KONSTRUKTION UMFANGREICHER SOFTWARE

Mit dem oben im Zitat genannten Stichwort "**Softwarekrise**" sind bereits Schwierigkeiten in Zusammenhang mit Softwareentwicklung angedeutet. Die Softwarekrise entwickelte sich, als angesichts der immer größeren Verbreitung von Datenverarbeitungsanlagen und komplexerer Aufgabenstellungen auch immer mehr geeignete Software benötigt wurde. Die häufig von Einzelpersonen allein erstellten Systeme hatten oft nur eine geringe Qualität und konnten nicht oder nur mühsam von anderen Personen erweitert werden. Jeder Entwickler verfolgte seinen eigenen Stil, allgemein gültige Methoden gab es nur in Ansätzen. Diese Art der Softwareherstellung konnte so angesichts der nun zu erstellenden Mengen, der gestiegenen Qualitätserwartungen und gestiegener Preise für Dienstleistungen nicht mehr fortgeführt werden. Man mußte nach Methoden suchen, die den Herstellungsvorgang ökonomischer machen und Produkte mit höherer Qualität liefern.

Angesichts der oben angedeuteten Bedeutung qualitativ hochwertiger Software in allen Bereichen der Praxis hat auch die Schule den Auftrag, sich mit Softwareanwendungen zu beschäftigen und den Entstehungsprozeß von Software im Informatikunterricht kritisch zu untersuchen.

1.3 Projektarbeit im Informatik-Unterricht

Schüler und Lehrer müssen wissen, daß Projektunterricht ist eine besondere Arbeitsform ist, die ihre eigenen Ziele hat, besonders interessant ist, aber auch einige Schwierigkeiten mit sich bringt. Die folgenden Ausführungen dazu beziehen sich auf verschiedene Projektarten (Projekttag, Projektwoche, Projekt über mehrere Tage (Wochen, Monate) in Unterrichtsfächern usw. - sie meinen aber besonders Projekte im Informatikunterricht und dabei vor allem Softwareprojekte.

Projektorientierter Unterricht

- will die festgelegten schulischen Fächergrenzen überwinden,
- behandelt komplexe, realitätsnahe Probleme und ist meistens fächerübergreifend,
- ist der Lern- und Arbeitsprozeß in einer sozialen Gruppe, deren Ziel oft selbst gesetzt ist,
- zeichnet sich durch die Arbeit im Team aus,
- will das Auseinanderfallen von Theorie und Praxis überwinden,
- hat ein Ziel, das sich aus der Integration gemeinsam gewonnener Erfahrungen realisiert,
- hat auch die benutzten Arbeitsmethoden zum Inhalt,
- erzeugt Produkte, über die mit anderen Personen kommuniziert werden kann.

Allgemeine und soziale Ziele

- Teamfähigkeit entwickeln,
- Notwendigkeit und Sinn von Arbeitsteilung einsehen,
- Artikulationsfähigkeit entwickeln.

Ziele zur Erlangung von Planungskompetenz

- Arbeitsmittel und zur Verfügung stehende Ressourcen richtig einschätzen,
- Kritikfähigkeit eigener und fremder Arbeit gegenüber entwickeln,
- Gewinnung und Auswertung von Informationen üben,
- einzeln und im Team Entscheidungen treffen.

Auf die Computeranwendung bezogene Ziele

- Komplexität realer Problemstellungen erkennen,
- unterschiedliche Interessenlagen von Auftraggeber, Hersteller, Anwender und von der Anwendung Betroffenen erkennen,
- die Auswirkungen unterschiedlicher Designentscheidungen erkennen.

Auf die Softwareerstellung bezogene Ziele

- Interdisziplinarität des Herstellungsvorgangs erkennen und bewältigen,
- die Methodik der Softwareerstellung als Mittel zur Bewältigung inhaltlich und organisatorisch komplexer Probleme begreifen und erfahren,

- Methoden der Softwarekonstruktion anwenden können.

Projektorientierter Unterricht bringt Schwierigkeiten mit sich!

1. Schüler und Lehrer sind i.a. nicht gewohnt, **mehrere Wochen an einem Problemkreis** miteinander zu arbeiten und viele Einzelergebnisse in ein großes übergeordnetes System zu integrieren,

2. Die **Arbeit im Team** über eine so lange Zeit hinweg erfordert viel gegenseitige Rück-sichtnahme und Einordnung.

3. Die **Vielseitigkeit der Aufgabenstellungen** innerhalb eines Projekts bedeutet, daß man aus dem bisherigen Informatik-Unterricht einen gewissen Überblick über Lösungsmethoden gewonnen haben muß und in der Lage ist, diese zunächst mit Anleitung, dann aber doch weitgehend selbständig auswählen und anwenden zu können.

4. In der Wahl des Themas liegt ein gewisses Risiko, da anfangs oft nicht alle möglichen auftretenden Probleme erkennbar sind. So kann selbst das **Scheitern eines Projekts** nicht ausgeschlossen werden.

5. Für den Lehrer bedeutet Projektunterricht eine **andere Art der Unterrichtsvorbereitung.** Sie kann ja nicht darin bestehen, das Projekt vorab für sich selbst durchzuführen. Vielmehr verschiebt sich das Schwergewicht der Vorbereitung auf organisatorische Fragen. Im Unterricht selbst wird vom Lehrer wegen der parallel und arbeitsteilig tätigen Schülergruppen ein besonderes Maß an Übersicht und Flexibilität gefordert.

In Kapitel 1.2 wurde auf die Bedeutung des Software-Engineering in der Datenverarbeitungspraxis hingewiesen. Im Informatik-Unterricht kann Software-Engineering, wie es in der Praxis vorkommt, nachempfunden werden.

Wir versuchen also, uns an der betrieblichen Praxis der Softwareentwicklung zu orientieren, übertragen aber die Ansätze auf die Schulpraxis, sofern sie dafür geeignet sind. Die passende Arbeitsform dafür ist in der Schule die Projektarbeit, die Schülern und Lehrern genügend Möglichkeiten gibt, die Vorgehensweise in der Praxis nachzuahmen.

Oben wurde bereits geschildert, was man sich in der Schule von Projektarbeit im Informatik-Unterricht verspricht. In der Figur 1.3.a sind wichtige Aspekte genannt, die in den Projektunterricht eingebracht werden sollten - selbstverständlich themenabhängig in unterschiedlicher Intensität.

Fachliche Aspekte

Kennenlernen grundlegender Methoden
- Software-Life-Cycle
- Prototyping
einsehen
- Projektmanagement
- Benutzeroberflächen
Informa-
- Modularisierung
- Unterstützung durch Tools

Pädagogische Aspekte

Berücksichtigung allgemeiner Ziele von Projektarbeit
- Notwendigkeit von Arbeitsteilung

- Teamarbeit
- Gewinnung und Auswertung von

tionen lernen
- Kritikfähigkeit eigener und fremder Arbeit gegenüber entwickeln
- ...

Aspekte aus dem gesellschaftlichen Bereich

- Beachtung von Datenschutzaspekten
- Auswirkungen des DVA-Einsatzes erkennen

Auseinandersetzung mit dem Anwendungsbereich

Kennenlernen der besonderen Kennzeichen des Bereichs
- soziale Probleme
- ...

Aspekte aus dem technischen Bereich

- Hardware-Entscheidungen treffen
- ...

Algorithmischer Bereich

- Design-Entscheidungen treffen
- Entwerfen, Modularisieren
- Programmieren
- Tools anwenden können
- ...

Das Ziel ist die Erstellung eines Softwareprodukts einschließlich einer benutzer- und wartungsbezogenen Dokumentation.

Aspekte der Wartung von Software
- Benutzung und Analyse von Software
- Einsatz von Methoden zur Durchführung von Änderungen an der Software
- Änderungsdokumentation

Figur 1.3.a : Aspekte bei Projektarbeit im Informatik-Unterricht

Software-Engineering in Projektarbeit ist also ein komplexes, dafür aber sehr interessantes Unternehmen, in das Sie viele Ihrer erworbenen Kenntnisse einbringen und zahlreiche neue Kenntnisse erwerben können. Software-Engineering bzw. Projektarbeit muß selbstverständlich im vorhergehenden Unterricht angemessen vorbereitet werden.

In der Regel haben Sie im vorhergehenden Unterricht auch den Bearbeitungszyklus "Analyse - Entwurf- Programmierung - Testen - Verbesserungen" schon bei anderen weniger umfangreichen Problemen durchlaufen, so daß der Einstieg in komplexere Fragestellungen nicht allzu schwer zu sein scheint, zumindest nicht bei den algorithmischen Fragestellungen.

Für welche Zwecke kann man in der Schule umfangreichere Software herstellen?

(1) Es kann sich bei dem Produkt um Software handeln, deren Einsatz in anderen Schulfächern vorgesehen ist. Dann sind die Lehrer dieses Faches und ihre Schüler die Benutzer.

(2) Die Software kann etwa auch für die ITG (Informationstechnische Grundbildung) als Grundlage einer projektartigen Unterrichtsreihe vorgesehen sein. Dann sind jüngere Schüler, die Grundlagen der DVA und ihre Anwendung an einem ausgewählten Thema kennenlernen, die Nutzer.

(3) Die Software kann aber auch in der Informatik selbst als Studienobjekt eingesetzt werden, etwa bei der sogenannten Software-Analyse, u.a. mit dem Ziel der Wartung eines Produkts. Das beinhaltet dann auch Änderungen, also fungieren die Schüler als Systembetreuer.

(4) Möglicherweise ist die Software auch für die Schulverwaltung von Bedeutung.

(5) Denkbar ist auch die Weitergabe der Software an andere Schulen oder an schulnahe Institutionen.

2. Wege zur erfolgreichen Konstruktion komplexer Software

2.1 Organisation der Arbeit
2.1.1 Ein Software-Life-Cycle

Projekte im Informatik-Unterricht werden in der Regel in Anlehnung an Methoden des Software-Engineering nach einem Software-Life-Cycle durchgeführt, wie er in Abbildung 2.1.1.a dargestellt wird. Die einzelnen Phasen stellen an den Bearbeiter unterschiedliche Anforderungen und die verwendeten Arbeitsformen sind unterschiedlicher Art.

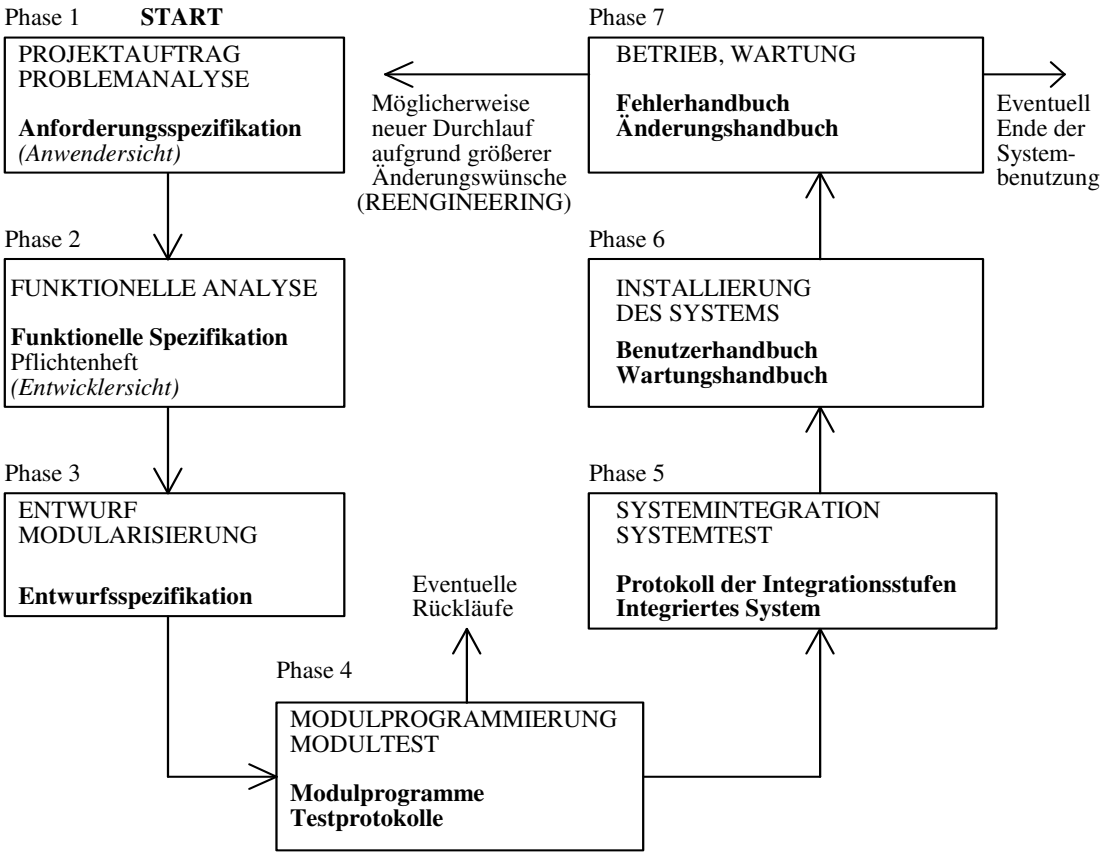


Abb. 2.1.1.a: Life-Cycle für Softwareprodukte

Die einzelnen Phasen sollen nun kurz kommentiert werden. An dem Beispiel TRABRENNEN werden sie später in Kapitel 3 ausführlicher dargestellt.

Zunächst muß bemerkt werden, daß der Software-Life-Cycle (wir schreiben kurz SLC) eine idealisierte Darstellung einer Softwareentwicklung ist. Es hat sich gezeigt, daß die hier genannten Phasen tatsächlich auftreten, daß aber oft Vermischungen zwischen den Phasen und insbesondere Rückgriffe auf Vorhergehendes nötig sind; vergleichen Sie hierzu die Ausführungen in dem Abschnitt "Prototyping" (Kapitel 2.1.2). Dennoch ist der SLC eine wesentliche Hilfe bei der Durchführung von Softwareprojekten.

Der Beginn eines Projektes zur Softwareentwicklung

Eine Softwareentwicklung beginnt mit dem Wunsch nach vollständiger oder teilweiser Automatisierung informationsverarbeitender Prozesse.

Beispiel 1: Eine Fachabteilung einer großen Firma will Vorgänge auf Datenverarbeitung umstellen und wendet sich zu diesem Zweck nach Genehmigung durch die Firmenleitung an die Software-Abteilung der Firma oder an ein Softwarehaus. Nun muß zunächst geprüft werden, ob sich die Vorstellungen der Abteilung wie gewünscht realisieren lassen und wie groß der Kostenaufwand sein wird. Mit diesen Überlegungen befindet man sich dann bereits in der Phase der Problemanalyse, die auch ergeben kann, daß die Umsetzung in ein passendes Softwareprodukt nicht möglich ist.

Beispiel 2: Der Fachbereich Englisch einer Schule äußert den Wunsch nach einem Vokabellernprogramm und wendet sich damit an den Informatiklehrer. Dieser möchte das Projekt mit seinem Informatikkurs realisieren. In diesem Fall wird das Projekt durch den "Auftrag" des Fachbereichs Englisch initialisiert.

Nun beginnt eine längere Zusammenarbeit (siehe "Prototyping") **zwischen den**

Softwareentwicklern
- Informatiklehrer als Projektmanager
- Informatikkurs als Projektgruppe

und den zukünftigen **Anwendern**
- Englischlehrer (Auftraggeber)
- Englischkurs (Benutzer, Helfer beim Testen)

Nach diesen Beispielen verschaffen wir uns zunächst einen Überblick über die einzelnen Phasen einer Softwareentwicklung. In diesem Zusammenhang wird auch auf das Kapitel 2.4 (Checklisten) verwiesen.

Phase I: Problemanalyse (Anforderungsspezifikation)

Die Problemanalyse beginnt mit der Untersuchung des **Einsatzfeldes** des geplanten Softwareproduktes. Der **Istzustand** muß aufgenommen, die **Rahmenbedingungen** müssen untersucht werden. Es geht also um das **Analysieren der informationsverarbeitenden Prozesse**, die ganz oder teilweise automatisiert werden sollen. Die Phase endet mit einem Schriftstück, in dem der Auftraggeber seine Anforderungen an das Programmsystem festlegt (**Anforderungsdefinition**). Insbesondere sind das der gewünschte Leistungsumfang des Systems und die herzustellenden Dokumente (Handbücher). Die Problemanalyse wird vom Auftraggeber durchgeführt, möglich ist aber auch schon hier eine Hilfestellung durch den späteren Entwickler. Die Problemanalyse kann also auch in enger Zusammenarbeit zwischen dem Auftraggeber bzw. späterem Anwender und dem Entwickler der Software stattfinden. Auch der Entwickler hat ja nach dem eigentlichen **Projektauftrag** eine Problemanalyse durchzuführen, die Grundlage für die von ihm zugesagten Leistungen (Pflichtenheft, siehe folgende Phase) ist. Je nach Zeitpunkt des Projektauftrags und Arbeitssituation kann also eine Vermischung der Phasen I und II (Problem-analyse und funktionelle Spezifikation) eintreten.

Phase II: Funktionelle Analyse (funktionelle Spezifikation, Pflichtenheft)

Nach Eingang des Projektauftrags (vor, während oder nach der Phase I) beginnt die Arbeit des Entwicklers. Dieser muß die **Durchführbarkeit** (und Wirtschaftlichkeit) des Auftrags untersuchen. Sollte diese gegeben sein, kann er dem Auftraggeber das für die Vertragspartner (Auftraggeber und Entwickler) verbindliche **Pflichtenheft** vorlegen. Dieses enthält die für das System vorgesehenen Funktionen, die Benutzerschnittstellen, Dateneingaben, Datenausgaben, Testdaten, Reaktion auf Fehleingaben. Eine wesentliche Hilfe für die Untersuchungen bildet eine Zerlegung des Problems in Teilaufgaben. Sollten sich die in der Anforderungsdefinition niedergelegten Vorstellungen nicht verwirklichen lassen, so kommt es zu einer Überarbeitung der Anforderungen oder sogar zur Aufgabe des Projekts.

Phase III: Entwurf, Modularisierung (Entwurfsspezifikation)

Das in der Regel sehr komplexe Gesamtsystem wird nun in **Einzelbausteine (Moduln)** zerlegt, Abb.2.1.1.b. Diese sollten unabhängig voneinander realisierbar sein. Ihr Zusammenwirken über **Import- und Exportschnittstellen** wird festgelegt. Man spricht von der **Modularisierung** des Systems. Diese führt zu einer **Modulhierarchie** und zu **Modulspezifikationen**, in denen Funktionen und Schnittstellen der Module beschrieben

werden. Die Verfeinerung der Module und der Beschreibungen geht bis in Prozeduren hinein. Auch hier zeigt sich der Vorteil der Verwendung bereits vorliegender (und schon dokumentierter) kleiner Bausteine. Für die Darstellungen sollten auch graphische Hilfsmittel benutzt werden, siehe Kapitel 2.2.4. Der Projektmanager entwirft einen **Zeitplan** für die Bearbeitung der Module und legt die **Zuständigkeiten** bei den Entwicklern fest.

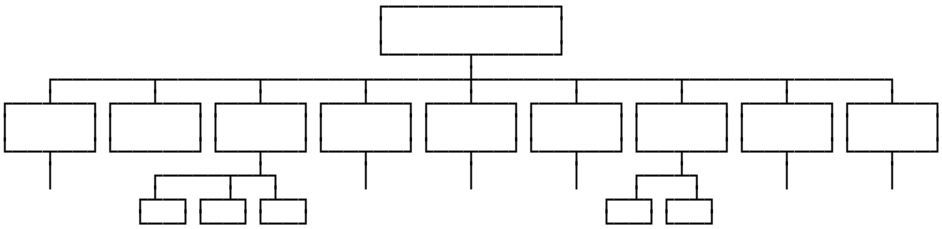


Abb. 2.1.1.b : Modulhierarchie

Phase IV: Modulprogrammierung, Modultest (Modulprogramme, Testprotokolle)

Die in der Entwurfsphase entwickelten Module werden in der ausgewählten Programmiersprache codiert und getestet. Dabei kommt es sehr auf die Beachtung der in den Vorphasen getroffenen Vereinbarungen an. Möglicherweise sind die Modulspezifikationen zu aktualisieren. Die **Programmierung** wird wesentlich vereinfacht und wird in kürzerer Zeit möglich, wenn man folgende Kriterien besonders beachtet:

(a) Anwendung der Methode der strukturierten Programmierung.
(b) Verwendung von Tool . Das sind z.B. Hilfsprogramme zur Menü- und Maskenerstellung.
(c) Verwendung fertig vorliegender Programmbausteine.
(c) Klare Schnittstellenfestlegungen durch Parametriesierung
(d) Verwendung aussagekräftiger Bezeichner für Prozeduren, Datentypen und Variablen.
(e) Übersichtliche Gestaltung des Programmtextes.
(f) Kommentartexte an geeigneten Stellen.

Phase V: Systemintegration, Systemtest (Protokoll der Integrationsstufen)

Die oben programmierten und für sich ausgetesteten Module müssen nun zu einem Gesamtsystem zusammengefügt werden, das alle Anforderungen der

Anforderungsdefinition erfüllt. Diese Phase erweist sich häufig als schwierig, insbesondere dann, wenn vorherige **Absprachen über die Schnittstellen** zwischen den Modulen nicht ausreichend erfolgt sind oder nicht eingehalten wurden. Für die **Systemintegration** ist eine schrittweise Erweiterung des Systems, bei der ein Menüpunkt nach dem anderen realisiert wird, empfehlenswert. Dabei ist das entstehende System jeweils wieder auf die Anforderungen hin zu testen.

Phase VI: Installierung des Systems, Abnahmetest (Abnahmeprotokoll)

Bei der **Installation** wird das Produkt in die Systemumgebung und die spezielle betriebliche Organisation des Auftraggebers eingebettet. Der Auftraggeber führt einen **Abnahmetest** durch. Außerdem erfolgt eine **Benutzerschulung** durch den Hersteller. Dabei werden auch Benutzerhandbuch und Wartungshandbuch übergeben.

Phase VII: Betrieb und Wartung (Wartungs- und Fehlerprotokolle)

Das Softwaresystem wird angewendet. Eventuell auftretende **Fehler** oder **Änderungswünsche** werden notiert. Gegebenenfalls werden Fehler vom Hersteller beseitigt. Zu gegebener Zeit erfolgen Änderungen am System durch die es an veränderte Gegebenheiten angepaßt wird. Häufig entstehen dabei neue Programmversionen. Der Vorgang der Anpassung wird auch als "**Reengineering**" bezeichnet. Hier setzt ein neuer Software-Life-Cycle ein, allerdings mit anderen Schwerpunkten (vergleiche Kapitel 4, "Software-Wartung").

2.1.2 Prototyping

Der in 2.1.1 beschriebene Software-Life-Cycle hat sich als wichtiges Konzept für die Entwicklung komplexer Software erwiesen. Er berücksichtigt jedoch nur in unzureichender Weise die Interessen des Auftraggebers oder des späteren Benutzers. Dieser formuliert zwar zu Beginn seine Wünsche und Anforderungen an die Software, mangels einschlägiger Kenntnisse kann er sich jedoch oft nicht so klar akzentuieren, daß die Softwarehersteller eindeutige Vorgaben haben. **Das Wissen von Anwender und Entwickler über das zu erstellende System ist also zu Beginn der Softwareentwicklung i.a. noch recht unsicher und unvollständig, so daß der Kommunikationsbedarf zwischen ihnen in der Regel besonders hoch ist.** Oft ist jedoch der erste Kontakt zwischen Kunde und Hersteller auch der letzte vor Fertigstellung des Produkts. So kann es leicht dazu kommen, daß der Benutzer später mit dem fertigen Produkt nicht vollständig zufrieden ist.

Häufig entwickeln sich die Anforderungen dynamisch während der Entwicklungszeit des Produkts!

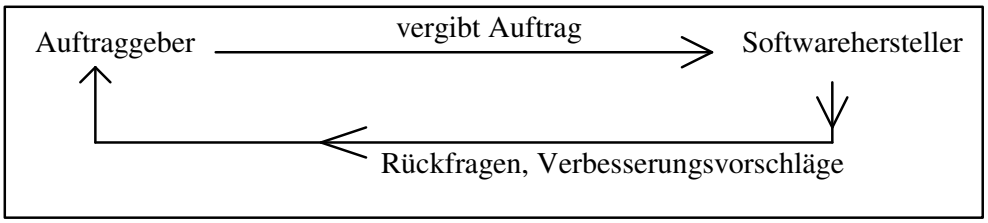


Abb.2.1.2.a: Kommunikationsbedarf zwischen Auftraggeber und Entwickler

Es geht also darum, den zukünftigen Benutzer während der gesamten Entwicklungszeit am Entstehungsprozeß der Software zu beteiligen. **Das wiederum bedeutet, daß bis hin zur Implementierungsphase der Software noch Änderungen am Produkt möglich sein müssen**, bzw. daß bei der Entwicklung komplexer Software eine Vermischung von Spezifikations- und Implementierungsphase stattfinden muß. Diese Forderung ist kennzeichnend für das sogenannte "**Rapid Prototyping**".

Beim Prototyping werden folgende Einzelschritte durchlaufen:

- Feststellung der Bedürfnisse des späteren Benutzers
- Spezifikation der Anforderungen
- Entwicklung eines Prototyps als Laborversion, meistens mit eingeschränkter Funktionalität
- Qualitätstest auf Funktionalität und Benutzbarkeit, insbesondere durch den späteren Benutzer
- Änderungen solange erforderlich.

Von besonderer Bedeutung für den späteren Anwender ist neben anderen Faktoren die Gestaltung der Oberfläche des Systems, also der sichtbaren Schnittstellen zwischen Mensch und Maschine. Die Oberfläche macht zu einem großen Teil die Benutzerfreundlichkeit des Systems aus. **Der spätere Anwender sollte also insbesondere bei der Gestaltung der Benutzeroberfläche mitbestimmen.**

Der hier verwendete Begriff des Prototyps ähnelt dem von den Ingenieuren benutzten Begriff des Prototyps, hat jedoch hier eine andere Bedeutung. Die Ingenieure bezeichnen die erste voll funktionsfähige Version eines Produkts als Prototyp. Mit ihm werden letzte Tests durchgeführt, bevor das Produkt möglicherweise in Massenproduktion geht.Bei der Softwareentwicklung unterliegt jedoch der Prototyp in der Regel weiteren Änderungen.

So erweist sich ein durch Rapid Prototyping entwickelter Prototyp als Grundlage der
- Kommunikation zwischen Benutzer und Hersteller,
- schrittweisen Einführung eines komplexen Systems beim Benutzer,
- schrittweisen Verfeinerung des Systems durch den Hersteller.

Prototypen kommen mit ihren Eigenschaften denen des zukünftigen Endprodukts zunehmend näher. Sie sind ablauffähige Versionen, die die Machbarkeit des Ganzen an eingeschränkten, jedoch repräsentativen Teilbereichen testen, bevor das Gesamtprodukt fertiggestellt ist. Den Prozeß des Rapid Prototyping kann man sich etwa wie in Abbildung 2.1.2.b dargestellt vorstellen.

Prototyping in der Schule

Für die Software-Entwicklung im Informatik-Unterricht an der Schule wird die Relevanz der Überlegungen in Bezug auf die Beteiligung eines späteren Anwenders sehr von der Themenstellung abhängig sein. Handelt es sich beispielsweise um den "Auftrag" eines Biologie-Lehrers, der ein Softwareprodukt zum Studium von Populationsentwicklungen benötigt, so wird eine ständige Rückkopplung mit ihm nötig sein. Programmieren wir jedoch ein Spiel, so können wir die Rolle des Anwenders selbst übernehmen und wohl auch von Anfang an eindeutige Anforderungen festlegen. Eine Rolle spielt dabei auch, daß die in der Schule entwickelbaren Systeme oft nicht die Komplexität und den Umfang haben wie in der Praxis.

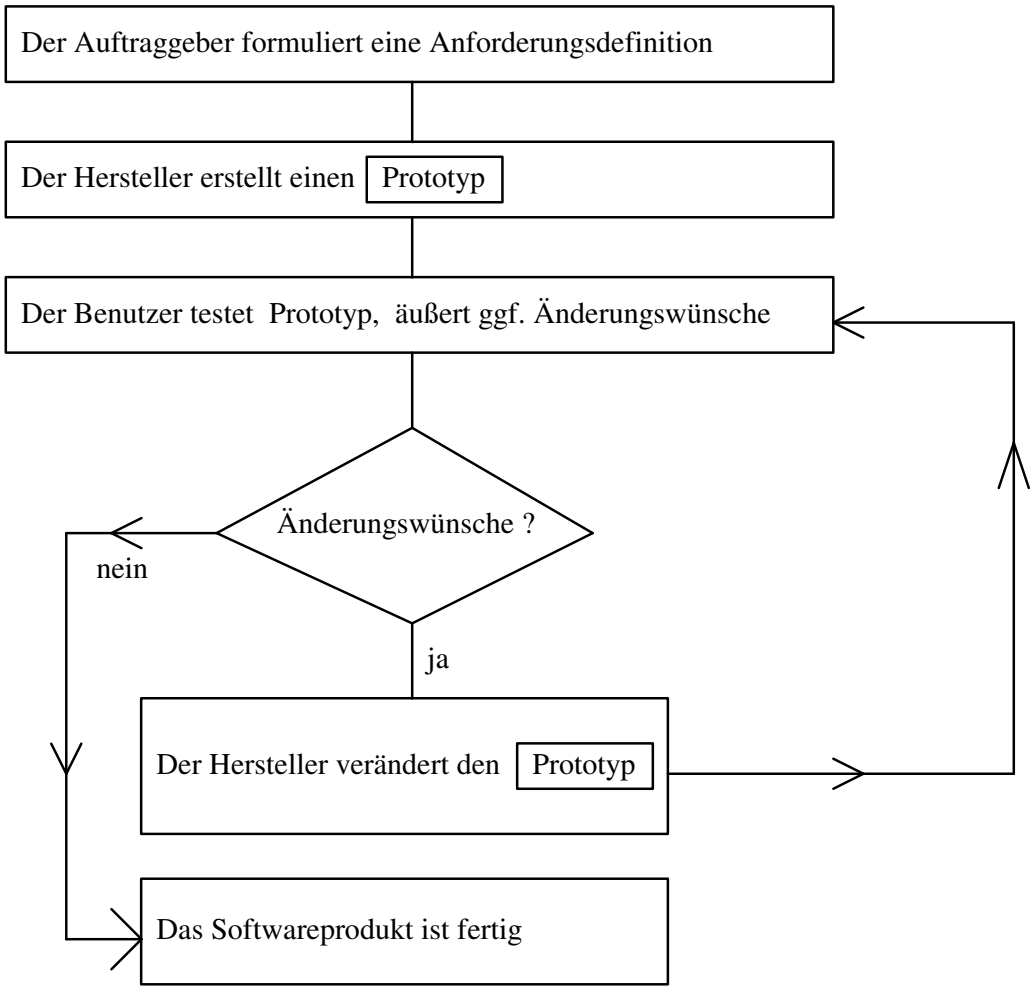


Abb. 2.1.2.b: Prototyping

Wir können also davon ausgehen, daß man in der Schule im wesentlichen weiterhin mit einem Software-Life-Cycle auskommen wird, wie er in Kapitel 2.1.1 dargestellt ist. **Wir können den SCL aber überlagern mit der ein oder anderen Idee des Prototyping:**

- Frühzeitige Erstellung der Benutzeroberfläche und Test auf Funktionalität und Benutzerfreundlichkeit,
- Offenhalten von Änderungsmöglichkeiten innerhalb einzelner Module.

2.1.3 Projektmanagement und Teamarbeit

Hinweis: Kapitel 2.4 handelt von der Projektkontrolle, spricht Warnungen aus und bietet Checklisten für den Projektleiter!

Die Konstruktion eines komplexen Softwareprodukts in vertretbarer Zeit kann in der Regel nur von einem Team geleistet werden. Die Anzahl der benötigten Mitarbeiter richtet sich nach dem Umfang und der Komplexität des Problems. Schon bei wenigen Mitarbeitern ergeben sich schnell Koordinierungs- und andere Probleme, die Führung des Projekts durch einen Projektmanager erforderlich machen. Entsprechende Überlegungen gelten auch für die Schule. **In der Schule wird in der Regel der Lehrer der Projektmanager (Projektleiter) sein; es ist aber auch denkbar, daß ein geeigneter Schüler Leitungsaufgaben übernimmt.** Innerhalb des Projektablaufs stellt sich die Frage

WAS muß WER WANN und WIE machen?

Projektmanagement in der Schule

Warum sollen sich auch Schüler mit den Problemen beschäftigen, die aus der Leitung eines Projekts entstehen?

Schüler arbeiten im Unterricht aller Fächer viel mit ihren Mitschülern zusammen - nun erfolgt die Zusammenarbeit auf einer anderen Ebene, nämlich in einem Team, in dem jeder Schüler seine besondere Verantwortung trägt. Nur das Zusammenwirken aller Mitarbeiter verspricht den Projekterfolg. Daher ist ein Bewußtmachen der Rollen der am Projekt Beteiligten - also auch der Rolle des Projektleiters - unerläßlich. Für den Projektablauf müssen immer wieder Entscheidungen getroffen werden, für den eigenverantwortlichen Anteil jedes Mitarbeiters und für umfassendere Probleme im Projektablauf durch den Projektleiter.

Wir gehen also davon aus, daß alle am Projekt Beteiligten am Gelingen des gewählten Projekts interessiert sind und damit auch die Notwendigkeit einer gut organisierten und verständnisvollen Zusammenarbeit einsehen.

Befassen wir uns also im Folgenden mit den Aufgaben des Projektleiters und mit den Anforderungen, die an ihn zu stellen sind!

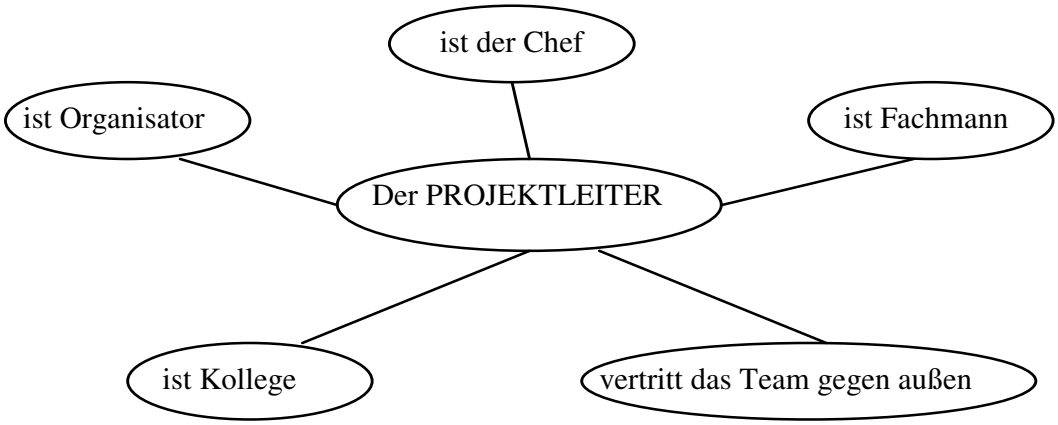


Abb. 2.1.3.a: Projektleitung

Interne Kommunikation

Der Projektleiter soll verwalten, leiten und führen!

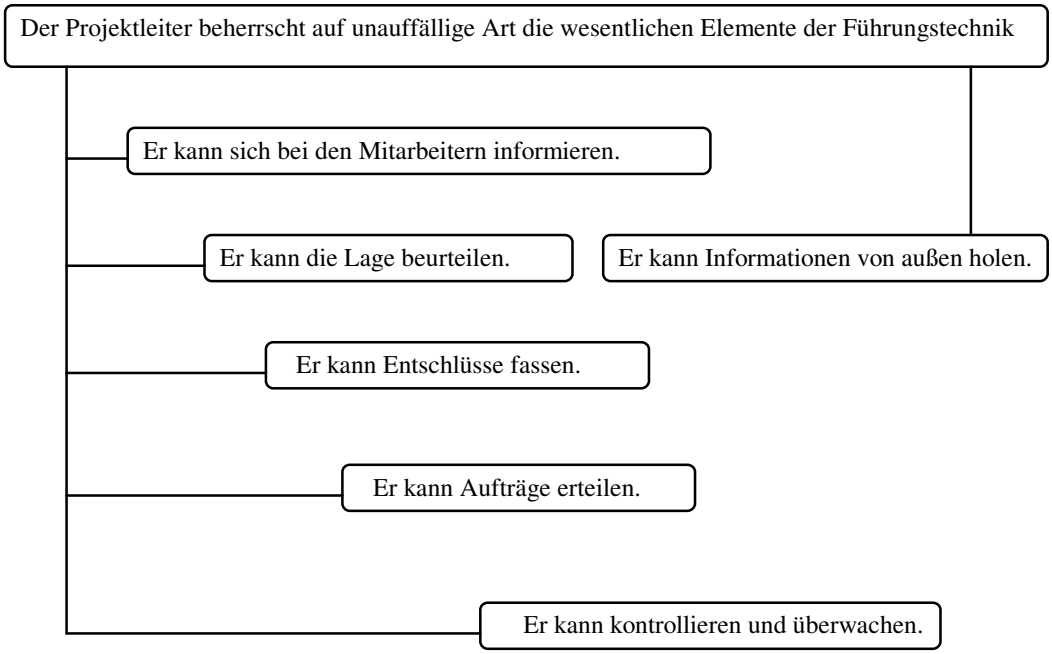


Abb.2.1.3.b: Führungstechnik des Projektleiters

Der Projektleiter in Aktion

Der Projektleiter informiert sich in den einzelnen Gruppen
- läßt sich über den Arbeitsstand berichten
- sichtet die neue entstandenen Dokumente
- achtet auf die Einhaltung der Prinzipien des Software-Engineering und
- insbesondere auf die des strukturierten Entwurfs und der strukturierten Programmierung
- achtet auf die Einhaltung der verabredeten Schnittstellen
- achtet auf die Übereinstimmung von Auftrag und tatsächlicher Durchführung
- achtet auf die angemessene Dokumentation der Ergebnisse
- fragt nach dem weiteren Fortgang
- gibt Hinweise für eventuelle Verbesserungen und für die Arbeit in der Gruppe
- ordnet die Ergebnisse in den Gesamtzusammenhang ein

Der Projektleiter organisiert und leitet regelmäßige gemeinsame Sitzungen
- Der jeweilige Arbeitsstand einer Arbeitsgruppe wird vorgestellt (mündliche Berichte, Folien, Demonstrationen, schriftliche Ausarbeitungen)
- Die anderen Gruppen gewinnen Überblick
- Führen von Protokollen über wichtige Fakten und Entscheidungen
- Erfahrungen werden ausgetauscht, Anregungen werden gegeben
- Die Arbeit wird koordiniert, der weitere Fortgang wird besprochen
- Entscheidungen werden gefällt
- Der Projektleiter setzt Termine ("Meilensteine") und achtet auf deren Einhaltung.

Der Projektleiter koordiniert die Gruppenarbeit
- Er ordnet die Ergebnisse in den Gesamtzusammenhang ein
- vervollständigt die Dokumentation,
- hält den Kontakt nach außen zum Auftraggeber und zum späteren Anwender aufrecht.

Die Kommunikation zwischen den Gruppen des Teams
- kann Schwächen in der Schnittstellen-Absprache aufdecken
- ist in der Schule nötig, denn leistungsstarke Schüler können anderen Schülern helfen
- darf jedoch nicht zu eigenmächtigen Entscheidungen ohne den Projektleiter führen.

Externe Kommunikation

Die ständige Kommunikation zwischen Auftraggeber und Entwickler ist dringend notwendig, um die gewünschte Qualität des herzustellenden Softwareprodukts zu sichern (siehe Kap. 2.1.2 Prototyping). Insbesondere achte man auf eine benutzerfreundliche Systemoberfläche. Zuständig für die Organisation des Kontakts nach außen ist der Projektleiter. Gespräche zwischen dem Auftraggeber oder dem späteren Benutzer und den Mitarbeitern des Teams sind erforderlich, damit das System nicht am

Benutzer vorbei entwickelt wird. **In der Schule können z.B. Softwareprojekte initiiert werden, die nützlich für den Unterricht in anderen Fächern oder für Verwaltungsaufgaben sind.** Hierzu einige Beispiele:

Biologie	*Räuber-Beute-Simulation*
Mathematik	*Demonstration der Polynomdivision*
Erdkunde	*Datenbank für geographische Grundbegriffe*
Verwaltung	*Buch-Leihverkehr*

Die späteren Benutzer (Lehrer, Schüler) stehen in häufigem Kontakt mit den Entwicklern (auch Schüler) und äußern ihre Wünsche, insbesondere zur Programmoberfläche.

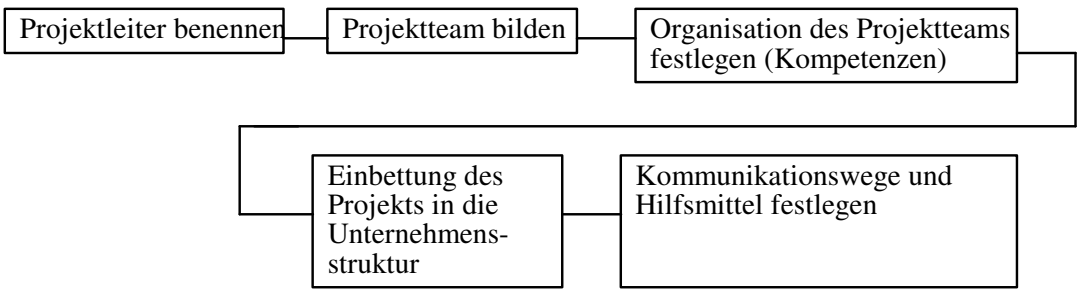
Die obigen Ausführungen zeigen die Notwendigkeit und Bedeutung einer guten Projektleitung. Sie weisen gleichzeitig die Komplexität der Vorgänge im Team nach und formulieren den hohen Anspruch, der auf dem Projektleiter (dem Lehrer) lastet.

Projektmanagement in einem Unternehmen

Unter Projektmanagement versteht man Organisationsformen und Tätigkeiten zur Planung, Steuerung, fachlichen Überwachung und (wirtschaftlichen) Kontrolle eines Pro-jekts. Projekte scheitern oft an einer ungeeigneten Organisationsform. So erfordert schon die Vorbereitung (erst recht die Durchführung) eines (Software-) Projekts seitens eines Unternehmens umfangreiche Maßnahmen:

- Projektanalyse (Vorstudie)
- Projektplanung: Personalplanung, Sachmittelplanung (Arbeitsplätze, Arbeitsmittel), Terminplanung, Kontrollplanung, Kostenplanung.

Kostenvergleichsrechnung, Nutzwertanalyse, Vorteile und Nachteile bei der Systemeinführung ergeben die Bilanz, die zur Projektentscheidung führt; ggf. kommt es zur Projektgründung. Falls die Projektfreigabe erfolgt ist, setzt das Projektmanagement ein. Die ersten Tätigkeiten bei der Projektorganisation sind



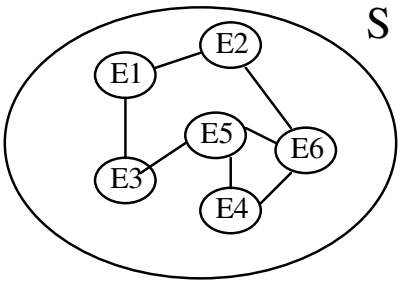
Der Projektleiter koordiniert den Personal- und Mitteleinsatz, er kontrolliert den Projektablauf und ist verantwortlich für das Erreichen der Projektziele. Er regelt den Mitarbeitereinsatz, so daß eine ausgewogene Arbeitsverteilung gewährleistet ist. Man erwartet von ihm fachliche Kompetenz, Geschick in der Mitarbeiterführung und Entscheidungsfreudigkeit.

2.2 Konzepte für die Konstruktion komplexer Software

2.2.1 Systemdenken und Modellbildung

Mit den Ausführungen zu den für die Informatik und auch für die Mathematik wichtigen Modellbildungsprozessen setzen wir die Betrachtungen von Kapitel 1.1 fort und vertiefen diese.

Ein System S ist eine Menge von Elementen E_i, die durch Beziehungen miteinander verbunden sind und gemeinsam einen bestimmten Zweck erfüllen.



Das System S besteht aus den Teilsystemen E1 bis E6. Diese sind durch Beziehungen miteinander verbunden.

Abb. 2.2.1.a: System und Teilsysteme

Ein Problem ist die Differenz zwischen dem, was vorhanden ist (dem IST), und einer Vorstellung von der Lösung des Problems (dem SOLL).

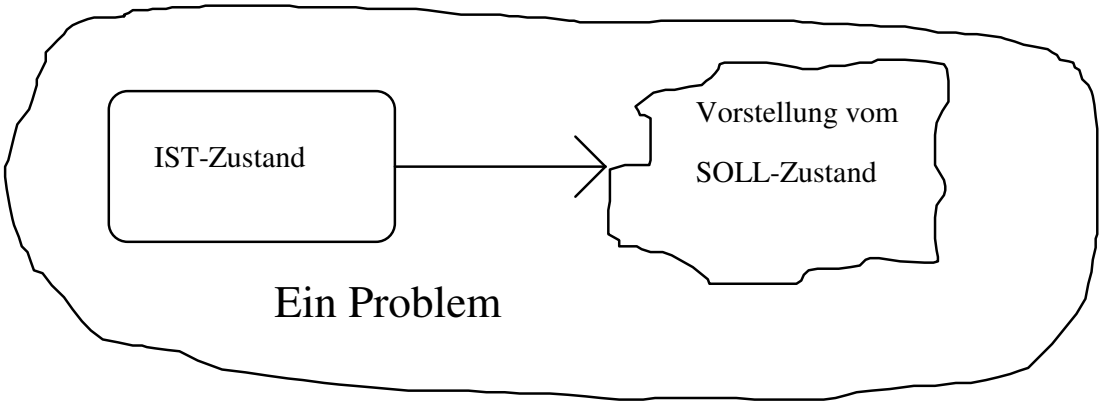
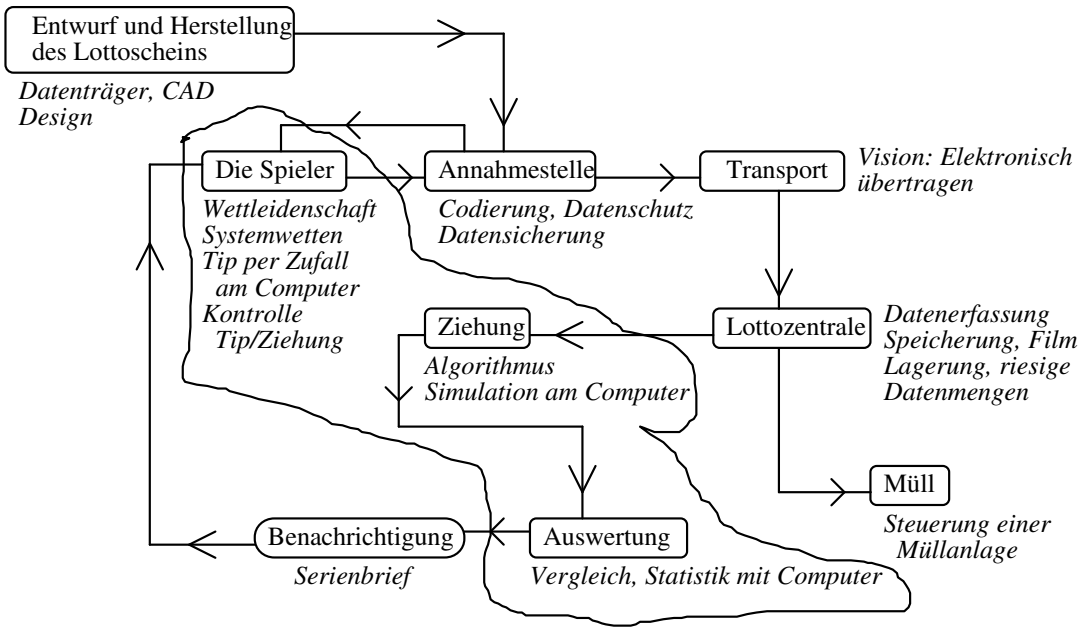


Abb. 2.2.1.b: Was ist ein Problem?

Das Systemdenken ist eine Denkweise, um komplexe Probleme einer Lösung zuzuführen.

Ausschnitte aus der Realität beschreiben.

Die folgende Abbildung zeigt eine Gedankensammlung, die das komplexe System "ZAHLENLOTTO" beschreibt. Mit dem Auftrag "Entwerfe eine Abbildung, die den >Lebenslauf eines Lottoscheins< beschreibt wurde es von Schülern erarbeitet, die außerdem den Auftrag hatten, nach Einsatzmöglichkeiten des Computers innerhalb dieses Systems zu suchen. So erklären sich die kursiv gedruckten Stichwörter. Für einen Ausschnitt aus dem System steht ein in Turbo-Pascal 6 programmiertes (und auch schon recht komplexes) Teilsystem LOTTO zur Verfügung, das die Bereiche *Tippen, Ziehung, Auswertung* umfaßt.



Der Lebenslauf eines Lottoscheins -

Stationen mit wesentlichen Tätigkeiten Rolle des Computers an den Stationen?

Abb. 2.2.1.c: Das komplexe System LOTTO

Das Programm LOTTO beschreibt ein Teilsystem. Es faßt mehrere Komponenten zu einem Ganzen zusammen. Die Komponenten bearbeiten wohlbestimmte Aufgaben, die u.a. im Menü des Programmsystems deutlich werden.

Ein System bearbeitet eine wohldefinierte Menge von Aufgaben. Man kann ein solches System in der Regel charakterisieren durch
- sein von außen sichtbares Verhalten
- seine innere Struktur
- seine Eigenschaften

Bei geschlossenen Systemen wird sein Verhalten nicht von außen beeinflußt. Ist das der Fall, spricht man von offenen Systemen.

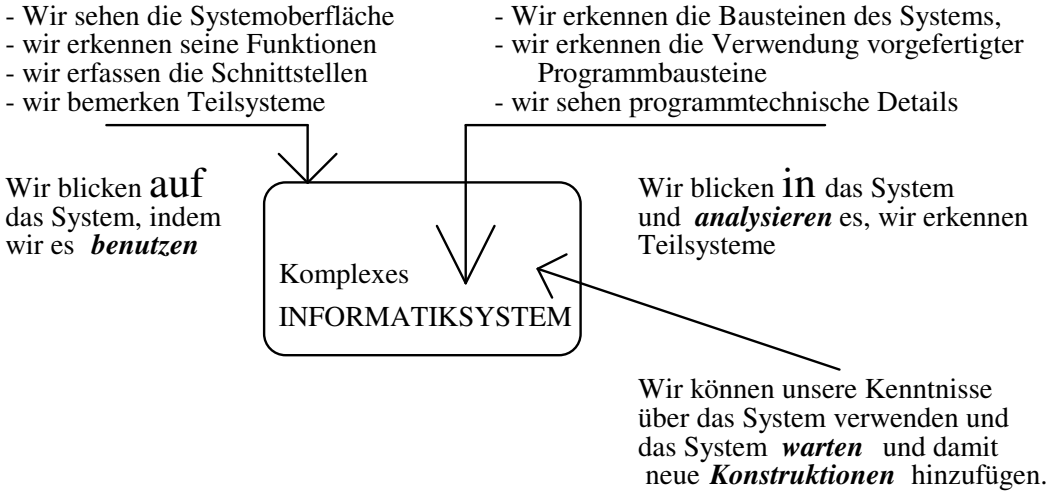


Abb. 2.2.1.d: Sichtweisen auf komplexe Systeme

Im Informatikunterricht interessieren uns an komplexen Systemen die Aspekte
- **benutzen, anwenden (die Sicht von außen auf das System)**
- **analysieren (der Blick in das System)**
- **konstruieren (der Bau eines neuen Systems)**
- **warten (die Anpassung eines vorhandenen Systems an neue Anforderungen).**

Der Zweck eines Softwareprodukts besteht in seiner Anwendung für von ihm verlangte Aufgaben. Schon die Anwendung und die Dokumentation verraten uns viel von dem **Leistungsumfang** des Systems, so daß wir damit schon den Anfang einer Systemanalyse vollzogen haben. Durch das **Hineinsehen in das System** (aus welchen Bausteinen besteht es, wie sind die Datenstrukturen, wie wurden die Teilprobleme programmiert ...?) setzen wir die Analyse fort. Diese Kenntnisse können wir nutzen, um das System an möglicherweise veränderte Gegebenheiten anzupassen, es zu warten.

Unser Hauptaspekt ist jedoch in diesem Buch die Konstruktion komplexer Systeme. Aber wir können aus vorhandenen Systemen viel für dieses Vorhaben lernen.

Wir erläutern die Ausführungen an dem Programmsystem LOTTO.

```
H A U P T M E N U E

1 Tippen,von Hand oder Computer
2 Ziehung u. Vergleich mit Tip·
3 Statistik·····················
4 Anleitung, Hilfen·····
5 Abbruch (Quit)···············

Bitte mit dem Cursor auswählen!
```

An dem Hauptmenü des Lottosystems erkennen wir die hauptsächlichen Funktionen Tippen, Ziehung, Auswertung (Statistik). Nach Aufruf von Option 1 können wir u.a. Tips von Hand eingeben:

```
Eine Option mit der Cursor (auf/ab)-Taste wählen und die
RETURN-Taste drücken!

HILFE    COMPUTERTIPS    HANDEINGABE     TIPS ANSEHEN    ZUM HAUPTMENÜ
```

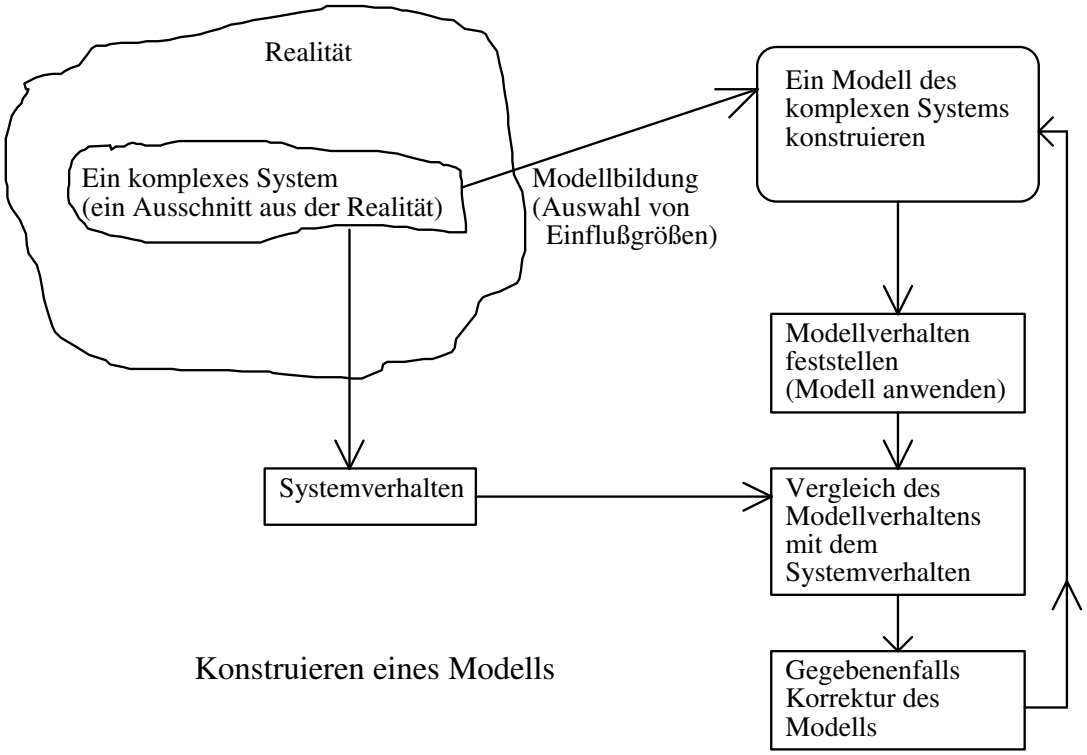
```
L O T T O
 6 aus 49

L O T T O S C H E I N

 1  2  3  4  5  6  7  8  9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49

Beenden mit "*" !
Name :
```

Ausgehend vom Hauptmenü können wir auch das Hilfemenü aktivieren.

```
          H I L F E - M E N U E

1.................Allgemeine Lottoregeln
2.................Tippen
3.................Ziehung
4.................Statistik / Auswertung
5.................Hauptmenue

Geben Sie Ihre Wahl ein :                ?
```

Modellbildung

In den Naturwissenschaften und der Technik dienen häufig Experimente dazu, Informationen zur Bestätigung oder Widerlegung von Hypothesen zu gewinnen. Modelle werden dagegen oft zur Lösung von Aufgaben eingesetzt, deren Durchführung am Original selbst nicht möglich oder zu aufwendig ist. Auch der Erstellung des Systems LOTTO ging eine Modellbildung voraus.



Konstruieren eines Modells

Abb. 2.2.1.e: Modelle konstruieren

2.2.2 Strukturiertes Programmieren

Top-Down und Bottom-Up

Beim Entwerfen und dem Realisieren umfangreicher Probleme in einer Programmiersprache kommt man ohne ein systematisches Vorgehen nicht aus. Das vorgegebene Problem wird in Teilprobleme zerlegt, jedes Teilproblem wird danach - falls noch nötig - wieder verfeinert usw. Schließlich kommt gelangt man auf eine Ebene, in der eine weitere Verfeinerung nicht mehr nötig erscheint, da die Teilprobleme so elementar sind, daß sie sich ohne weiteres in die gewählte Programmiersprache übersetzen lassen.

Die Verfeinerung betrifft sowohl die Algorithmen des Systems als auch die ihnen zugrundeliegenden Datenobjekte.

Auf die Datenstrukturierung wird weiter unten noch vertieft eingegangen.

Der Grad der Verfeinerung ist von verschiedenen Faktoren abhängig: Von

- den elementaren Befehlen der gewählten Programmiersprache
- dem Kenntnisstand des Bearbeiters
- den vorhandenen Hilfsmitteln , z.B.
  - Bausteinbibliotheken
  - Maskengeneratoren
  - wiederverwendbaren Teilen aus anderen Softwareprodukten

Die Beziehungen zwischen den einzelnen Teilproblemen werden durch Schnittstellen hergestellt.

Die Struktur eines Systems wird dann besonders deutlich, wenn man bei der Darstellung des Systems von nebensächlichen Details absieht, sich also auf die wesentlichen Merkmale konzentriert.

Eine gute Strukturierung ist anzustreben beim

- Systementwurf (durch Modularisieren und Hierarchisieren)
- Programmieren (durch Anwendung der Methoden der strukturierten Programmierung auf Daten und Algorithmen).

Strukturiertes Programmieren führt zu

- einer besseren Programmqualität
- einer Verkürzung der Programmierzeit
- besserer Verständlichkeit des Programms
- leichterer Testbarbeit
- leichterer Änderbarkeit (Wartbarkeit) des Programmsystems

- erleichtert bei komplexen Problemen das arbeitsteilige Vorgehen.

Für das strukturierte Programmieren sind folgende Aspekte wichtig:

- Zerlegung in Teilprobleme und damit zunehmende Verfeinerung (Top-Down-Vorgehensweise)
- die Zerlegung orientiert sich an den Funktionen oder Datenobjekten des Systems, sie ist "problemorientiert" (also nicht rechnerorientiert)
- die Teilprobleme können unbhängig von den anderen bearbeitet werden
- strukturiertes Programmieren bedeutet in der Regel ein Programmieren ohne Sprünge
- die Verbindung zwischen den Teilproblemen wird durch Schnittstellen realisiert
- manche Teilprobleme lassen sich so formulieren, daß ihre Lösung später wiederverwendet werden kann
- die Struktur des Ausgangsproblems findet sich wieder in der Struktur der Lösung des Problems.

Strukturiertes Programmieren wird u.a. unterstützt durch

- das Prozedurkonzept (siehe Kap. 2.2.3)
- das Modulkonzept (Kap. 2.2.4)
- Bausteinbibliotheken (Kap. 2.2.7)
- Menü- und Maskengeneratoren (Kap. 2.2.6)
- Verwendung wiederverwendbarer Softwareteile
- besondere graphische Darstellungsformen (Kap. 2.2.5).

Die oben beschriebene Top-Down-Methode wird durch die Bottom-Up-Methode er-gänzt, bei der man das Gesamtsystem von unten nach oben zusammensetzt, also bei der Programmierung mit der untersten Ebene beginnt. Man steigt also von einfachen Anwei-sungen auf zu Anweisungen höherer Komplexität. Manchmal erweist es sich als günstig, beide Methoden miteinander kombiniert einzusetzen.

Bei der Bottom-Up-Programmierung wird mit der Programmierung der untersten Teil-probleme begonnen, ohne daß ihr Bezug zum Gesamtsystem schon voll beschrieben ist. Für komplexe Systeme besteht durch den frühen Codierungsbeginn auf unteren Ebenen die Gefahr, daß die isoliert entwickelten Teillösungen später nicht zusammenpassen, so daß Neuentwicklungen nötig werden. Entsprechendes gilt für den Bottom-Up-Entwurf.

Strukturierung von Datenobjekten

Insbesondere bei umfangreicheren Problemen ist eine passende **Strukturierung der Datenobjekte**, auf denen die Algorithmen arbeiten, von großer Bedeutung. Man vergleiche hierzu auch das Kapitel 2.2.3 (das Modulkonzept), vor allem die Ausführungen zu abstrakten Datentypen.

Bezüglich der Datentypen wird bekanntlich unterschieden zwischen einfachen und bereits strukturierten Datentypen gemäß der Abbildung 2.2.2.a:

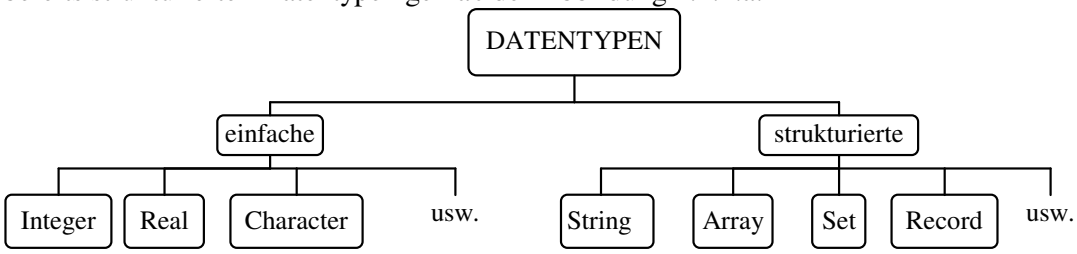


Abb. 2.2.2.a: Datentypen

Die genannten und weitere Datentypen können nun entsprechend der jeweiligen Problemstellung zu neuen, wieder strukturierten höheren Datentypen zusammengesetzt werden. Eine Menge von Daten, die durch logische Beziehungen zueinander eine Einheit bilden, bezeichnet man als eine **Datenstruktur.** Das folgende Beispiel zeigt die Datenstruktur eines Theaterplatzbuchungssystems. Das System arbeitet mit drei Dateitypen unterschiedlicher Struktur.

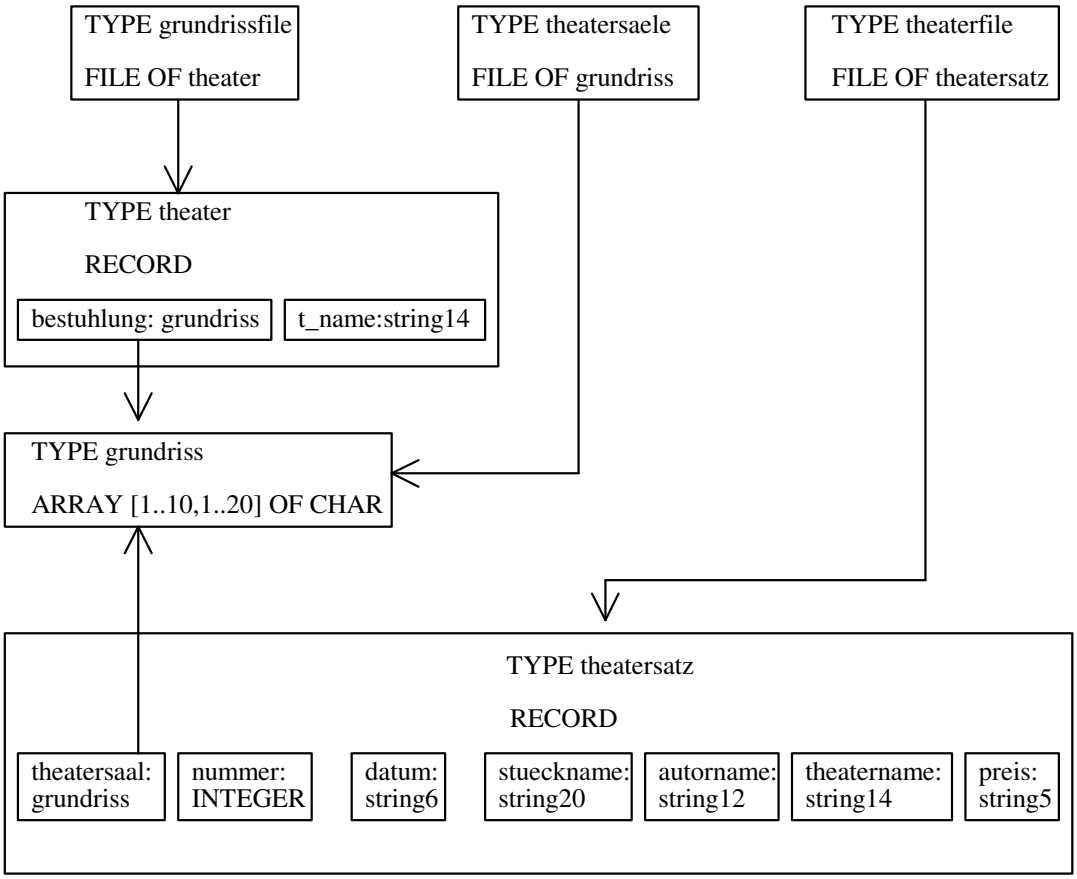


Abb. 2.2.2.b: Datenstruktur eines Theater-Platzbuchungssystems

```
(*Beginn Datenmodul*)
UNIT T_DMODUL;
INTERFACE
USES crt,ino_u,balk_u;
CONST satzanzahl=18;
TYPE grundriss   = ARRAY[1..10,1..20] OF CHAR;
     string5     = STRING[5];
     string6     = STRING[6];
     string14    = STRING[14];
     string12    = STRING[12];
     string20    = STRING[20];
TYPE theatersatz = RECORD
                     nummer: INTEGER;
                     datum : string6;
                     beginn: string5;
                     stueckname : string20;
                     autorname  : string12;
```

```
                    theatername: string14;
                    preis      : string5;
                    theatersaal: grundriss;
                  END;
TYPE theater     = RECORD
                    bestuhlung: grundriss;
                    t_name    : string14;
                  END;
TYPE grundrissfile = FILE of theater;
    theaterfile    = FILE OF theatersatz;
    theatersaele   = FILE OF grundriss;
    theatersaetze  = ARRAY[1..satzanzahl] OF theatersatz;
(*-------------------------------------------------------*)
VAR platz         : grundriss;
    theatername   : string14;
    theaterhaus   : theater;
    theaterhaeuser: grundrissfile;
    theaterdatei: theaterfile;
    einsatz : theatersatz;
    satzfeld: theatersaetze;
    satznummer: INTEGER;
    c:CHAR;
(*Ende Datenmodul*)
IMPLEMENTATION
BEGIN
END.
```

2.2.3 Das Prozedurkonzept

Zu den wichtigsten Strategien des Programmierens in imperativen Programmiersprachen wie PASCAL gehört das Prozedurkonzept. Was ist darunter zu verstehen?

Prozeduren bei der Zerlegung eines Problems in Teilprobleme

Viele Probleme sind so umfangreich und komplex, daß eine Zerlegung in Teilprobleme sinnvoll ist. Damit wird die Bearbeitung durchsichtiger und kann ggf. auch an verschiedene Bearbeiter delegiert werden. Für diese Teilprobleme kann man nun passende Prozeduren konstruieren, so daß sich schließlich die gesamte **Problemlösung als eine Sammlung geeignet angeordneter Prozeduren** darstellt.

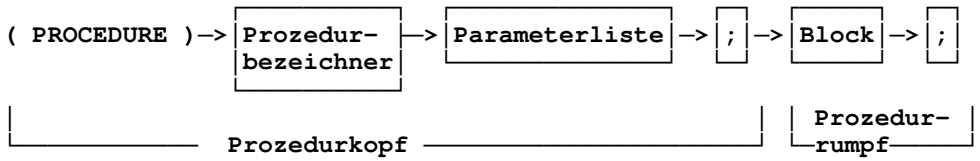
Wiederverwendbarkeit von Prozeduren

Man wird dabei auch bemerken, daß immer wieder Prozeduren benötigt werden, die die gleiche Aufgabe bearbeiten, etwa die Eingabe von Zeichenketten. Damit gewinnen manche Prozeduren einen universellen Charakter, so daß sie für eine Problemlösung mehrfach oder auch bei der Bearbeitung anderer Problemstellungen eingesetzt werden können. Das setzt allerdings eine geeignete, universell brauchbare Formulierung voraus. Derartige Prozeduren sollte man in Bibliotheken zusammenfassen, um sie immer auf einfache Weise zur Verfügung zu haben. Dieser Aspekt wird in Kapitel 2.2.4 beim Modulkonzept ausführlich behandelt.

Definition von Prozeduren

Prozeduren müssen im Vereinbarungsteil des übergeordneten Programmteils definiert werden. Wie das Syntaxdiagramm zur Prozedurvereinbarung zeigt, haben Prozeduren einen entsprechenden Aufbau wie ein Programm; sie werden jedoch mit "END;" beendet. Die Prozeduraufrufe können an jeder Stelle des Anweisungsteils des Programms erfolgen. Zwei Syntaxdiagramme erläutern die Zusammenhänge zwischen der Prozedurvereinbarung und dem Prozeduraufruf.

Prozedurvereinbarung

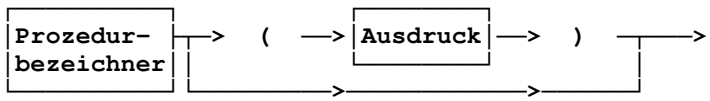


Beispiel:

```
PROCEDURE Position(x,y:INTEGER);
                   formale Parameter
```

```
BEGIN ... END;
```

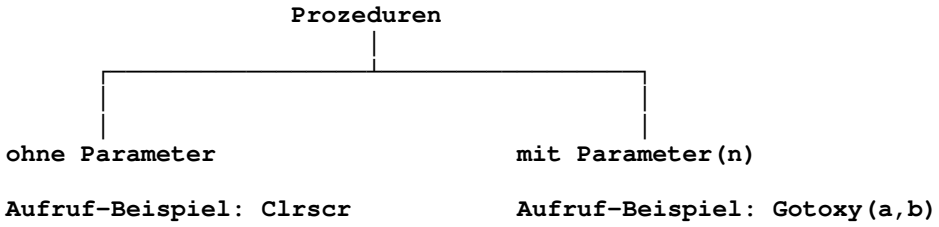
Prozeduraufruf



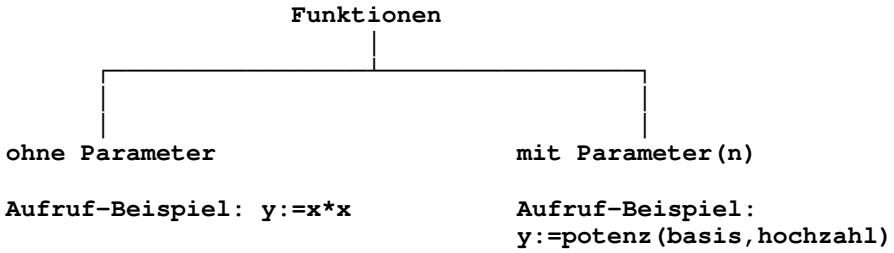
Die im Prozedurkopf möglicherweise genannten Parameter sind sogenannte "formale Parameter". Beim Prozeduraufruf werden sie ersetzt durch die "aktuellen Parameterwerte". Dabei muß auf gleiche Reihenfolge der Parameter geachtet werden.

Prozedurarten

Es gibt verschiedene Arten von Prozeduren.



Eine besondere Art von Prozeduren sind die Funktionen.



Ein Funktionsaufruf liefert also stets einen Wert zurück und ist damit Teil einer Anweisung, während eine Prozeduraufruf eine eigene Anweisung ist.

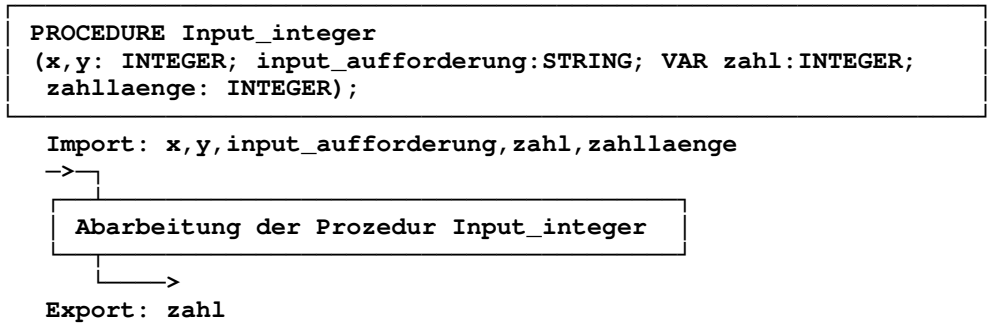
Datenaustausch zwischen Programmteilen (Schnittstellen)

Die Prozeduren kommunizieren mit den anderen Programmteilen

a) durch globale Variable - in der Regel ein schlechter Weg, da oft unerwünschte Nebenwirkungen bei Bearbeitung der Variablen in der Prozedur eintreten und der Durchblick über das Verhalten der Variablen leicht verloren geht,

b) über die Parameter (Eingangsparameter, Ausgangsparameter).

Beispiel:



x,y,input_aufforderung,zahllaenge sind Eingangsparameter, *zahl* ist Ausgangsparameter (auch Übergangsparameter) und wird an den übergeordneten Programmteil zurückgegeben.
Nach dem Aufruf einer Prozedur und deren Abarbeitung kehrt das Programm zu der Anweisung, die auf den Prozeduraufruf folgt, zurück.

Beschreibung von Prozeduren (Prozedurspezifikation)

Für Dokumentationen benötigt man eine geeignete Beschreibungsform von Prozeduren, aus der die wichtigsten Eigenschaften der Prozeduren schnell ersichtlich sind. Es folgt ein Vorschlag für eine derartiges Formular:

```
DOKUMENTATION DER PROZEDUR:
LETZTES UPDATE:                                  AUTOR:
-------------------------------------------------------------
PROZEDURKOPF:

FUNKTIONSBESCHREIBUNG:


-------------------------- SCHNITTSTELLEN ---------------------
A) PARAMETER (IMPORT / EXPORT)      - B) PROZEDUREN (IMPORT)


C) GLOBALE VARIABLE (WERTÄNDERUNG?) - D) DATEIEN (WERTÄNDERUNG?)


E) AUFRUFBEISPIEL:
```

SONSTIGES (z.B. bekannte Fehler):

2.2.4 Das Modulkonzept

Module in technischen Anlagen

Eine HIFI-Anlage ist modular aufgebaut, z.B. aus *Empfänger/Verstärker, Plattenspieler, CD-Spieler, Kassettenrecorder.* Die Anlage besteht aus verschiedenen Bauteilen (Modulen). Die Einzelgeräte sind weitgehend unabhängig voneinander und können z.B. nacheinander gekauft und zusammengesetzt werden. Sie können auch einzeln repariert bzw. gegen neuere Teile ausgetauscht werden. Die Vorteile einer solchen Konzeption sind klar ersichtlich! Weitere Beispiele solcher modular aufgebauten Systeme lassen sich leicht finden.

In der Datenverarbeitung wird der Begriff des Moduls in unterschiedlichen Bedeutungen verwendet.

Module als Bausteine von Software-Systemen

```
                          HAUPTMENÜ
-------------------------------------------------------------
 1: Komponist (Autor) bekannt     --> weitere Daten suchen
 2: Stichwort         bekannt     --> weitere Daten suchen
 3: Dokument-Titel    bekannt     --> weitere Daten suchen
 4: Dokument-Nummer   bekannt     --> weitere Daten suchen
-------------------------------------------------------------
 5: Ausgabe einer Stichwortliste,                  sortiert
 6: Ausgabe einer Komponistenliste (Autorenliste), sortiert
-------------------------------------------------------------
 7: Anzahl der Datensätze in der Datei angeben
 8: Ausgabe aller Datensätze
 9: Ausgabe eines bestimmten Datensatzes
-------------------------------------------------------------
 E: Eingabe von Datensätzen (Dokumenten)
 A: Änderungen der Datei
 P: Datei packen (aktualisieren / Leersätze löschen)
-------------------------------------------------------------
 F: Info-Menü / Dateiname / Drucken / ▌ Programmende ▐
-------------------------------------------------------------
Auswahl einer Option mit den Cursortasten AUF/AB, danach <┘
```

Das Menü stammt aus einem Softwareprodukt MUSIK, mit dem man Daten über Schallplatten, CDs, Musikkassetten usw. speichern und bearbeiten kann, wie z.B. Komponist, Musikstück. Man erkennt an den einzelnen Menüpunkten, daß das System offenbar aus mehreren Teilen mit unterschiedlichen Funktionen besteht. Diese Teile kann man auch als Bausteine (Module) bezeichnen.

Aufgabe: Inwiefern sind die genannten Bausteine zur Realisierung des Systems in einer Programmiersprache geeignet?

MODUL (Funktionsmodul)
Ein Software-System besteht aus Bausteinen, die in ihrem Zusammenwirken die von dem System geforderte Leistung erbringen. *Die Bausteine heißen Module*, und die Beziehungen zwischen ihnen werden durch *Schnittstellen* beschrieben.

Bei dieser Interpretation des Modulbegriffs kann man auch von **Funktionsmodul** sprechen, da die Zerlegung des Problems wie z.B. in dem obigen Menü nach den gewünschten Funktionen des Systems erfolgt. Neben dieser mehr **funktionsorientierten Sicht** kann man aber auch die benötigten Datentypen und Datenstrukturen mehr in den Vordergrund rücken **(datenorientierte Sicht).** So können z.B. die wesentlichen Datentypen und Datenstrukturen in einem Modul zusammengefaßt werden - ein **Datenmodul** entsteht. Die Operationen mit diesen Daten sind dann jedoch möglicherweise zusammen mit weiteren Prozeduren auf die verschiedenen Funktionsmodule des Systems verteilt. **Aus datenorientierter Sicht ist es deshalb vorteilhaft, wenn die Datentypen und Datenstrukturen mit den darauf definierten Operationen zusammengeführt werden. Man gelangt so zu dem Begriff des Datentyps und des abstrakten Datentyps**.

Datentypen - abstrakte Datentypen

Unter einem Datentyp wird die Zusammenfassung von Wertebereichen und Operationen auf diesen Wertebereichen zu einer Einheit verstanden.

Dabei unterscheidet man zwischen abstrakten und konkreten Datentypen:

- Abstrakter Datentyp: Die Eigenschaften der Operationen und Wertebereiche stehen im Blickpunkt der Betrachtung (und werden vom Anwender benutzt, Sicht des Anwenders). Dabei werden die Eigenschaften ohne Bezugnahme auf den internen Aufbau des Datentyps beschrieben (spezifiziert).
- Konkreter Datentyp: Die konkrete Realisierung in einer Programmiersprache steht im Blickpunkt. Wie ist der Datentyp aus einfacheren Datentypen zusammengesetzt? (Sicht des Konstrukteurs des Datentyps).

Der Begriff "Datentyp" wird häufig auch an Stelle des Begriffs "Datenstuktur" verwendet. In engerem Sinn versteht man aber unter einer **Datenstruktur** die Konstruktion von Wertebereichen aus elementaren Wertebereich (die Datenstruktur RECORD z.B. aus INTEGER und BOOLEAN).

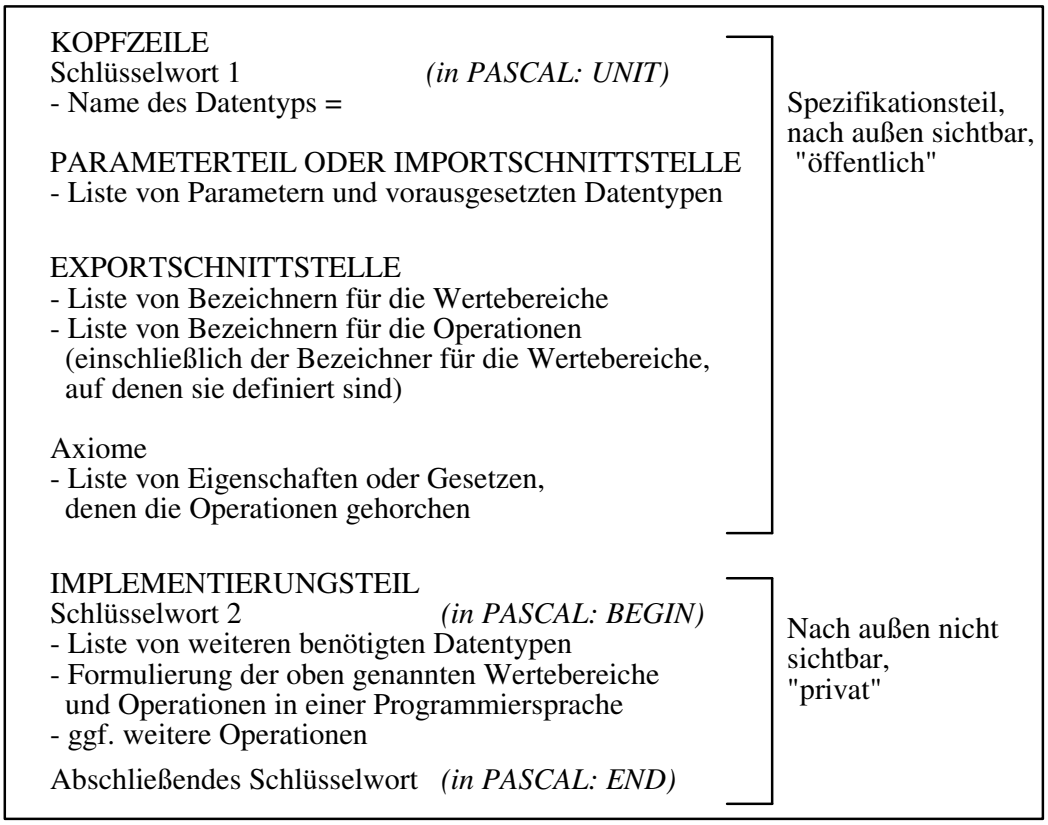


Abb 2.2.4.a: Formaler Aufbau von abstrakten Datentypen

Wir erläutern zunächst an einem allgemein bekannten Beispiel, dem Datentyp INTEGER, daß der Begriff der Operation hier recht weit zu verstehen ist. Man denkt bei ganzen Zahlen zunächst an die Operation +, -, *. Weitere Operationen auf dem Wertebereich von INTEGER sind jedoch z.B.

Wertzuweisung an eine Variable: *nummer:=2;*
Eingabe über Bildschirm : *READLN(nummer);*
oder auch als Baustein : *Input_integer (1,2,'Eingabe=',intwert,2);*
Ausgabe auf den Bildschirm : *WRITE(zahl);*
Vergleiche (<, =, >) : *IF nummer1<nummer2 THEN ...*

Dieser Datentyp ist allerdings unstrukturiert, da INTEGER ein besonders einfacher Standarddatentyp ist. Wir betrachten daher als weiteres Beispiel einen komplexeren strukturierten Datentyp, der sich bei der Arbeit an einem Projekt SKI ergeben hat.

Der (konkrete) Datentyp s_date, formuliert in Turbo-PASCAL:

```
{ Kopfzeile }
UNIT s_date;
INTERFACE

{ Importschnittstelle }
USES Ino_u,Crt,hilfen_u, s_hilf_u,bild_u;

{ Exportschnittstelle                                    ("öffentlicher Teil")}
{ Liste von Bezeichnern für die Wertebereiche }
TYPE farbe      = (blau,rot,schwarz);
TYPE string20   = STRING[20];
TYPE string3    = STRING[3];
TYPE belag      = (normal,vereist,steinig);
TYPE betrieb    = (offen, geschlossen);
TYPE satzinhalt = (voll, leer);
TYPE pistensatz = RECORD
          pistennummer : string3;
          pistenname   : STRING;
          schwierigkeit: farbe;
          lifte_unten_an, lifte_unten_ab, lifte_oben_an,
          Lifte_oben_ab       : ARRAY[1..5] OF string3;
          pisten_unten_ab,
          pisten_oben_an      : ARRAY[1..5] OF string3;
          huetten_oben,
          huetten_unten       : ARRAY OF string20;
          pistenbelag         : belag;
          pistenbetrieb       : betrieb;
          pisten_satzinhalt   : satzinhalt;
     END;
TYPE pistendateien       = FILE OF pistensatz;
TYPE art_des_datensatzes = (satz_neu,satz_aendern);
VAR  pistendatei         : pistendateien;

{ Liste von Bezeichnern für die Operationen auf s_date }
PROCEDURE pistensatz_initialisieren1
     (VAR pisten_datensatz:pistensatz);
PROCEDURE pistensatz_initialisieren2
     (VAR pisten_datensatz:pistensatz;
      VAR datensatznummer :INTEGER);
```

```
PROCEDURE pistensatz_eingeben
     (name_des_skigebiets:STRING;
     VAR pisten_datensatz: pistensatz;
     VAR satz_uebernehmen: CHAR;
     VAR zurueck: BOOLEAN;
     VAR eingabeart: art_des_datensatzes;
     VAR datensatznummer: INTEGER);
PROCEDURE datensaetze_eingeben
     (VAR name_des_skigebiets: STRING;
      VAR pistendatei : pistendateien;
      VAR eingabeart: art_des_datensatzes;
      VAR datensatznummer: INTEGER);
PROCEDURE einen_datensatz_ausgeben(satznummer:INTEGER);
PROCEDURE datensaetze_loeschen;
PROCEDURE pistendatei_lesen;
PROCEDURE in_pistendatei_suchen;
PROCEDURE pistendatei_verkuerzen;

{ Implementierungsteil                          ("privater Teil") }
IMPLEMENTATION
...
PROCEDURE einen_datensatz_ausgeben(satznummer:INTEGER);
VAR dateiname        : STRING;
    pisten_datensat z: pistensatz;
    zeile,spalte     : INTEGER;
BEGIN
... {Anweisungen der Prozedur einen_datensatz_ausgeben}
END;
...

BEGIN
...
END.
```

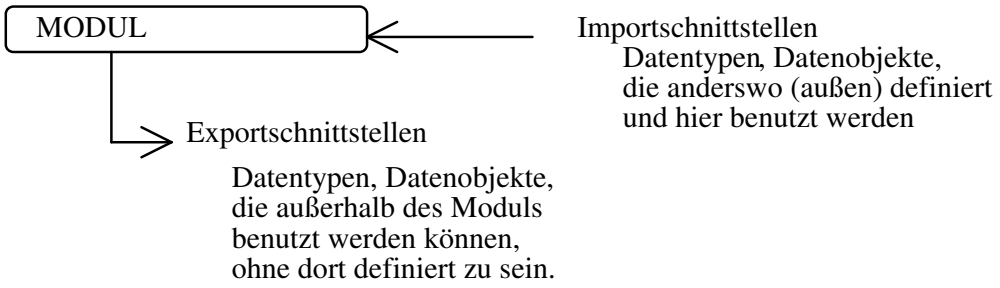
Wir erkennen im ersten Teil die Datenstruktur, im zweiten Teil die auf ihr definierten Operationen. Der Datentyp s_date wurde konstruiert in dem formalen Aufbau von Abbildung 2.2.4.a. Axiome sind in dem gewählten Beispiel s_date nicht vorhanden. Beim Datentyp INTEGER würde man z.B. das Axiom a+b=b+a verlangen.

Es wurde unterschieden zwischen

- **Öffentlicher Teil des Datentyps --> das interessiert den Benutzer**
- **Privater Teil des Datentyps --> das interessiert nur den Konstrukteur des Datentyps**

Module als abstrakte Datentypen

Der Begriff des Moduls wird auch in Zusammenhang mit dem oben erklärten Begriff des abstrakten Datentyps verwendet, indem man sagt:



Abstrakte Datentypen können gemäß ihrer Spezifikation in größeren Programmsystemen als Modul benutzt werden. In dieser Sichtweise wird das **Modul als abstrakter Datentyp** aufgefaßt.

Hinweis: Beachten Sie in diesem Zusammenhang auch Kapitel 2.2.7 (Baustein-Bibliotheken). Dort werden Bausteine in PASCAL-Units zusammengefaßt. Jede solche Unit kann als Modul (abstrakter Datentyp) aufgefaßt werden.

Eigenschaften von Modulen

Von einem Modul werden folgende Eigenschaften verlangt:

- Logische und funktionale Abgeschlossenheit
- wohldefinierte Schnittstellen nach außen
- Schnittstellen- Minimalität, möglichst wenig Schnittstellen
- Geheimnisprinzip - die interne Arbeitsweise des Moduls bleibt nach außen, d.h. für den Anwender unbekannt (information hiding)
- gegenseitige Nicht-Beeinflussung von Modulen
- Überschaubarkeit , übersichtlich und nicht zu umfangreich
- leichte Testbarkeit
- leichte Integrierbarkeit in das jeweilige Gesamtsystem
- Austauschbarkeit gegen andere Module gleicher Funktionalität

Beschreibung von Modulen (Modulspezifikation)

In der Dokumentation von Modulen stellt man ihre wichtigsten Eigenschaften zusammen, insbesondere um spätere Änderungen am Modul (durch einen

Programmierer) leichter vollziehen zu können. Ein mögliches Beschreibungsschema wird im folgenden vorgestellt.

Modulspezifikation (Modulbeschreibung)

Modulname **Letztes Update / Autor:**
Modulfunktionen:
Schnittstellen (Import / Export) *hier können je nach Auftreten folgende Angabe stehen* **Benutzte Module** **Benutzte globale Variable** **Prozeduren, ihre Bedeutung** **Parameter (Art, Bedeutung)**
Aufrufbeispiel:
Sonstiges:

Beispiel einer Modulspezifikation

Modulname WETTEN ANNEHMEN (T_M3_U, Modul 3 des Programmsystems)

Letztes Update / Autor: 28.11.88, Gruppe 3
Modulfunktionen: Dieses Modul ermöglicht das Ausfüllen eines Wettscheines von Hand oder per Zufall. Die Eingaben erfolgen in eine Wettscheinmaske. Der Wettschein wird abgespeichert. Eine weitere Option erzeugt gleich 10 zufällige Wettscheine auf einmal. Die Wettscheindatei kann ausgegeben werden.
Schnittstellen (Import/Export) **Benutzte Module:** Diverse Prozeduren aus den allgemein verwendbaren Modulen Crt,Ino_u, Balk_u, Bild_u und dem programmspezif. Modul t_hi_u, Datentypen aus Datenmodul t_ds_u **Benutzte globale Variable:** Keine **Prozeduren** Wetten_abschließen (renntag:STRING8; *Eingabeparameter, z.B. 13.08.94* renntag_feld: feld_of_renntag; *Eingabeparameter, Renndaten für einen Renntag* VAR datei:file_of_wettschein; *Ausgabeparameter, Datei mit den Wettscheinen* VAR schein_anzahl:INTEGER); *Anzahl der eingegebenen Scheine* scheine_datei_aus; *keine Parameter*
Sonstiges:

Modulhierarchien

Unter einer **Modulstruktur** versteht man die Menge aller Moduln eines Programmsystems und die Menge aller Beziehungen zwischen ihnen. Ein Modul A benutzt einen Modul B, wenn er zur Realisierung seiner Aufgaben Daten oder Operationen aus B einbezieht. Das geschieht über das Interface des Moduls B. A erteilt also B Aufträge, B führt diese aus und gibt die Resultate der Ausführung wieder an A.

An die Modulstruktur werden einschränkende Forderungen gestellt: Voneinander unabhängige Entwickelbarkeit der Module, Übersichtlichkeit, einfache Wartbarkeit, Überprüfbarkeit der Korrektheit, Möglichkeit des Austauschens von Modulen mit anderen Modulen gleicher Funktion.

Daraus resultiert u.a., daß zyklische Teilstrukturen zwischen Modulen vermieden werden müssen. Vielmehr sollen durch Verwendung der top-down-Methode (von oben nach unten) **baumartige Strukturen** entstehen. Die Funktionen der Moduln jeder Entwurfsebene werden nur durch die Funktionen der unmittelbar darunterliegenden Ebene verwirklicht. So entsteht eine Hierarchie von Verfeinerungsstufen (**Modulhierarchie**).

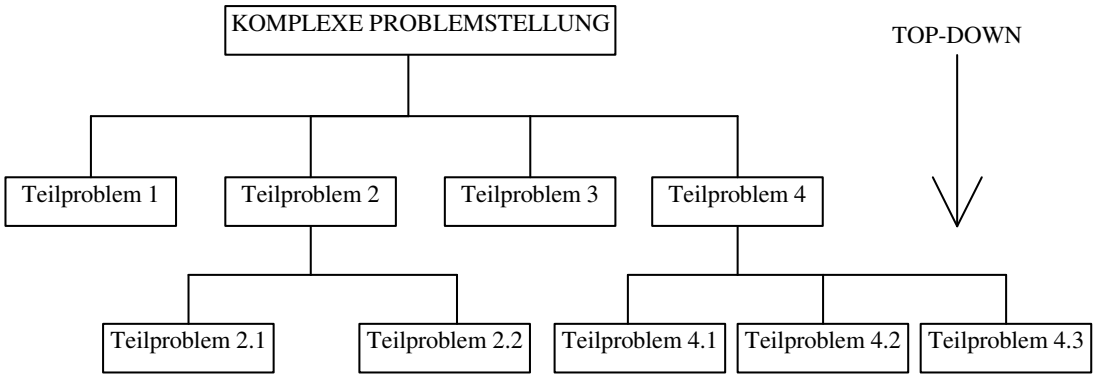


Abb.2.2.4.b: Modulhierarchie

2.2.5 Graphische Darstellungsmittel

Struktogramme

Bei der Bearbeitung komplexer Probleme sind zahlreiche Algorithmen zu entwerfen. Ein bewährtes Hilfsmittel zur Veranschaulichung von Algorithmen sind die Ihnen wohl schon bekannten Struktogramme. Hier werden dennoch einige Ausführungen angeschlossen, die die Bedeutung dieser Darstellungsform unterstreichen und auf die Beschreibungstiefe von Struktogrammen eingehen.

Struktogramme unterstützen das strukturierte Entwerfen und Programmieren

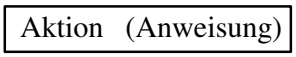
- Sie zeigen den Ablauf eines Algorithmusses in übersichtlicher Form und beschreiben ihn in Kurzform
- sind häufig die Vorstufe vor seiner Programmierung
- dienen zur Dokumentation von Algorithmen, so daß spätere Änderungen leichter durchgeführt werden können

Als Hilfsmittel zur Zeichnung von Struktogrammen bieten sich einfache Grafikprogramme an, mit denen man die Symbole - im wesentlichen Rechtecke und Dreiecke - malen und mit dem gewünschten Text füllen kann. Solche Grafikprogramme sind in der Regel in Textverarbeitungssysteme integriert. Die Grafik kann also leicht in den Text eingefügt werden. Es werden jedoch auch Hilfsprogramme angeboten, mit denen man Struktogramme entwerfen kann.

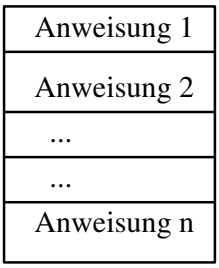
Struktogrammsymbole

Die Aktionen eines Algorithmusses werden in geeigneter Form in Strukturblöcke eingetragen. Diese Strukturblöcke können je nach Problemstellung beliebig ineinander geschachtelt sein.

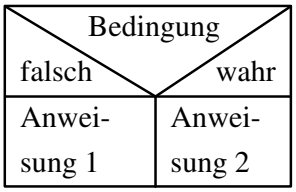
Die Darstellung kann in unterschiedlicher Beschreibungstiefe erfolgen, indem ein grobes Struktogramm zunehmend verfeinert wird, bis eine Stufe erreicht ist, die sich leicht in einer Programmiersprache realisieren läßt. Das müssen nicht unbedingt die elementaren Anweisungen der Sprache sein, gelegentlich ist die Übersetzungsstufe schon erreicht, wenn sich für den Teilalgorithmus ein passender Baustein findet. Das folgende Beispiel verdeutlicht diesen Aspekt und zeigt gleichzeitig die Mächtigkeit dieses Algorithmen-Beschreibungsmittels.



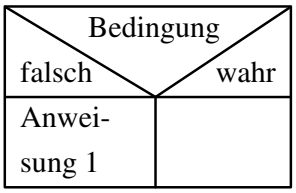
Einzelner Strukturblock



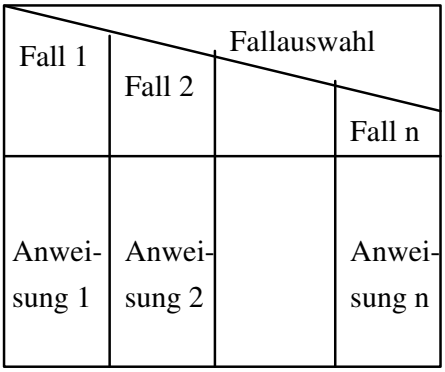
Aneinanderreihung von Strukturblöcken



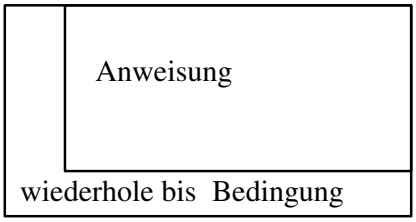
Bedingte Anweisung
(IF THEN ... ELSE)



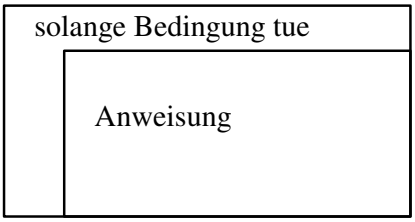
Bedingte Anweisung
(IF THEN ...)



Fallunterscheidung (CASE ... OF)



Schleife REPEAT - UNTIL

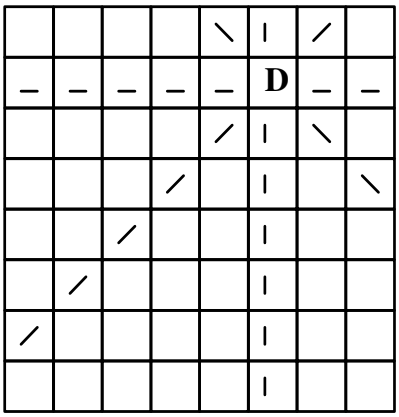


Schleife WHILE - DO

Abb. 2.2.5.a:
Struktogramm-Zeichen

Beispiel für die Verwendung von Struktogrammen

Das 8-Damenproblem: Acht (n) Damen sind auf einem 8x8 (nxn) Schachbrett so zu po-sitionieren, daß sie sich nicht gegenseitig bedrohen. Eine Bedrohung tritt gemäß den Re-geln des Schachspiels dann ein, wenn andere Damen in der gleichen Spalte oder Zeile oder in den gleichen schrägen Linien stehen wie die betrachtete Dame.



Eine Dame mit den von ihr bedrohten Feldern

Abb.2.2.5.b: 8-Damenproblem

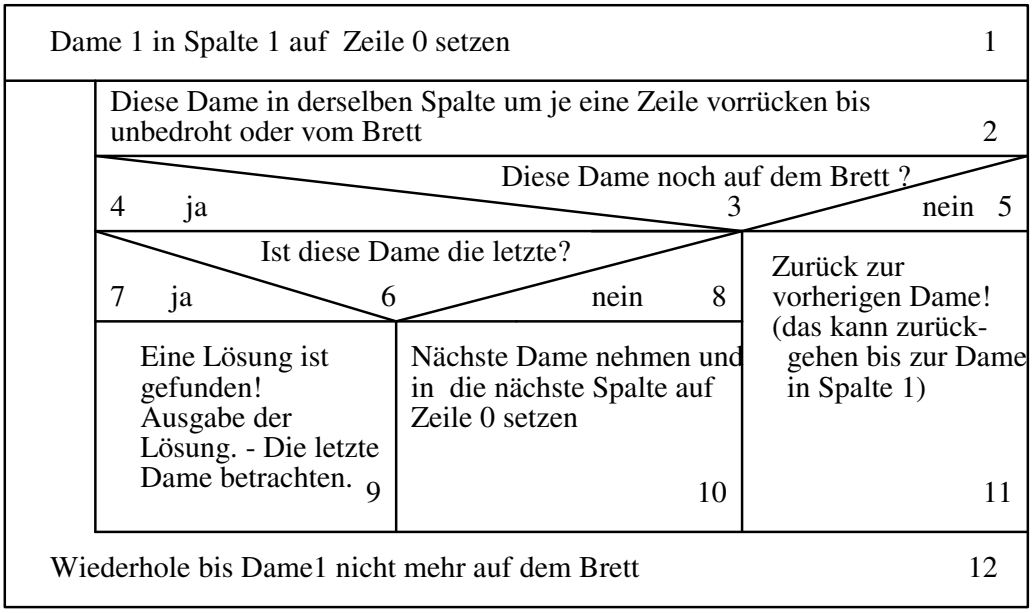
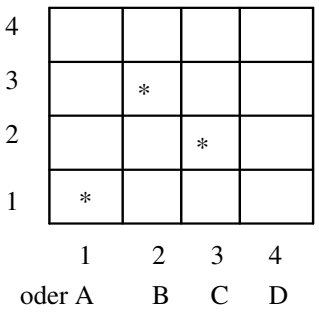


Abb.2.2.5.c: Struktogramm zum 8-Damen-Problem (die Zahlen dienen einer eventuellen verbalen Beschreibung des Algorithmus)

Die im Struktogramm von Abb.2.2.5.b angebotene Lösung beruht auf dem Backtrak-king-Verfahren, bei dem die Damen nacheinander von links nach rechts in die einzelnen Spalten gesetzt und bei Bedrohung durch andere Damen um eine Zeile weiter nach oben gesetzt werden. Geht das für die aktuell betrachtete Dame nicht, so wird sie vom Brett entfernt, und es geht bei der vorigen Dame weiter, indem diese eine Zeile nach oben gesetzt wird.

Man erkennt, daß die Texte in den einzelnen Strukturblöcken des Struktogramms Inhalte unterschiedlichen Umfangs beschreiben.
Die Frage in Block 7: "Ist die Dame die letzte?" läßt sich in einer einzigen Zeile programmieren (IF damenindex=n THEN ...), dagegen bedarf der Block 3 einer genaueren Betrachtung - ein weiteres Struktogramm (nun für dieses Teilproblem) entsteht. Aus ihm resultiert später die längere Prozedur "Vorruecken_und_pruefen".



Aufgaben

Das Spielbrett zeigt eine mögliche Spielsituation!
An welcher Stelle im Struktogramm befindet man sich?

In welchem Struktogrammteil steckt am meisten Programmierarbeit?

Was passiert im Struktogramm, wenn man eine Lösung gefunden hat (z.B. die erste Lösung für Dame 1 auf A2)?
Wird nach einer zweiten Lösung für Dame 1 auf A2 gesucht?
Ja, denn man setzt in Block 3 fort.

Programmablaufpläne

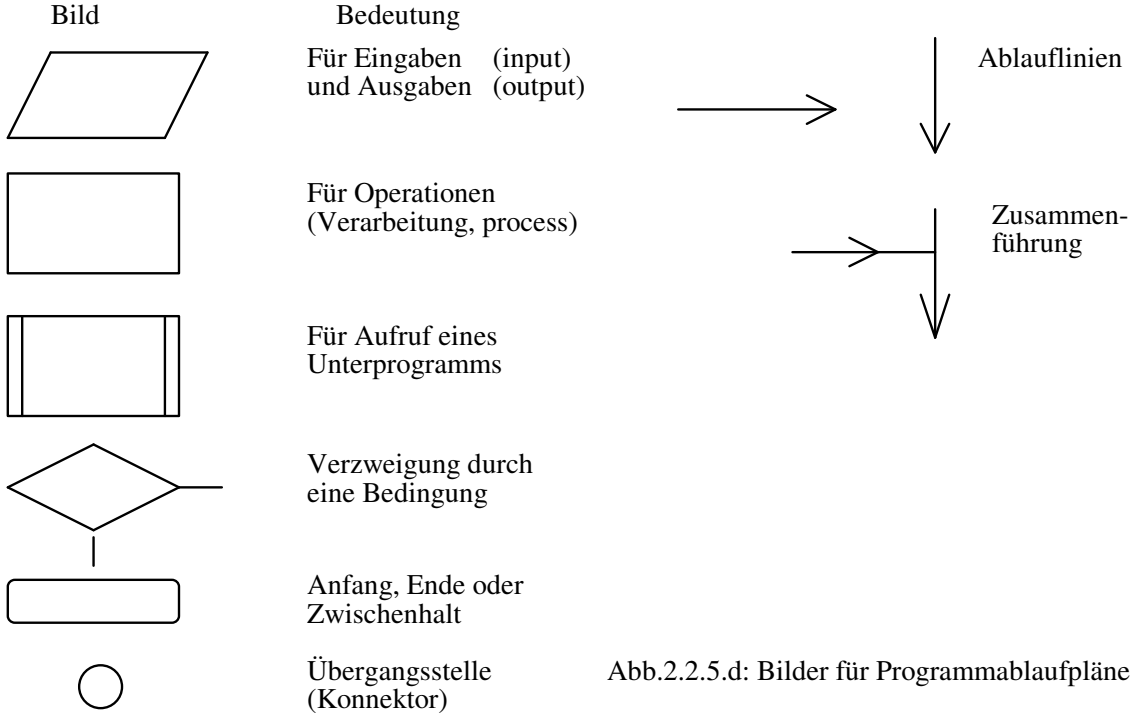


Abb.2.2.5.d: Bilder für Programmablaufpläne

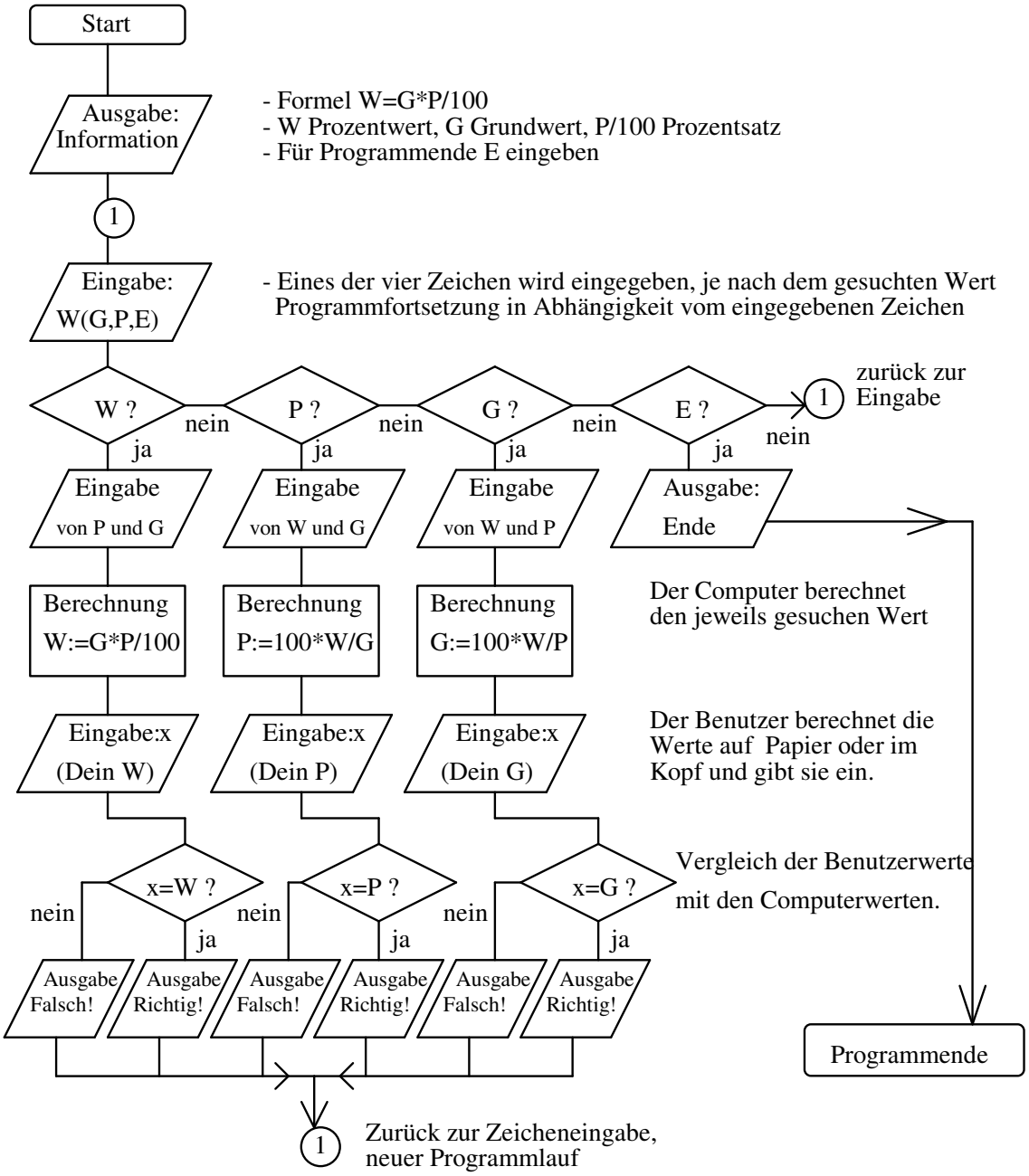


Abb.2.2.5.e: Programmablaufplan für die Konstruktion eines Trainingsprogramms zur Prozentrechnung

Gelegentlich sind auch die sogenannten Flußdiagramme **(Programmablaufpläne)** zur Beschreibung von Algorithmen geeignet, insbesondere bei maschinennaher

Programmierung und bei wenig umfangreichen Algorithmen, gerade auch aus der Mathematik. Allerdings führt die Programmiersprachenübersetzung solcher Ablaufpläne leicht zu unübersichtlichen (schlecht strukturierten) Programmen und dabei auch zur unkontrollierten Verwendung von Sprüngen. Die wichtigsten Bilder für Programmablaufpläne sind oben genannt. Das Beispiel zeigt, wie man die einzelnen Bilder durch Ablauflinien zusammenführt.

Baumdiagramme

Immer wenn es um die Untersuchung und Darstellung verschiedener Fälle geht, sind Baumdiagramme von großer Bedeutung. Zum Beispiel läßt sich für das obige Damenproblem ein Baumdiagramm mit den möglichen Damenpositionen anlegen, in dem man den Backtracking-Algorithmus gut verfolgen kann - im Beispiel beginnend mit der Position A2 von Dame 1 (für das 4-Damenproblem). Man erkennt, daß es für jede Damen-Startposition (A, Zeile) jeweils 64 grundsätzliche Möglichkeiten für die Damenstellungen gibt, insgesamt also 256 Möglichkeiten. Davon scheiden viele aus, der Backtracking-Algorithmus erkennt das sehr schnell:

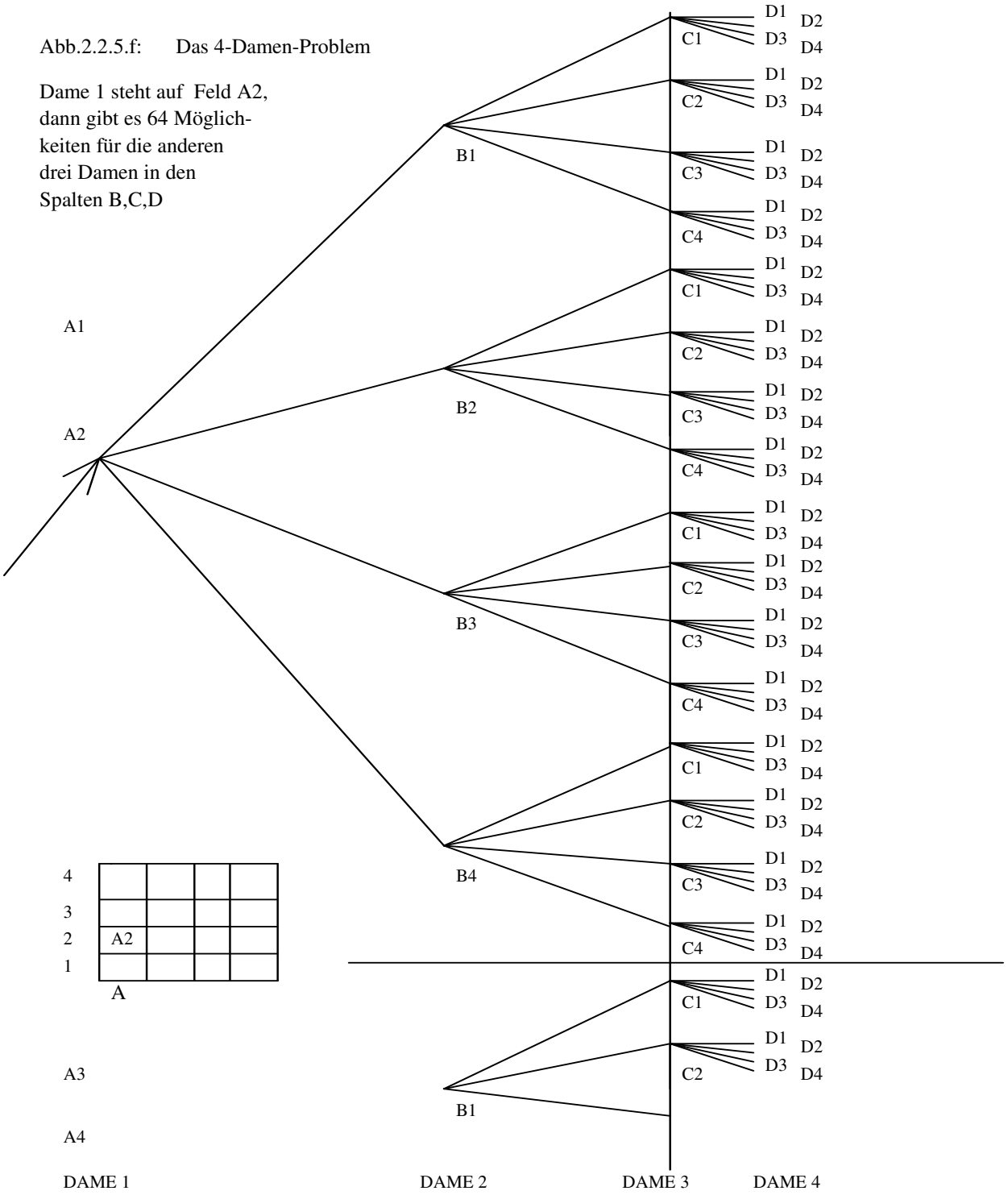
```
A2--> B1 (geht nicht),
A2--> B2 (geht nicht),
A2--> B3 (geht nicht);
A2--> B4 (geht),
A2--> B4--> C1  (geht),
A2--> B4--> C1--> D1 (geht nicht),
A2--> B4--> C1--> D2 (geht nicht),
A2--> B4--> C1--> D3 (geht, damit ist eine Lösung gefungen)
```

Die hier genannten Wege lassen sich im Baumdiagramm gut verfolgen - vorwärts und auch rückwärts. Man erkennt dabei, wie gewisse Äste des Baumes wegfallen bzw. nicht mehr untersucht werden müssen.

Die Bedeutung von Baumdiagrammen zeigt sich auch in ihrer Verwendung als Hierarchiebäume, Wahrscheinlichkeitsbäume, Suchbäume und weiteren Problemen.

Abb.2.2.5.f: Das 4-Damen-Problem

Dame 1 steht auf Feld A2, dann gibt es 64 Möglichkeiten für die anderen drei Damen in den Spalten B,C,D



SADT-Methode (structured analysis and design technique)

Die SADT-Methode dient zur Analyse und graphischen Darstellung beim Entwurf neuer Systeme oder bei der Untersuchung bestehender Systeme. Sie dient nicht zur Darstellung von Algorithmen, da man mit ihr z.B. keine Verzweigungen darstellen kann. Hierfür benutzt man z.B. Struktogramme, s.o.

Innerhalb des Entwicklungsprozesses von Software ist die SADT-Methode insbesondere bei der Problemanalyse und in der Entwurfsphase nützlich. Man unterscheidet zwei graphische Grundelemente, die den gleichen formalen Aufbau haben:

a) für Aktivitäten (**funktionale Sicht** eines Systems) und
b) für Daten und Datenstrukturen (**daten(objekt-)bezogene Sicht** eines Systems).

Die **Eingabedaten** werden durch **Aktivitäten** in **Ausgabedaten** transformiert. Entsprechend bewirken Eingabeaktivitäten die Umwandlung von Daten und führen zu Ausgabeaktivitäten.
Die Vorgänge werden durch **Steuerungsdaten** und **Steuerungsaktivitäten** sowie durch **Ausführungsmechanismen** ermöglicht. In einem Diagramm können mehrere Kästchen (Rechtecke) von links oben nach rechts unten dargestellt werden. Dabei sollten in einem Diagramm nicht mehr als sechs Rechtecke vorkommen, um nicht die Übersicht zu gefährden.

Entsprechend dem Konzept der strukturierten Zerlegung beginnt man bei der Systembeschreibung auf der höchsten Abstraktionsebene und verfeinert dann in Teilsysteme, so daß eine hierarchische Gliederung des Systems entsteht. Diese wird dann in verschiedenen, aufeinander aufbauenden SADT-Diagrammen beschrieben. Innerhalb einer Verfeinerungsstufe werden die Teilsystems von links oben nach rechts unten treppenartig angelegt - womit jedoch keine zeitliche Reihenfolge in der Abarbeitung gemeint sein muß.

Für eine Weinhandlung könnte die höchste Abstraktionsebene in funktionaler Sichtweise folgendermaßen aussehen:

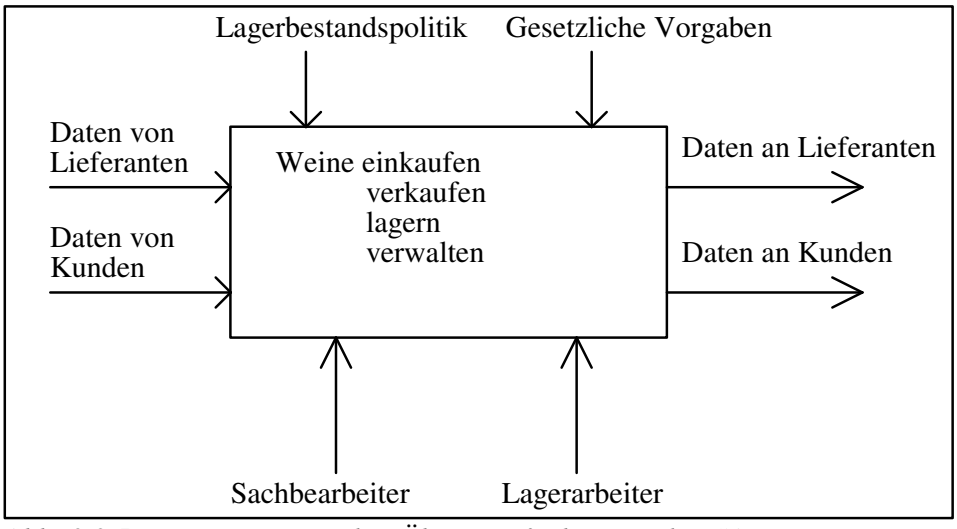


Abb. 2.2.5.g Hinweis: Bei den Übungsaufgaben werden SADT-Diagramme vorgelegt, die die Rechnungsstellung beim (Wein)Verkauf an einen Kunden - beschreiben.

Die Vorgehensweise bei der Konstruktion von SADT-Diagrammen kann man so beschreiben:
1. Aufstellen des Aktivitätenmodells
    1.1 Erfassen der Funktionen
    1.2 Festlegung der Aktivitäten und der Daten
    1.3 Hierarchische Gliederung der Aktivitäten mit bis zu 6 Aktivitäten je Verfeinerung
    1.4 Definition der Schnittstellen zur Umgebung (Ein- und Ausgabedaten, Steuerungspfeile und Pfeile für die Mechanismen
    1.5 Wiederholung der Schritte 1.3 und 1.4 bis eine ausreichende Verfeinerung erreicht ist.
    1.6 Überprüfung durch den Anwender, ggf. Nachbesserung durch den Konstrukteur
2. Aufstellen des Datenmodells
    2.1 Hierarchische Gliederung der Daten mit bis zu 6 Datenteile je Verfeinerung.
    2.2 Definition der Schnittstellen (Eingangs- und Ausgangsaktivitäten, Steuerungspfeile und Pfeile für die Mechanismen.
    2.3 Wiederholung der Schritte 2.1 und 2.2 bis eine ausreichende Verfeinerung erreicht ist.
    2.4 Überprüfung durch den Anwender, ggf. Nachbesserung durch den Konstrukteur.
3. Vergleich von Datenmodell und Aktivitätenmodell auf Stimmigkeit
4. Notieren zusätzlicher Informationen.

SADT-Diagramme haben also die im folgenden allgemein formulierte Form:

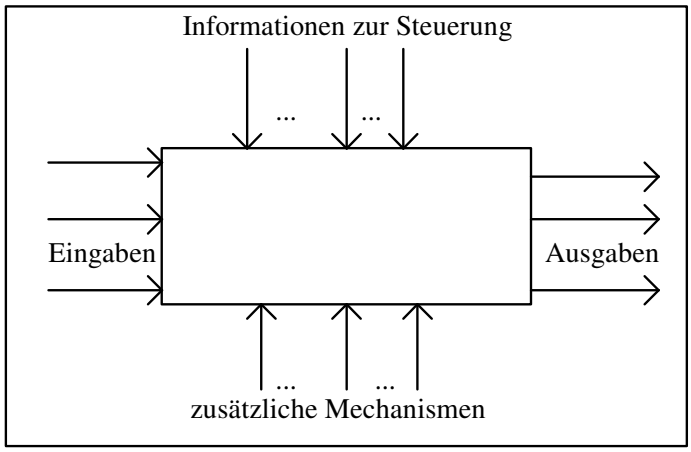


Abb.2.2.5.h

Aktivitäten-Diagramm

Funktionale Sicht auf ein System

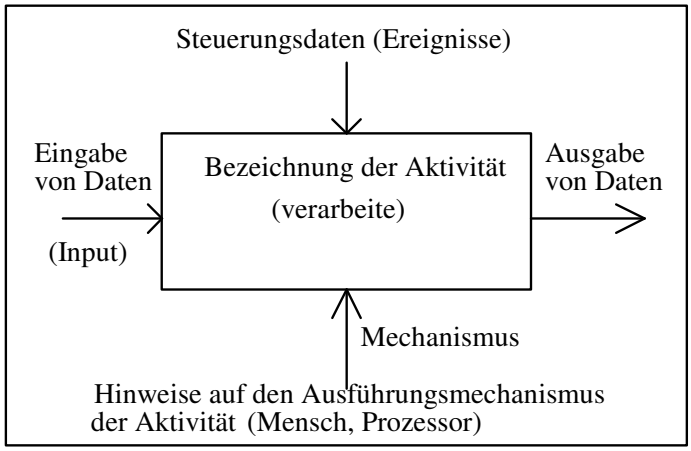


Abb. 2.2.5.i

Datendiagramm

Datenbezogene Sicht auf ein System

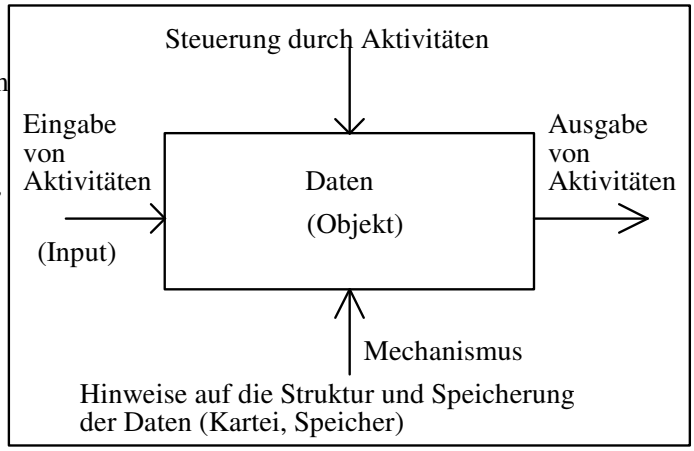


Abb.2.2.5.j

HIPO-Methode

Die HYPO-Methode (Hierarchie plus Input, Process, Output) ist eine von der Firma IBM eingeführte Methode für den Entwurf und die Dokumentation von Datenverarbeitungsprojekten. Die wichtigsten Elemente dieser Technik bestehen in einer hierarchischen Gliederung des Systems und seiner Funktionen mit Hilfe von Übersichts- und Detaildiagrammen.

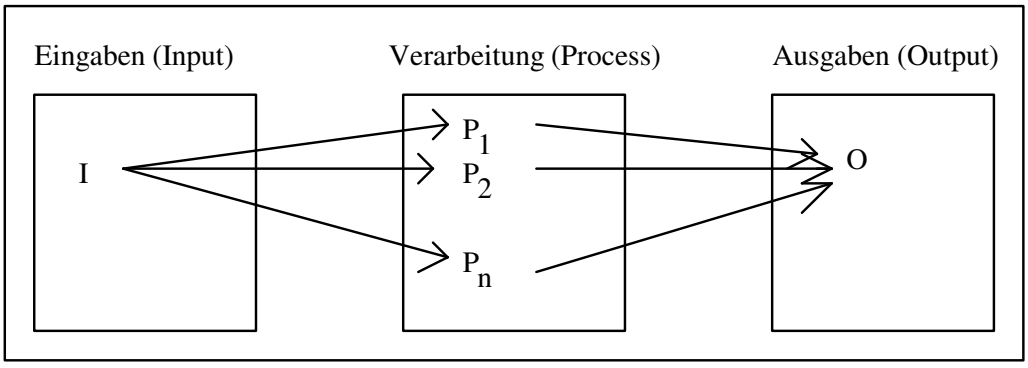
Die Überlegungen gehen davon aus, daß jedes Programm und jedes Programmodul nach dem EVA-Prinzip arbeitet: Eingabe - Verarbeitung - Ausgabe (Input - Process - Output). Es liegt also eine Abbildung V von der Menge der Eingabedaten I in die Menge der Ausgabedaten O vor: $P : I \longrightarrow O$.

Die Vorgehensweise bei der Konstruktion von HIPO-Diagrammen kann man so beschreiben:

1. Festlegung der Ausgabedaten des Gesamtprogramms
2. Festlegung der dazu notwendigen Eingaben
3. (Umgangssprachliche) Beschreibung der Abbildung (Funktion) $P : I \longrightarrow O$
4. Zerlegung der Abbildung P in Teilabbildungen $P_1, P_2, ... P_n$ und deren umgangssprachliche Beschreibung
5. Festlegung der Ausgabe- und Eingabedaten Teilabbildungen
6. Gegebenenfalls erneute der Zerlegung in Teilabbildungen, bis die Teilabbildungen so elementar werden, daß man sie leicht in eine Programmiersprache übertragen kann.

Der grundlegende Aufbau eines HIPO-Diagramms ist also:

P (Name des Prozesses)



Die Pfeile geben an, welche Eingaben zu den den angezeigten Verarbeitungsvorgängen führen bzw. welche Verarbeitungsprozesse bestimmte Ausgaben erzeugen.
Abb. 2.2.5.k

Die Art der Darstellung wird im folgenden an einem Programmsystem zur Auswertung von Multiple-Choice-Tests erläutert. MC-Tests werden häufig bei Bewerbungsvorgängen eingesetzt. Zu jeder Frage eines längeren Fragebogens werden verschiedene Antworten vorgeschlagen. Die zu testende Person muß eine Antwort ankreuzen.

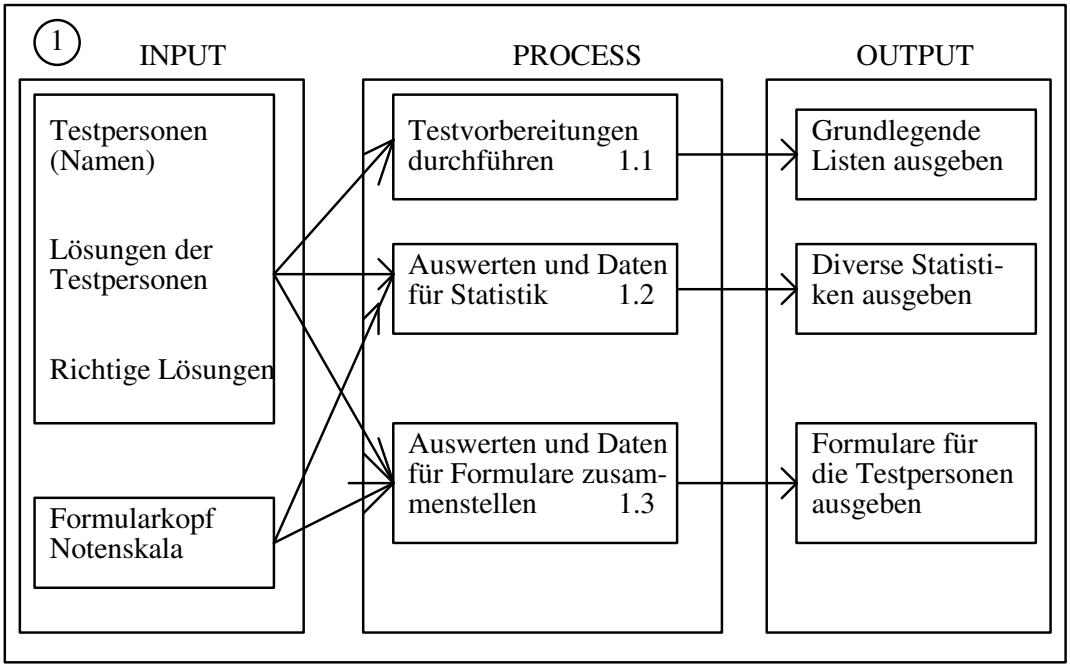


Abb. 2.2.5.l: Darstellung von "Multiple Choice" als Modulhierarchie in einem HIPO-Diagramm

Anschließend wären die Prozesse 1.1 bis 1.3 durch weitere Diagramme zu beschreiben.

Hinweis: Eine detaillierte Projektbeschreibung für das Multiple-Choice-System findet man in [Lehmann, E.: Projektarbeit im Informatikunterricht, Teubner-Verlag 1985]; das zugehörige Programm ist in Turbo-Pascal geschrieben und kann beim Autor erworben werden.

2.2.6 Oberflächendesign - Masken

Die Bedienoberfläche eines Programms ist die wichtigste Schnittstelle zwischen dem Computer und dem Benutzer der Software. Beim Entwurf von Software müssen viel Mühe und gestalterische Leistungen (Design-Qualitäten) aufgewandt werden, um ein benutzerfreundliches Erscheinungsbild der Bedienoberfläche herzustellen. Es handelt sich damit um ein Teilgebiet der **Software-Ergonomie**. Beim Entwurf von Bedienoberflächen müssen z.B. folgende Fragen beantworten werden:

- Wie sollen die Informationen auf dem Bildschirm gruppiert und angeordnet werden?
- Wie qualifiziert sind die Benutzer (sie sind in der Regel keine homogene Gruppe)?
- Welche Tätigkeiten sollen den Anwendern abgenommen werden, ohne ihnen die Kompetenz zu nehmen?

Von einer Bedienoberfläche verlangen wir Funktionalität und Attraktivität. Diese Eigenschaften spiegeln sich wider in einem funktionellen Design.

Wir unterscheiden hier zwei Arten von Bedienoberflächen (Benutzungsoberflächen):
a) **Die Festlegung der Programmfortsetzung durch Auswahl eines "Menüpunktes"** (einer Option) aus einer vorgegebenen Menge von Fortsetzungsmöglichkeiten, also aus einem "Menü". Die wählbaren Optionen sind i.a. in Menübildern angeordnet, aus denen der Benutzer auswählen kann. Dem Benutzer begegnen in den verschiedenen Programmen u.a. folgende Bedienmöglichkeiten zur Auswahl einer Option:

- Eingabe eines der Option zugeordneten Zeichens oder einer Zahl
- "Anfahren" der gewünschten Option mit Cursortasten
- Anklicken mit einer Maustaste

Es folgt ein Beispiel für eine Benutzeroberfläche:

```
T-MENUE1
THEATER  -  PLATZBUCHUNG  -  VERWALTUNG
1: Informationen über Veranstaltungen
2: Platzbuchung
3: Theater-Verwaltung
4: Informationen über das Programmsystem
5: Hilfe
6: Ende
```

```
Auswahl der Menüpunkte mit Cursortasten
```

b) **Für die Eingabe von Datensätzen an Bildschirmgeräten benutzt man Bildschirm-masken**. Der Benutzer findet auf dem Bildschirm eine Maske (ein vorgefertigtes Formu-lar) vor und kann seine Daten nun nur noch an den vorher bestimmten Stellen eingeben. Diese müssen außerdem noch aus dem zulässigen Definitionsbereich stammen, da die Eingabepositionen entsprechend abgesichert sind. Mit dieser Technik können Eingabefehler weitgehend vermieden werden, so daß auch ein Laie mit dem Programm arbeiten kann.

Beispiel:

```
EINGABE EINES PISTEN-DATENSATZES                        Satznummer:  8
                                                        abc     .ski
         Pistennummer : 0    (# beendet die Satzeingabe)
         Pistenname   : ....................................

         Schwierigkeit (b   blau, r rot    , s schwarz):  .
         Pistenbelag   (n normal, v vereist, s steinig):  .
         Pistenbetrieb (o  offen, g geschlossen       ):  .

LIFTE unten an  unten ab   oben an  oben ab    PISTEN  unten ab  oben an
           ...       ...       ...      ...                ...      ...
           ...       ...       ...      ...                ...      ...
           ...       ...       ...      ...                ...      ...
           ...       ...       ...      ...                ...      ...
           ...       ...       ...      ...                ...      ...

HÜTTEN   unten                 oben                  HILFE: Am Anfang
         ..................    ..................    einer Eingabepo-
         ..................    ..................    sition ? drücken
         ..................    ..................

Eingaben ok (j,n) ?                                Satz speichern(j/n) ?
```

Eingaben sind nur an den mit .... markierten Stellen möglich, die noch dazu jeweils farbig angezeigt werden.

Maskengeneratoren

Da der Entwurf von Bedienoberflächen i.a. zeitraubend ist, arbeiten die Softwareentwickler häufig mit Hilfsprogrammen, z.B. mit Maskengeneratoren. Diese sind jedoch für den Anfänger und den Schulunterricht häufig zu komplex und erfordern lange Einarbeitungszeit, so daß man sie in der Schule erst im fortgeschrittenen Informatikunterricht einsetzen kann, was allerdings zur Zeit leider noch weitgehend unterbleibt.
In diesem Buch werden daher die leicht zu bedienenden Hilfsprogramme MENUE.EXE und BILD_DEF.EXE benutzt, die in dem Buch "Programmieren in Turbo-Pascal mit Bausteinen", Dümmler-Verlag 1993, ausführlich beschrieben werden.

Hilfsmittel zur Gestaltung der Bedienoberfläche unterstützen von Anfang an benutzerorientierte Entwürfe. Sie fördern damit das Denken in Bauteilen (Modulen), die z.B. aus geeigneten Problemzerlegungen entstehen.

Sie ermöglichen damit auch zumindest ansatzweise ein Arbeiten im Sinne eines Prototyping, siehe Kapitel 2.1.2.

- MENUE hilft bei der Erstellung von Menüs, bei denen man Menüpunkte mit Hilfe der Cursortasten (auf/ab) oder mit der Maus auswählen kann. Durch mehrfache Anwendung lassen sich damit auf einfache Weise auch Pop-down-Menüs realisieren. Der Aufruf des Menüs in einem Pascal-Programm erfolgt durch Aufruf der Prozedur MENUE_WAHL(... Parameter...).

- BILD_DEF ermöglicht dem Anwender das "Malen" und Schreiben auf der gesamten Fläche des Bildschirms. Dieser wird dann als Ganzes gespeichert und ist aus Programmen heraus wieder leicht aufrufbar. Dieses elementare Hilfsmittel ist für viele Anwendungen sehr nützlich. Insbesondere kann man sich damit leicht Masken erstellen.

Das Hilfsprogramm MENUE

Hinweis: MENUE finden Sie auf der Diskette LEH-TOOLS, siehe Kapitel 7.

Der Vorgang zur Menüerstellung wird durch Informationen und Auswahl aus Menüpunkten gesteuert und ist leicht verständlich.

Die Vorgehensweise:

(1) Nach dem Starten des Programms MENUE.EXE erhält man die Information:

Mit diesem Programm kann man Menüs konstruieren, ansehen und ändern, in denen man mit dem Cursor wandern und mit der Enter-Taste einen Menüpunkt auswählen kann.

(2) Man erhält nun das Auswahl-Menü

```
      ERSTELLUNG - INSPEKTION - ÄNDERUNG VON MENÜS

 1: Erstellen eines Balkenmenüs zu einem anderen Programm, Speicherung
 2: Ansehen eines vorhandenen Balkenmenues, Balkenpositionen ausgeben
 3: Ändern  eines vorhandenen Balkenmenüs, Speicherung
 4: Lesen der Hilfe / Dokumentation des vorliegenden Programms
 5: Programmende

Auswahl einer  Option mit den Cursortasten AUF/AB
```

Übernehmen der Option mit der Taste RETURN

(3) Mit der Cursortaste wählt man z.B. Menüpunkt 1 und kann nun **den Anweisungen folgen** um die Textpositionen für das Menü und damit die Menüpunkte zu definieren und die Texte zu formulieren.

(4) Das ganze **Menü kann** nun unter einem frei wählbaren Namen (z.B. "test1.bal") **in einer Datei abgelegt werden**.

(5) Die **Einbindung des Menüs in ein Programm** erfolgt durch den Aufruf der Prozedur MENUE_WAHL aus der Unit BALK_U z.B. so:

```
Uses Balk_u;
     ...
     dateiname:='test1.bal';
     menue_wahl(green, yellow, dateiname, option);
     ...
```

Das Hilfsprogramm BILD_DEF

Hinweis: BILD_DEF finden Sie auf der Diskette LEH-TOOLS , siehe Kapitel 7.

Mit dem Programm BILD_DEF.EXE können Sie auf einfache Weise auf dem Bildschirm in der Art einer Blockgrafik "malen". Das Bild wird abgespeichert und kann durch Aufruf der zugehörigen Prozedur aus der Unit BILD_U in Pascal-Programme übernommen werden.

Die Vorgehensweise:

(1) Das Programm mit wird BILD_DEF.EXE gestartet.

```
            B I L D S C H I R M   B E A R B E I T E N

(1) Informationen zum Programm
(2) Neues Bild erstellen, speichern
(3) Gespeichertes Bild ansehen
(4) Gespeichertes Bild ändern
(5) Bildattribute ändern
(6) Programmende
                              CURSOR AUF/AB WÄHLT DIE OPTION AUS
```

(2) Es wurde mit den Cursortasten Option 2 gewählt: Der Name der Datei wird eingegeben.
In welcher Datei wollen Sie das Bild speichern? Name= bildtest.mas

(3) Man erhält eine Information:

Sie können nun den Bildschirm-Inhalt durch "Malen" mit Hilfe der Tasten oder Alt-Codes und Cursorbewegungen erstellen. Der Zeichenbereich: Spalten [1..80], Zeilen [1..24] - Zeichnung beenden mit der F1-Taste!

(4) Der Bildschirm kann nun wie ein Editor behandelt werden. Das Bild kann "gemalt" werden, z.B. auch mit Sonderzeichen, die über den ASCII-Code eingegeben werden. Am unteren Rand laufen die Spalten- und Zeilenpositionen mit.

(5) Mit der F1-Taste wird die Zeichnung beendet und in der oben angegebenen Datei gespeichert. Man gelangt zurück ins Hauptmenü, siehe (1).

Der Aufruf der Maske erfolgt z.B. so:

```
USES BILD_U;
  ...
  dateiname:='testbild.mas';
  textbild_aus(dateiname);  {Ausgeben der Maske testbild.mas}
  ...
```

Anwendungsmöglichkeiten von BILD_DEF

(1) Man erstellt sich ein Bild als Vorspann eines Programms.

(2) Man schreibt einen Informationstext auf den Bildschirm, der später von einem Programm an geeigneter Stelle aufgerufen wird.

(3) Mehrere solcher Bilder mit Texten oder "Grafik" können hintereinander geschaltet werden und damit einen "Film" bilden. Hierzu kann die Prozedur SHOW aus der Unit BILD_U benutzt werden: *SHOW(dateiname,zeit).*

(4) Aus einem Anwenderprogramm heraus kann man auf ein Bild zusätzlichen Text schreiben oder Teile löschen.

(5) Man erstellt direkt auf dem Bildschirm eine Bedienoberfläche. Mit Hilfe von Prozeduren aus der Unit BALK_U kann man Positionen in der Bedienoberfläche ansteuern und die Optionen mit der Maustaste anklicken oder auf andere Art auswählen..

Beispiel einer mit BILD-DEF erstellten Maske:

```
                    W E T T S C H E I N
  Datum   :
-----------------------------------
  Rennen  : 1 2 3 4 5 6 7 8 9 10
-----------------------------------
  Wettart : S     P     KE    GE
-----------------------------------
```

1.Platz : 1 2 3 4 5 6 7 8 2.Platz : 1 2 3 4 5 6 7 8 3.Platz : 1 2 3 4 5 6 7 8 --- DM : 2.50 5.00 7.50 10.00 20.00 50.00 100.00 200.00 500.00	

2.2.7 Baustein-Bibliotheken

Beim Entwurf von Software und bei ihrer programmiersprachlichen Realisierung treten häufig ähnliche oder gleiche Probleme auf wie bei einem anderen schon früher entworfenen Softwareprodukt. Warum soll man sich nicht dieser Kenntnisse bedienen? Wie das geschehen kann, zeigt Abbildung 2.2.7.a.

Das Hauptproblem dabei ist, die wiederverwendbaren Komponenten zu erkennen, verständlich zu dokumentieren und so abzulegen, daß man sie wiederfindet!

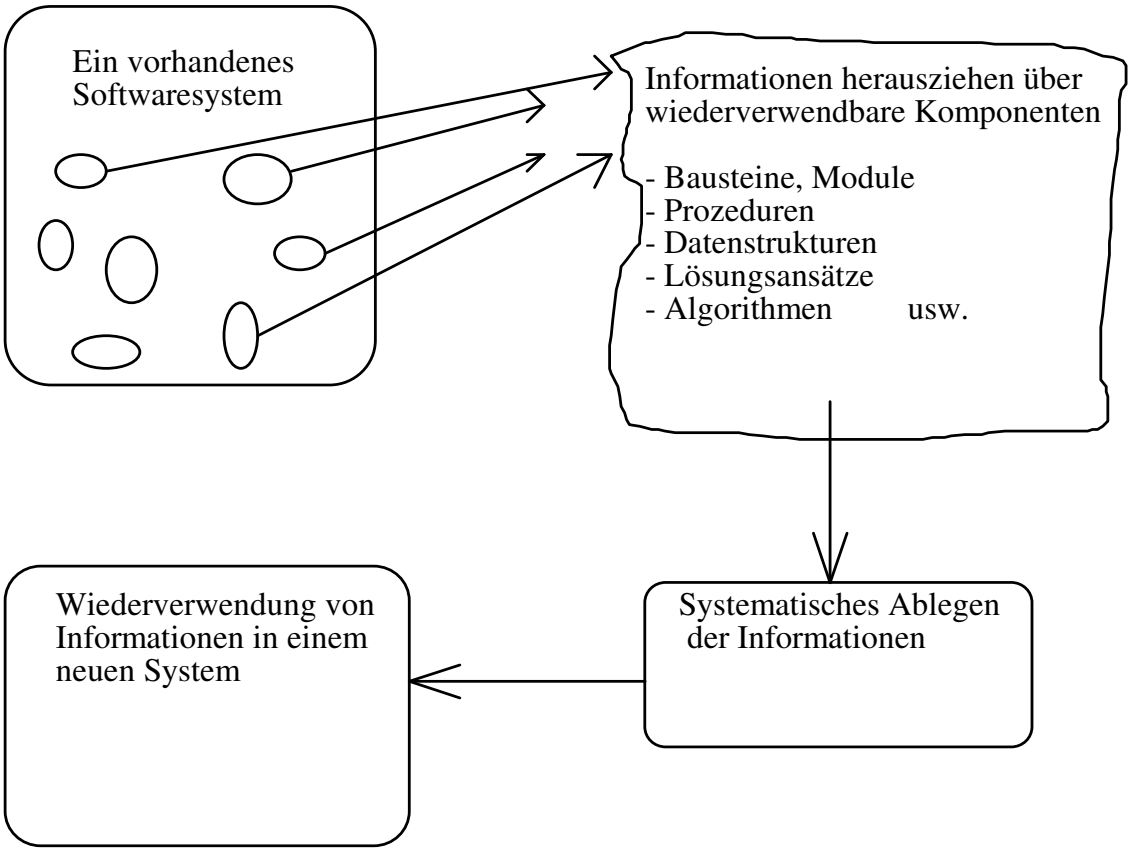


Abb. 2.2.7.a: Wiederverwendung

Allerdings gibt es auch gewisse Standardaufgaben, von denen man weiß, daß sie häufig wiederkehren, z.B. :

- Eingabe einer Zeichenkette

- Eingabe einer ganzen Zahl
- Ausgabe eines Hilfetextes usw.

Für derartige Fälle sollten Bausteine zur Verfügung stehen, die dann immer wieder verwendet werden können und deren innerer Aufbau für den Verwender uninteressant ist. Er muß lediglich die Funktionen dieser Bausteine kennen.

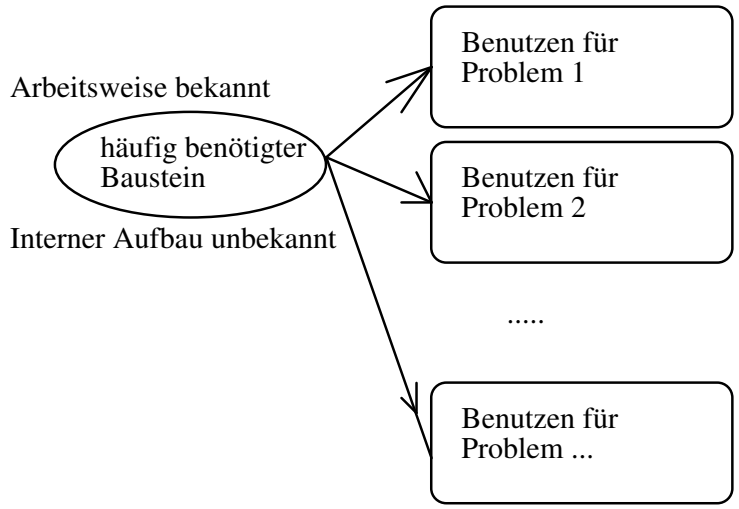


Abb. 2.2.7.b: Bausteine benutzen

Die Verwendung vorgefertiger Bausteine trägt wesentlich zu einer rationellen Softwareerstellung bei, sie spart viel Zeit.

Deshalb wird dringend empfohlen, solche Bausteinbibliotheken zu benutzen. Man programmiert mit ihnen auf einer höheren Ebene, spart also viel Detailarbeit und damit auch viel Zeit. Damit kann man sich auch an Projekte heranwagen, die komplexer und zeitaufwendiger sein können als bei Bearbeitung ohne Hilfsmittel.

Beispielsweise ermöglicht die Verwendung des Prozeduraufrufs ***Input_string*** in Zusammenhang mit zwei FOR-Schleifen die maskenartige Eingabe mehrerer Zeichenketten an Positionen, die durch die Spalten 10x und die Zeilen 2y angegeben sind..

```
FOR x:=1 TO 6 DO
    FOR y:=1 TO 8 DO
     Input_string(10*x,2*y,'',str,6,['A'..'Z']);
```

Die Diskette LEH-TOOLS stellt zahlreiche Bausteine zur Verfügung, von denen hier einige angegeben werden. Eine genaue Beschreibung dieser Bausteine mit zahlreichen Anwendungen und Übungsaufgaben finden Sie in dem Dümmler-Buch: "Programmieren in Turbo-Pascal mit Bausteinen", 1994 (E.Lehmann). Die Bausteine sind in verschiedenen Units zusammengefaßt. Beispielsweise enthält die Unit INO_U die folgenden vielfach verwendbaren Prozeduren:

LISTE DER DATENTYPEN UND PROZEDUREN / INTERFACE von INO_U

```
USES Crt;
TYPE zeichenmenge,read_cset = SET OF CHAR;

(1) FUNCTION  Holetaste: INTEGER;

(2) PROCEDURE Lauftext  (in_zeile,von_spalte,bis_spalte:INTEGER; text:STRING;
        tempo:INTEGER);

(3)PROCEDURE Input_textdatei (var datei_name:STRING;
                name_erfragen:BOOLEAN);

(4) PROCEDURE Output_textdatei
        (x1,y1,x2,y2:INTEGER;Dateiname:STRING40;SuchWort:string40);

(5) PROCEDURE Input_integer (x,y: INTEGER; input_aufforderung:STRING;
                VAR zahl:INTEGER; zahllaenge: INTEGER);

(6) PROCEDURE Input_real  (x,y: INTEGER; input_aufforderung:STRING;
                VAR zahl:real; zahllaenge:INTEGER);

(7) PROCEDURE Input_zeichen(x,y:INTEGER; input_aufforderung:STRING;
                VAR antwort:CHAR;  erlaubte_zeichen: zeichenmenge);

(8) PROCEDURE Input_string (x,y: INTEGER; input_aufforderung : STRING;
        VAR input: STRING; stringlaenge  : INTEGER; erlaubt: Zeichenmenge);

(9) PROCEDURE Read_i  (VAR i:INTEGER);

(10) PROCEDURE Read_r   (VAR r:REAL);

(11) FUNCTION Eoln_n: BOOLEAN;
```

Beispiel einer Funktions-Spezifikation

FUNCTION Holetaste: INTEGER;

Diese Funktion holt eine Taste (Eingabe eines Zeichens c) und wandelt das Zeichen in den entsprechenden Integer-Wert des ASCII-Codes um. Dieser wird dann weitergegeben, z.B. zwecks Abfragen. Handelt es sich dabei um eine Sondertaste (falls

c<>#0), wird zum ASCII-Code (ohne Sondertaste) die Zahl 256 addiert. So kann z.B. abgefragt wer-den, um welche Tasten es sich handelt.

Programmier-Beispiel:

```
PROGRAM x;
USES Ino_u;
VAR bewegung: INTEGER;
BEGIN
  ...
  bewegung:=Holetaste;
  IF (bewegung=27) {escape} THEN
  BEGIN
    WRITE('Programm-Abbruch');
    EXIT;
  END;
  ...
END.
```

Ein Baustein für Hilfetexte und Informationen - *Output-textdatei(...)*

Hilfen werden vom Benutzer an verschiedenen Stellen eines Programmsystems erwartet. Diese Hilfen können mehrere Funktionen erfüllen:

- Hilfen für die Bedienung des Programmsystems
- Aktuelle Hilfen zur Programmfortsetzung
- Hilfen bei Fehlermeldungen
- Hilfen zur Auswertung von Ergebnissen
- Informationen über aufrufbare Menüpunkte
- Beschreibung der Leistungen von Programmteilen
- Sonstige Informationen

Besonders vorteilhaft ist es, wenn man eine **"kontextsensitive Hilfe"** zur Verfügung stellen kann. Eine derartige Hilfe unterstützt den Benutzer bei dem gerade auftretenden Problem. Sie ist also nicht umfassend, sondern bezieht sich auf die jeweilige vorliegende Situation - den jeweiligen Zusammenhang, auf den "Kontext". Zur Realisierung solcher Anforderungen sind Prozeduren wie die im folgenden beschriebene Prozedur außerordentlich nützlich.

PROCEDURE Output_textdatei
(x1,y1,x2,y2:INTEGER;Dateiname:string40;SuchWort:string40);

Die Prozedur zeigt eine Textdatei (ASCII-Format) in einem Fenster auf dem Bildschirm mit Blättermöglichkeit an.

Besonderheiten:

- Die Datei wird auch gefunden, wenn sie nicht im aktuellen Verzeichnis steht, sondern in dem, in dem die EXE-Datei des Programms steht.

- Die Anzeige kann in einem Fenster mit den Spalten- und Zeilen-Positionen (x1,y1,x2,y2) erfolgen. Allerdings muß der Text in diesem Spalten-, Zeilenbereich auch vorher eingegeben worden sein.

- Der ursprüngliche Bildinhalt wird nach Rückkehr aus der Prozedur wiederhergestellt.

- Mit "SuchWort" kann ein Stichwort vorgegeben werden, auf das der Textausschnitt automatisch positioniert wird. Wird das Suchwort nicht gefunden oder ist keines angegeben (Leerstring), so wird der Dateianfang gezeigt.

- Im Text können Blättersperren vorgesehen werden, die insbesondere in Verbindung mit der Stichwortsuche nützlich sind (Darstellung von kontextsensitiven Hilfeseiten, welche in einer gemeinsamen Datei stehen): Zeilen im Text, welche mit der Zeichenfolge '{**}' beginnen, können nicht überblättert werden (und werden auch nicht dargestellt). Wird das Suchwort in einer solchen Zeile gefunden, so wird der auf die "Sperrzeile" folgende Text ausgegeben.

Parameterbedeutung:
x1,y1,x2,y2: Linke obere und rechte untere Ecke des Fensters,
dateiname : Name der auszugebenden Datei

Programmier-Beispiele:
a) *Output_textdatei(1,1,80,25,'datei1.dok','');*
Die Datei 'datei1.dok' wird auf dem gesamten Bildschirm vom ersten Zeichen an ausgege-ben.

b) *Output_textdatei(5,5,75,20,'datei1.dok','computer');*
Die Datei 'datei1.dok' wird im Fenster 5,5,75,20 vom Suchwort "computer" an ausgege-ben. Die Zeichen von datei1.dok müssen in dem Bereich 5,5,75,20 eingegeben worden sein.

c) Es sind mehrere zu lesende Dateien vorhanden:

```
...
dateiname:=´?´;
input_string(1,25,´Welche Datei wollen Sie lesen?´, dateiname, [ ´a´..´z´]);
Output_textdatei(1,1,80,25,'datei1.dok','');
...
```

Bekanntlich stellt das Turbo-Pascal-System auch selbst solche Bausteine zur Verfü-gung, z.B. in der Unit Crt (*Gotoxy(...), Clrscr*, usw.).

2.2.8 Qualitätsanforderungen an Software aus Benutzer- und Programmierersicht

Vorüberlegungen

Sie möchten ein Softwareprodukt für einen bestimmten Zweck kaufen (Katalogisieren Ihrer Schallplatten, Zeichnen räumlicher Bilder, Führung Ihrer Finanzen, Terminplanung oder Ähnliches). Was erwarten Sie von dem Programm, welche Eigenschaften soll es haben? Vermutlich werden Sie an Eigenschaften denken wie leichte Bedienbarkeit, Erzielung korrekter Ergebnisse usw. - und preiswert soll die Software auch noch sein.

Software wird benutzt bzw. angewendet (**Anwendersicht**), sie wird gewartet (**Operator- und Programmierersicht**) und die Arbeitsergebnisse werden ausgewertet (**Auswerter-sicht**).

Qualitätsanforderungen sind auch eine Frage der **(Software-)Ergonomie**, die sich mit der Anpassung von Arbeitsmitteln und Arbeitsumgebungen an die Leistungsfähigkeit des arbeitenden Menchen beschäftigt. Abbildung 2.2.8.a informiert über einige Zusammen-hänge.

Benutzerfreundlichkeit bezieht sich in erster Linie auf die Schnittstelle zwischen Programmsystem und Benutzer (Maschine,Mensch), also auf die Benutzeroberfläche. Diese umfaßt u.a.

- o Benutzersteuerung
- o Menüvorgaben
- o Kommandoeingaben
- o Maskeninhalte
- o Maskendesign
- o Hilfeangebote
- o Einheitlichkeit der Benutzerführung

Die **Einfachheit** eines Programmsystems äußert sich vornehmlich in der leicht verständlichen und einfach anwendbaren Dialogsprache, in den gut strukturierten und übersichtlich angebotenen Funktionen, in einer übersichtlichen Gestaltung der Bildschirmmasken und in der Einheitlichkeit von Steuermerkmalen und Tastenbelegung.

Dialog-Flexibilität bedeutet die Möglichkeit der jederzeitigen Unterbrechung eines Dialogs, eine benutzerorientierte Dialoggestaltung (z.B. in einer geeigneten Sprache) sowie die Ausgabe von Zwischen- und Endergebnisse.

Die **Zuverlässigkeit** zeigt sich u.a. darin, daß der Benutzer bei der Verwendung des Systems keine Überraschungen erlebt. daß sich Systemverhalten bei einer gleichartigen

Aufgabenstellung nicht ändert, die Antwortzeiten kurz sind und die Arbeitsausführung stabil ist. Begangene Fehler sollte man rückgängig machen können.

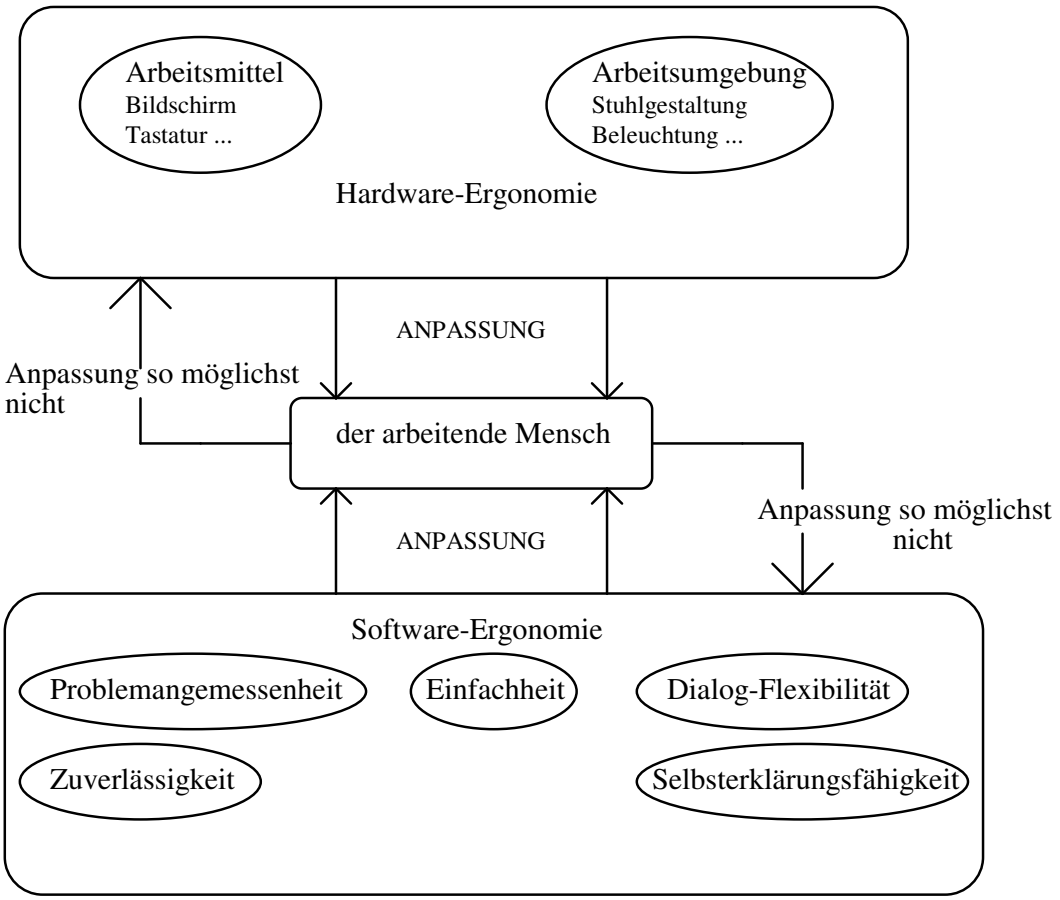


Abb. 2.2.8.a : Software-Ergonomie

Bei der Benutzung eines Programms sollte der Anwender jederzeit ersehen können (ggf. durch Hilfeaufruf) in welcher Programmebene er sich zur Zeit befindet und wie der Systemzustand ist. Das System soll in diesem Sinn **selbsterklärend** sein. Weiterhin soll das System nur solche Kommandos anbieten und vom Benutzer erwarten, die zur Ausführung der anstehenden Arbeitsaufgabe nötig sind (**Problemangemessenheit**).

Wir haben oben die Frage gestellt, welche Anforderungen ein Benutzer an Software stel-len würde. Etliche Aspekte sind inzwischen angesprochen worden, so daß eine Zusam-menfassung mit einigen Ergänzungen angebracht ist.

Qualitätsmerkmale von Software aus Benutzersicht

Funktionalität	Leistungsfähigkeit, Befehlsumfang, Kompatibilität mit anderer Software, Hardwareanforderungen, Systemanforderung
Bedienbarkeit	Tastatur, Maus, Funktionstasten, einfache Tastenkombinationen, Hinweis auf Bedienelemente, Menügestaltung, Maskengestaltung, Menüverschachtelung, Art der Eingaben
Ergonomie	Optische Wirkung, Bildschirmaufteilung, Farbgestaltung, Anordnung von Informationen
Hilfestellungen	Informationen zu Bedienmöglichkeiten, Information zur aktuellen Bildschirmmaske, angemessene Kommentare, Hilfesuche mit Stichwörtern (Stichwortliste), Lernprogramme, Schulungsangebote, Hotline
Dokumentation	Umfang, Gliederung, Verständlichkeit, Stichwortverzeichnis, Beispiele, Demoprogramm, Lernprogramm, Handbuch oder Diskette

Eine entsprechende Zusammenstellung läßt sich auch für die Programmierersicht anferti-gen.

Qualitätssicherung

Im Prozeß der Softwareerstellung spiegelt sich die Frage der Qualität der zu erstellenden Software in dem Begriff "Qualitätssicherung" wider. Das Pflichtenheft für eine Softwareproduktion enthält außer den funktionalen Anforderungen auch Qualitätsanforderungen wie

- Benutzerfreundlichkeit (leichte Handhabung)
- Zuverlässigkeit (auch bei besonderen Bedienungsmaßnahmen)
- Wartbarkeit (Fehlerkorrektur oder Anpassung an veränderte Gegebenheiten, nicht nur durch den Autor)
- Portabilität (Übertragung auf andere Rechenanlagen ohne große Änderungen)
- Effizienz (Lösung in möglichst kurzer Zeit, mit möglichst geringem Aufwand)
- Ergonomie (Gestaltung des Programms entsprechend den geistigen und körperlichen Bedürfnissen der Benutzer)

Durch die Methoden des Software-Engineering versucht man, die Qualität zu sichern: **"Do the right things right the first time!"** Die Erfahrungen zeigen, daß die Einhaltung von Qualitätsanforderungen häufig nicht in dem gewünschten Maße gelingt.

Die Überprüfung der geforderten Qualität kann erfolgen durch

- statische Analyse (Untersuchung des Aufbaus des Produkts) oder
- dynamische Analyse (Untersuchung des Verhaltens der Software)

Statische Analyse für den Qualitätsfaktor "Wartbarkeit" (am Beispiel Prozeduren)
- Güte der Dokumentation (Kommentare, Prozedurspezifikation)
- Einfachheit (Schachtelungstiefe, Anzahl der Anweisungen, Anzahl der "Goto")
- Bezeichner (passende Benennung von Bezeichnern)
- Testbarkeit (Anzahl der zu testenden Pfade)

Metriken

Man kann den einzelnen Qualitätsfaktoren sogenannte "Metriken" zuordnen (zum Beispiel geeignete Maßzahlen) und damit zu einer quantitativen Abschätzung der Qualität eines Softwareprodukts kommen. Bei den Untersuchungen eines im Dialog arbeitenden Systems kann der Schwerpunkt liegen auf

(a) der **Dialogoberfläche** (Benutzerschnittstellen) oder

(b) dem **Dialoghintergrund** (die Funktionalität des Systems, die sich hinter dem sichtbaren Dialog verbirgt).

Zu (a): Für den Benutzer ergibt sich eine Folge von Bildschirmmasken. Man kann das Dialogsystem als einen Graphen darstellen. Die Knoten des Graphen sind die Bildschirmmasken, die gerichteten Kanten sind die Übergänge von einer Maske (i) zu einer Maske (j). Die verschiedenen Dialoge erscheinen als Maskenfolgen, die die unterschiedlichen Programmpfade repräsentieren.

Zu (b): Hinter jeder Kante aus (a) verbergen sich Funktionen für den Übergang von einer Maske zu einer anderen. Die einfachen Kanten wären nun in Mehrfachkanten aufzuspalten, die jeweils einer Funktion entsprechen.

2.2.9 Das Testen von Programmen - Testmethoden

Tests sind Verfahren, die dazu dienen, Fehler im Entwurf und im Programm zu erkennen und zu lokalisieren. Das Ziel ist, die Lösungen fehlerfrei zu machen. Je komplexer die Programmentwicklungsprozesse sind, desto größer ist die Wahrscheinlichkeit von Fehlern. Diese können schon bei der Entwicklung der Lösungsideen, aber wegen der notwendigen Genauigkeit auch bei der Übersetzung in ein Programm auftreten.

Für Spezifikation und Programm besteht zunächst das Problem, ob man zu jeder Fragestellung x eine korrekte Antwort erhält. Anschließend kann man zur Verifikation des Programms übergehen. Dabei wird mit formalen Methoden überprüft, ob sich für alle Fragestellungen x ergibt $S(x) = P(x)$.

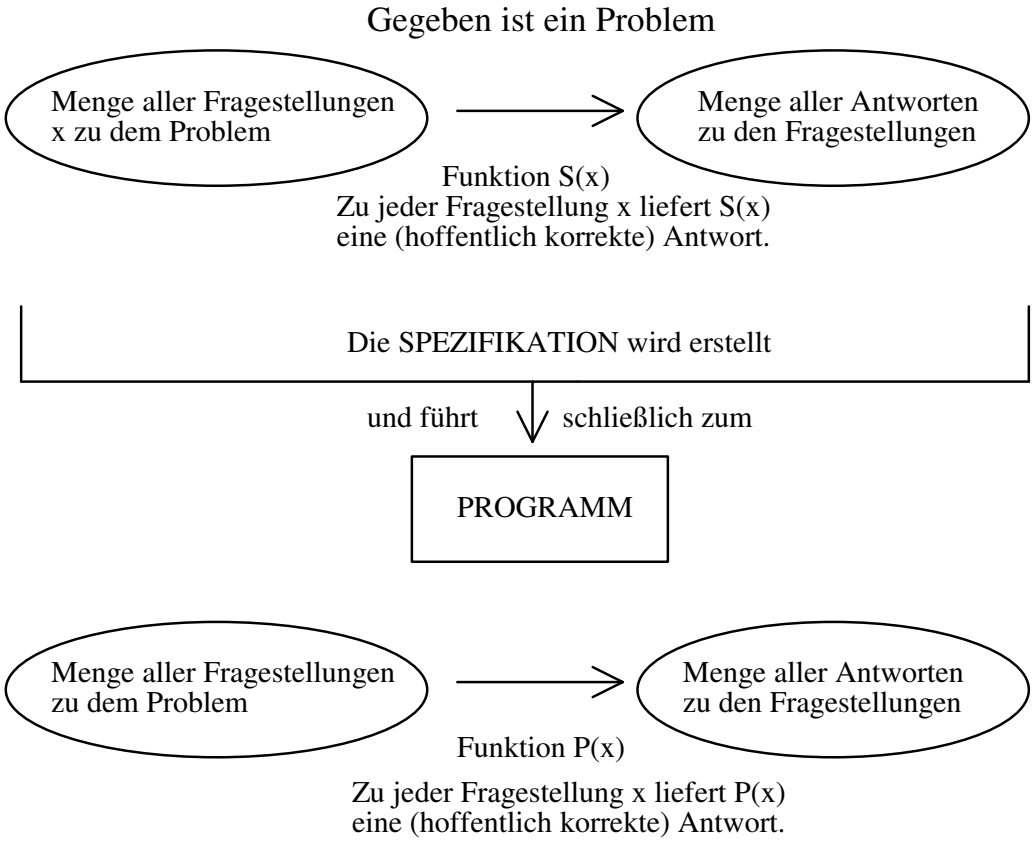


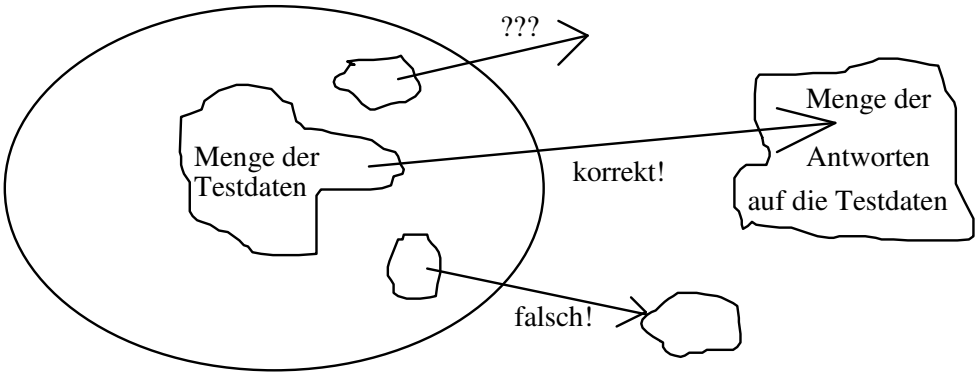
Abb. 2.2.9.a: Testen

Die Verifikation von Programmen, also der Nachweis mit formalen Methoden (ein Beweis!), ist wegen der großen Anzahl der möglichen Fragestellungen und der dann folgenden Verarbeitungsprozesse außerordentlich schwierig - bei den meisten Systemen unmöglich (und für die Schulinformatik schon gar nicht erreichbar).

Man hilft sich in der Regel mit einer einfachen, abgewandelten Form von "Verifika-tion", dem Testen. Hierbei wird nur eine Teilmenge der möglichen Fragestellungen untersucht.

Dabei ergibt sich sofort das Problem, wie man diese Teilmenge aussucht. Sie sollte "repräsentativ" für die Problemstellung sein. Aber was heißt das? - Über die Auswahl von Testdaten unten mehr!

Entwürfe können durch sogenannte **Schreibtischtests** überprüft werden. Ist z.B. ein Algorithmus in einem Struktogramm festgelegt, so kann man das Struktogramm mit seinen Feldern erneut hinzeichnen, aber die einzelnen Felder leer lassen. In diese Felder können dann Daten geschrieben werden, wie sie bei einem Programmdurchlauf mittels der Anweisungen des Algorithmus entstehen.



Durch Testen kann die Anwesenheit von Fehlern gezeigt werden, jedoch nicht ihre Abwesenheit!

Abb.2.2.9.b: Fehlerfrei?

Beim Programmtest wird das *Programm* gegen die *Spezifikation* getestet (siehe obige Abbildung), in der die erwartete Verhaltensweise des Programms genau dokumentiert sein soll. Anhand des Eingabe- und Ausgabeverhaltens des Programms überprüft man

- die formale Richtigkeit des Programms (Syntax)
- die logische Richtigkeit von Programmteilen (Semantik)
- die Funktionen einzelner Blöcke, Prozeduren und Module
- das Zusammenwirken der Module im Gesamtprogramm
- die Anfälligkeit bei Fehleingaben
- die Leistungsanforderungen (Speicherplatzbedarf, Verarbeitungsgeschwindigkeit,...)

Zu jedem zulässigen Eingabewert muß sich die laut Spezifikation vorgesehene Ausgabe ergeben. **Damit wäre die Korrektheit des Programms gezeigt.**

Vor der Testdurchführung

Zum Testen müssen geeignete Testdaten bereitgestellt werden. Hierfür werden einige Regeln angegeben:

- Die Menge der Testdaten sollte in Klassen (elementefremde Teilmengen) eingeteilt werden, deren Elemente jeweils gleiche oder ähnliche Eigenschaften haben. Von die-sen Klassen werden dann nur jeweils wenige Elemente eingegeben.
- Testmengen sollen typische und untypische Eingabewerte enthalten! Untypische Wer-te sind z.B. leere Eingaben, größte Werte, kleinste Werte.

- Man überlege sich vor der Durchführung, welche Ausgaben das Programm auf die Testeingabe hin liefern muß.

Strategien zur Testdurchführung

- Man unterteilt das Programm in zusammengehörige Abschnitte und testet diese einzeln. Wenn das Programm nach den Prinzipien der strukturierten Programmierung geschrieben ist, ergeben sich solche Abschnitte von selbst. **So geschriebene Programme sind testfreundlich!**

- **Strategie der Anweisungsüberdeckung:** Prinzipiell kann jede Anweisung des Programms zu einem Fehler führen, so daß eine Strategie sinnvoll erscheint, bei der jede Anweisung mindestens einmal durchlaufen wird. Man könnte daher die Menge der Testdaten so wählen, daß es zu jeder Programmanweisung eine Eingabe gibt, bei de-ren Verarbeitung diese Anweisung ausgeführt wird.

- **Strategie der Entscheidungsüberdeckung:** Grundlage ist eine Darstellung des Algorithmus in einem Programmablaufplan. Hier kann man die Menge der Testdaten so wählen, daß zu jedem Pfeil des Programmablaufplans eine Eingabe (Meßpunkt) erfolgt, bei deren Verarbeitung das Programm den Pfeil durchläuft.

Je nach Problem und Programmaufbau können verschiedene Teststrategien eingesetzt werden. Für ein komplexes modularisiertes Programmsystem bieten sich zwei Strategien an:

Test eines komplexen modularisierten Programmsystems

Top-Down-Testen: Das Testen beginnt auf der obersten Hierarchieebene und wird durch schrittweise Hinzunahme von Modulen der jeweils nächsten Hierarchieebene fortgesetzt, bis schließlich das Gesamtsystem mit all seinen Funktionen getestet werden kann. Das Erstellen von Hilfsprogrammen zum Test einzelner Module (siehe Bottom Up) erübrigt sich. Dafür müssen die Prozeduraufrufe /Modulaufrufe simuliert werden, etwa durch eine Meldung wie "Aufruf der Prozedur A ist erfolgt" (Dummy=Attrappe). Das jeweils einzufügende Modul muß mindestens eine Schnittstelle zu dem bereits getesteten Systemteil haben.

Beispiel:

MÜNZSPIEL

Spielen 1 (für Anfänger)
Spielen 2 (Fortgeschrittene)
Spielregeln
Textdatei zur freien Verfügung
Hinweise für den Lehrer
Programmende

Zum Auswählen Cursor auf /ab bewegen. Dann ENTER-

Taste drücken.

Das hinter dem Aufruf "Textdatei zur freien Verfügung" stehende Modul ist noch nicht in das System "eingehängt". Das wird beim Aufruf dieses Menüpunkt deutlich:

An dieser Stelle können Sie eine Textdatei mit eigenen Texten aufrufen. Die Datei heißt MUENZEN.AUF - sie muß als ASCII-Datei erstellt werden, z.B. im Turbo-Pascal-Editor.

Bottom-Up-Test: Das Testen beginnt auf der untersten Hierarchieebene durch Testen einzelner Module oder Prozeduren. Hierfür müssen i.a. kleine Hilfsprogramme geschrieben werden, die ein Hauptprogramm simulieren. Schrittweise werden dann die weiter oben stehenden Module zusammen mit den schon getesteten Untermodulen überprüft.

Je nach Organisation der Teamarbeit kann der Top-Down- oder der Bottom-Up-Test sinnvoll sein. Häufig ergänzen sich beide Methoden.

Test eines einzelnen Moduls

Ein solcher Test kann etwa so ablaufen: Die Modulumgebung, d.h. die anderen Module, mit denen das zu testende Modul zusammenarbeitet, seien bekannt, ebenso die Testdaten (s.u.), die an das Modul übergeben werden. Es geht nun um die Konstruktion eines geeigneten Hilfsprogramms (**Testrahmen**) mit folgenden Funktionen:

1. Aufruf des zu testenden Moduls TM mit Übergeben der Testdaten.
2. Simulation aller von Modul TM aufgerufenen Module mit Erzeugung geeigneter Kontrollausgaben und ggf. Rückgabe von Daten an TM.
3. Überprüfung der Ausgaben des Testmoduls TM, ggf. Kontrollausgaben.

Hinweise zur Fehlersuche

- Protokollierung der Programmausführung
- Für markante Stellen kann man im Programm (vorläufige) zusätzliche Ausgabeanweisungen oder Bedingungen in das Programm einschieben, denen Variable dort gehorchen müssen.
- Wichtige Hilfsmittel sind Debugger (bug , *engl.* = Wanze, debuggen = Fehler beseitigen),das sind Dienstprogramme mit denen man schrittweise die Ausführung des Programms verfolgen kann. Das Programm kann angehalten werden und man kann sich an den Haltepunkten (Breakpoints) z.B. momentane Inhalte von Variablen ansehen.

Test von (fertigen) Softwareprodukten

Für fertige Softwareprodukte existieren **DIN-Testnormen**. Zum Beispiel legt die DIN 66285 Gütebedingungen und Prüfbestimmungen für Anwendungssoftware fest und

dient damit u.a. als Grundlage für die Zertifikat-Erteilung von Softwareprodukten. Diese DIN-Norm fordert, daß ein Softwareprodukt aus Programmen, Daten und einer zugehörigen Dokumentation besteht, und daß eine Produktbeschreibung existieren muß. Diese Teile sind die Prüfgegenstände.

2.3 Projektdokumentation

Handbücher - Dokumentationshilfsmittel

Die Dokumentation des entstandenen Softwareprodukts, d.h. die Anfertigung von noch näher zu kennzeichnenden Schriftstücken, ist insbesondere aus zwei Gründen von großer Bedeutung:

(1) Das Produkt soll von möglicherweise vielen ungeübten Anwendern benutzt werden.
(2) Das Produkt muß neuen Anforderungen angepaßt (gewartet) werden.

Aus diesen Forderungen resultiert der Wunsch nach mindestens zwei Handbüchern:
(1) Benutzer-Handbuch, Adressaten sind die Benutzer des Systems
(2) Wartungs-Handbuch, Adressaten sind Systembetreuer beim Hersteller oder bei der Firma, die das Produkt erwirbt.

Wahrscheinlich haben Sie als Schüler auch bereits Software erworben. Welche Schriftstücke haben Sie beim Kauf erhalten?

Vermutlich haben Sie beim Kauf auch ein Handbuch erhalten. Für welche Zwecke haben Sie es benutzt?

Möglicherweise erscheint es Ihnen als überflüssig, zu dem Softwareprodukt, welches Sie gerade (z.B. im Schulunterricht) bearbeiten, eine längere Erläuterung aufzuschreiben. Schließlich können Sie das System aufgrund ihrer intensiven Arbeit daran verständig benutzen und sind auch in der Lage, eventuelle Änderungen durchzuführen. Was aber ist in einem halben Jahr? Was ist, wenn andere Schüler das System später überarbeiten sollen? Andererseits werden Ihnen Ihre eigenen Erfahrungen mit Software zeigen, daß eine gute Dokumentation der Software nötig ist, um damit - insbesondere auch vertieft - umgehen zu können. Deshalb darf man sich nicht auf das hergestellte lauffähige Programm beschränken, sondern muß es "dokumentieren". Auch in der Datenverarbeitungspraxis gibt es in diesem Zusammenhang erhebliche Probleme; dort kommt in der Regel noch der Termindruck bei der Entwicklung der Software dazu, der eine angemessene Dokumentation erschwert.

Wie oben schon bemerkt, ist die Dokumentation des Systems in erster Linie für andere Benutzer oder Programmierer bestimmt und für diese auch unbedingt nötig. **Die Überlegungen zeigen aber auch, daß es sehr auf die Qualität der Dokumentation ankommt und weniger auf einen großen Umfang**.

Die Dokumentation eines Softwareprodukts darf nicht zu einer Informationsflut führen, die die entscheidenden Aspekte verdeckt und den Anwender (Systembetreuer) "zuschüttet".

So ist es zum Beispiel nicht sinnvoll,
- den Entstehungsprozeß der Software sklavisch protokollieren zu lassen,
- alle benutzten Algorithmen graphisch oder anders darzustellen,
- Ansätze zu notieren, die als untauglich verworfen wurden,
- zu allen Änderungen bei der Erstellung einer neuen Version Änderungsprotokolle anfertigen zu lassen (bei weiteren Versionen sollten jedoch alle Änderungen gegenüber der Vorversion protokolliert werden, damit sich eventuell auftretende Fehler zurückverfolgen zu lassen).

Vielmehr halten wir fest

Die Dokumentation soll in möglichst übersichtlicher, knapper und verständlicher Form nur das enthalten,
- was der Benutzer für seine Arbeit mit dem System tatsächlich benötigt,
- was der Betreuer der Software für eventuelle Wartungsarbeiten, d.h. insbesondere Änderungen am System braucht.

Der Umfang der Dokumentationen läßt sich durch zwei Maßnahmen besonders einschränken:

- Das System soll sich möglichst weitgehend selbst dokumentieren (Benutzerfreundlich-keit, ggf. Hilfetexte).
- Da der die Wartung durchführende Systembetreuer in der Lage ist, den Quelltext zu lesen, kommt es sehr darauf an, daß der Quelltext an geeigneten Stellen Kommentare enthält (Spezifikation der Module, Prozeduren, Datenstrukturen). Das erspart das Notieren in der schriftlichen Dokumentation. Auf diese Weise ist auch eher gesichert, daß es sich um aktuelle Informationen handelt. Dazu können knappe Erläuterungen an kritischen Programmstellen kommen. Allerdings darf die Dokumentation im Programm nicht so umfangreich werden, daß man den eigentlichen Quelltext dazwischen "suchen" muß!

Die vorstehenden Überlegungen gelten allgemein für Softwareerstellung, besonders aber auch für die Schule. Dabei gehen wir davon aus, daß die Software nach ihrer Erstellung nicht beiseitegelegt wird, sondern später Verwendung findet.

Wer können in der Schule die Benutzer bzw. "Systembetreuer" sein?

(1) Es kann sich bei dem Produkt um Software handeln, deren Einsatz in anderen Schulfächern vorgesehen ist. Dann sind die Lehrer dieses Faches und ihre Schüler die Benutzer.
(2) Die Software kann etwa auch für die ITG (Informationstechnische Grundbildung) als Grundlage einer projektartigen Unterrichtsreihe vorgesehen sein. Dann sind jüngere Schüler, die Grundlagen der DVA und ihre Anwendung an einem ausgewählten Thema kennenlernen, die Nutzer.
(3) Die Software kann aber auch in der Informatik selbst als Studienobjekt eingesetzt werden, etwa bei der sogenannten Software-Analyse, u.a. mit dem Ziel der Wartung eines Produkts. Das beinhaltet dann auch Änderungen, also fungieren die Schüler als Systembetreuer.
(4) Möglicherweise ist die Software auch für die Schulverwaltung von Bedeutung.
(5) Denkbar ist auch die Weitergabe an andere Schulen oder an schulnahe Institutionen.
(6) Und schließlich werden die Schüler das erstellte Produkt mit nach Hause nehmen und damit einem ganz anderen Benutzerkreis zur Verfügung stellen.

Wir sehen:

Es gibt auch im Schulbereich und im Umfeld der Schule genügend Abnehmer für die erstellte Software. Und diese rufen nach einer Dokumentation! Zum ersten Abschätzen der Eignung des Produkts werden die Interessenten über eine Kurzinformation dankbar sein, also über einen "Werbe-Prospekt".

Was bleibt nun als sinnvolle Dokumentation für die Erstellung eines Softwareprodukts in der Schule übrig? Als Inhalte von Benutzer- und Wartungshandbuch werden vorgeschlagen:

Das Benutzerhandbuch (für die Programmbenutzer)

1. Inhaltsverzeichnis
2. Zweck des Programms, kurze überblicksartige Beschreibung seiner Funktionen, Modulhierarchie, soweit für die Bedienung wichtig (man kann hier z.B. an einen "Prospekttext" denken) - Fachkenntnisse werden in diesem Teil nicht vorausgesetzt.
3. Hardwarevoraussetzungen, Hinweise zur Installation
4. Beschreibung der Programmsteuerung: Menüs, Masken, Modulhierarchie, Beispiele
5. Vollständige Beschreibung aller Systemkommandos mit Details zu den einzelnen Programmfunktionen, dazu Beispiele (Nachschlagebuch (reference manual))

Das Wartungshandbuch (für das Wartungspersonal)

1. Inhaltsverzeichnis
2. Aufgabenstellung, Anforderungsdefinition (vom Auftraggeber)
3. Pflichtenheft (vom Hersteller)
4. Modulhierarchie
5. Datenstruktur, Datenformate

6. Hinweis auf Kommentare im Programmtext
7. Modulspezifikation
8. Spezifikation der Prozedurköpfe, Prozeduraufruf-Tabelle (cross-reference)
9. Spezifikation der Datenstrukturen
10. spezielle Erläuterungen
11. Programmcode
12. Anhang 1: Testdaten
13. Anhang 2: Revisionsprotokoll, Versionsnummern

Aus der Formulierung der **Anforderungsdefinition** durch den Auftraggeber und des **Pflichtenheftes** durch den Hersteller ergeben sich möglicherweise die Unterschiede zwischen dem Gewünschtem und dem Machbaren. Diese können für eine spätere Wartung (Änderung, Erweiterung) des Systems von Nutzen sein.
Der erste entscheidende Schritt für den Entwurf ist das Auffinden einer geeigneten Zerlegung, die sich dann in der Regel als **Modulhierarchie** darstellen läßt. An ihr erkennt z.B. ein Systembetreuer in welches Modul er sich hineinbegeben muß, um eine geplante Änderung vorzunehmen. Zudem sieht er die übergreifenden gegenseitigen Abhängigkeiten. Tiefergehende Abhängigkeiten lassen sich z.B. an cross-reference-Tabellen erkennen (--> Glossar). Danach kann er sich den einzelnen **Spezifikationen** zuwenden, die sich auch im **Programmtext** widerspiegeln. Als nützliche Hilfe für die Überprüfung eventueller Änderungen erweist sich die Zusammenstellung von **Testdaten**. Programmrevisionen müssen festgehalten werden.

Der "Werbe-Prospekt"

Produktinformation - Herstellung von Vortragsmaterial

Eine interessante, weil sehr realitätsnahe Aufgabenstellung kann darin bestehen, eine Werbeschrift für das entstandene Produkt zu entwickeln. Das kann teilweise bereits während der Produktentwicklung geschehen oder auch als ergänzende Aufgabe danach. Wir können uns auch in die Lage eines Firmenvertreters versetzen, der das Produkt werbewirksam anbieten soll, etwa auch in einer Veranstaltung zur Repräsentation des Produkts. Mit dieser Idee können mehrere Ziele verfolgt werden:

Der Werbeprospekt (für Kaufinteressenten)

1. Herausarbeitung der wichtigsten Leistungen des Produkts, Fähigkeit der Trennung von Wesentlichem und Unwesentlichem. Damit wird u.a. eine Reflexion über die geleistete Arbeit erreicht.
2. Durchführung zielgerichteter Überlegungen zur ansprechenden Gestaltung der Werbeschrift (Design-Entscheidungen).
3. Kennenlernen und Benutzen von Programmsystemen, die diese Design-Entscheidun-gen realisieren können (Textverarbeitungssystem, Desktop-Publishing, Grafiksysteme usw.).
4. Herstellen einer Demonstrationsversion des Produkts, ähnliche Zielsetzung wie in (1)

5. Auswahl und Herstellung von Vortragsmaterial (Demo-Version, Folien usw.).

Entwicklungsdokumentation (für die Projektbeteiligten)

Die vorstehenden Überlegungen zeigen, daß viele Teile der Gesamtdokumentation erst in der Endphase der Produktentwicklung erstellt werden können! Die sofortige **projektbegleitende Dokumentation** (**Entwicklungsdokumentation) gewinnt in der Schule ihren Wert insbesondere dadurch, daß man später den Ablauf des Projekt zusammenfassend kritisch reflektieren kann. Außerdem hat man im Unterricht die Möglichkeit, auf frühere Absprachen zu verweisen.** Insofern wird es sich im Wesentlichen um ein Abheften (in zeitlicher Reihenfolge) von Unterlagen handeln über

- organisatorische Maßnahmen
- Aufgabenbeschreibungen
- Arbeitsanweisungen, wie z.B. Programmierrichtlinien
- gemeinsame Zwischenbesprechungen, etwa in Form von Schülerprotokollen
- Beschreibungen von Arbeitszuständen an bestimmten Haltepunkten (Protokolle)
- Entwurfsentscheidungen, Quelltexte, Testprotokolle innerhalb der Arbeitsgruppen
- usw.

Für die Unterrichtspraxis empfiehlt sich ein zeitlich geordnetes Abheften der Unterlagen, ohne zwanghaften Drang nach Vollständigkeit. **Von diesen Schriftstücken sind diejenigen besonders wichtig, die später in die Gesamtdokumentation (Benutzerhandbuch, Wartungshandbuch) aufgenommen werden sollen. Es bringt nichts und ist (für den Schulunterricht) Zeitverschwendung, wenn man während der Entwicklungszeit laufend Papiere produziert, die sich später als unwichtig erweisen!**

Zusammengefaßt:

Die begleitende Dokumentation soll sich ausrichten nach
- den Anforderungen von Benutzerhandbuch, Wartungshandbuch (und Werbeschrift)
- der kritischen Reflexion des Projektablaufs am Ende des Projekts (Entwicklungs-dokumentation)

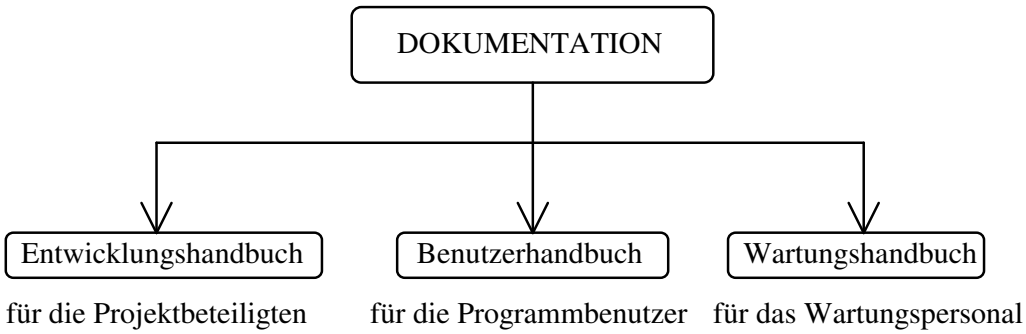


Abb. 2.3.a: Handbücher

Dokumentationshilfsmittel

Für die Dokumentation kann man sich verschiedener Hilfsmittel des Computers bedienen.

Für das Schreiben von Texten (Schüler-Protokollen) kann man z.B. den Turbo-Pascal-Editor benutzen und damit ASCII -Dateien erstellen. Komfortabler sind Textverarbeitungsprogrammem, in die man auch graphische Darstellungen einbauen kann. Ohnehin sollte man sich in der Dokumentation **häufig graphischer Darstellungen bedienen**, denn "ein Bild sagt mehr als 1000 Wörter". Für die Entwurfsphase sei hierfür besonders auf die in Kapitel 2.2.4 beschriebenen Hilfsmittel wie Struktogramm, SADT-Diagramm, Baumdiagramm usw. hingewiesen, die sich für übersichtliche Dokumentationen gut eignen. Die Übersichtlichkeit wird auch durch das **Anlegen von Tabellen** gefördert.

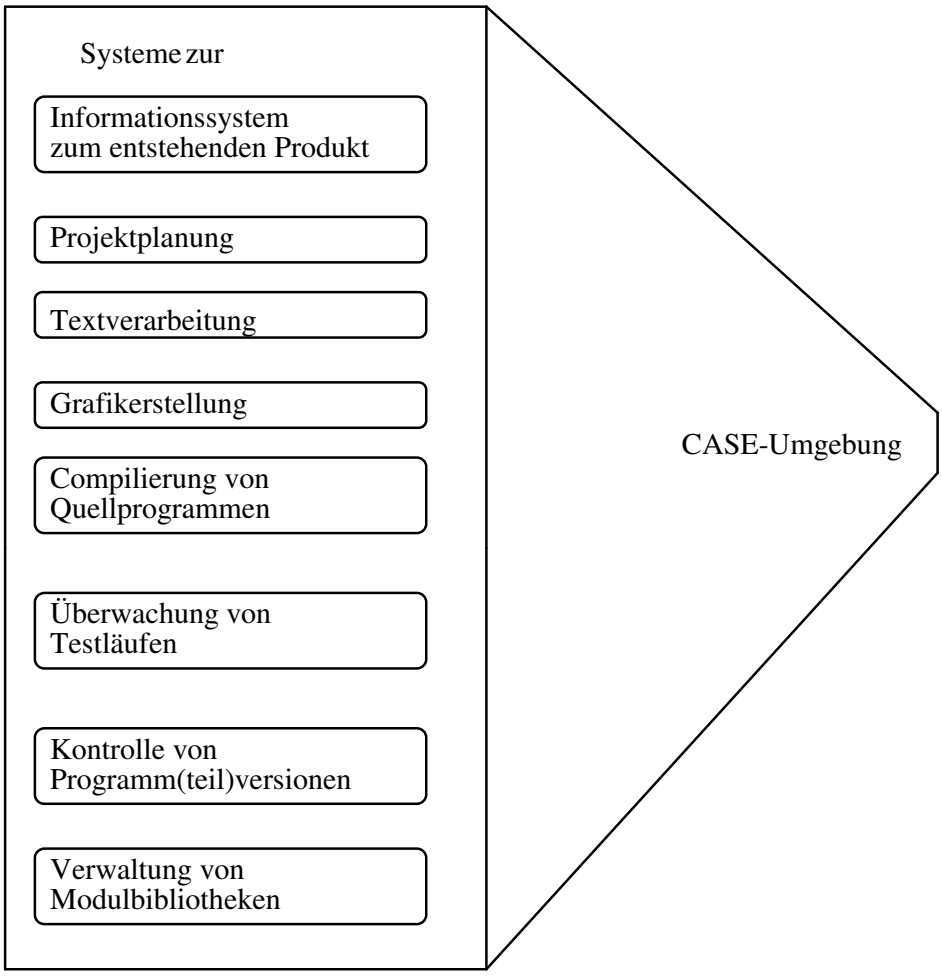


Abb. 2.3.b: Beispiel einer CASE-Umgebung

Die entstehenden Dateien können durch den Computer verwaltet werden. Dazu gehört z.B., daß die Namen der Dateien gut überlegt wurden, etwa durch sinnvolle Numerierungen, passende Bezeichnungen der Stellen nach dem Trennpunkt (z.B. xxx.pas) oder durch Berücksichtigung des Datums.

Die Verwaltung von Texten, der Aufbau von Hilfetexten oder der Aufbau eines kleinen Informationssystems über gewissen Schlagwörter kann gut mit der Prozedur OUTPUT_TEXTDATEI (...) erfolgen, siehe Kapitel 2.2.7.

In der Datenverarbeitungspraxis werden insbesondere für große Systeme mächtige "Werkzeuge" verwendet (Hilfsprogramme), die man unter dem Begriff **CASE** (computer aided software engineering) zusammenfaßt. Oft handelt es sich nur um einzelne Hilfsmittel; gelegentlich sind diese jedoch auch unter einer einheitlichen Oberfläche zu einer **Software-Entwicklungsumgebung** zusammengefaßt (CASE-Umgebung); siehe auch Kapitel 2.2.6.

2.4 PROJEKTKONTROLLE

2.4.1 Warnungen

Die Erfahrungen zeigen: Softwareprojekte in der Schule
werden oft nicht beendet!

Deshalb werden in diesem Kapitel einige Warnungen und Hinweise formuliert. Sie richten sich gleichermaßen an alle am Projekt Beteiligten, also an Lehrer und Schüler.

Warnungen

- Das Projektthema ist für die Teilnehmer irrelevant, es interessiert sie nicht
- die Rolle der notwendigen Vorkenntnisse wird unterschätzt
- die Einarbeitung in das Projektthema dauert zu lange
- man nimmt sich zuviel vor
- die Problembearbeitung geht am Ziel vorbei

- das Projektmanagement wird nicht ernst genommen
- die spezifischen Probleme bei Teamarbeit werden vernachlässigt
- man läßt die Projektideen zu sehr wuchern
- es fehlen Zwischenberichts- und Koordinierungsphasen

- dem Wunsch der Teilnehmer, frühzeitig mit dem Programmieren zu beginnen, wird nachgegeben

- der Einsatz von Bausteinen und Hilfsprogrammen (CASE) wird vernachlässigt bzw. die Einarbeitungszeit in diese Technik dauert zu lange

Der Projektleiter darf das Projekt nicht einfach so laufen lassen. Es darf nicht heißen, die Schüler werden es schon schaffen.

Hinweise zur Beschränkung des Projektumfangs

Die Zeitdauer eines Projektes kann durch verschiedene Parameter gesteuert werden. Eini-ge (vom Lehrer) einstellbare Parameter, die die Projektdauer beeinflussen werden ge-nannt.

Eine zeitliche Begrenzung wird u.a. durch folgende Maßnahmen erreicht:

- Vorgabe des Projektthemas und enge Projektführung durch den Lehrer
- Auswahl eines Themas, das nur wenige Kenntnisse aus dem gewählten Anwendungsbereich erfordert
- Beschränkung (zunächst) auf eine lauffähige Minimalversion
- durchgehender Einsatz von Bausteinen und Hilfsprogrammen
- stärkere Hilfe des Lehrers in verschiedenen Projektphasen
- Beschränkung der Testfälle
- Beschränkung der Dokumentation auf das für unbedingt erforderlich gehaltene Maß
- gezielte Hausaufgaben, die dem Projektfortschritt dienen (u.a. Ausnutzung der Rechnerausstattungen der Schüler)
- Vermeidung von Leerlaufphasen einzelner Schüler(gruppen)

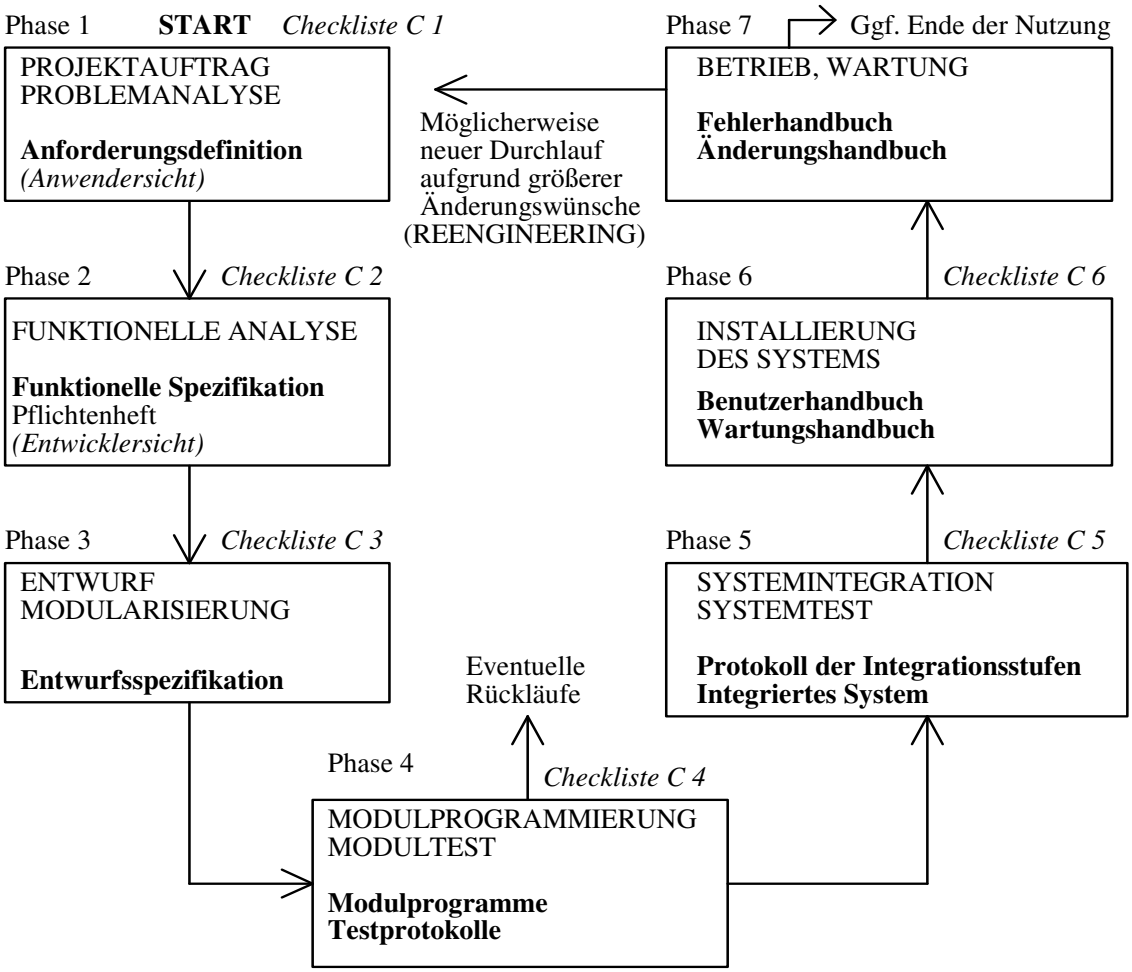
2.4.2 Checklisten für die Softwareproduktion

Die wichtigsten Probleme und Verfahren des Software-Engineering wurden in den vorhergehenden Kapiteln durchgesprochen. Wir sind nun hoffentlich gut vorbereitet auf das von uns zu bearbeitende Projekt. Übrigens: Wir könnten auch gleich mit dem vorliegenden Kapitel 2.4 beginnen und immer, wenn wir Unterstützung brauchen, in den zurückliegenden Kapiteln Hilfe suchen!

Es kann eine gute Methode sein, ein Projekt durch Checklisten zu begleiten. Diese sollte in der Regel der Projektleiter führen. **Im Informatikunterricht tragen die Checklisten u.a. dazu bei,**

- die Vollständigkeit der Arbeit zu überprüfen, also eventuelle Lücken aufzuspüren und zu beseitigen
- Arbeitsaufträge zu vergeben
- den Projektstand zu markieren,
- Überblick über die geleistete Arbeit und das noch zu Bearbeitende zu gewinnen,

Die Checklisten begleiten den Software-Life-Cycle (siehe Kapitel 2.1.1) , den wir deshalb hier noch einmal angeben und durch die Zuordnung von Checklisten ergänzen:



Checklisten für die Projekttätigkeiten in den Phasen der Softwareentwicklung

Hinweise: Die im folgenden formulierten Fragen sind nicht gleichermaßen für jedes Projekt wichtig. Sie sind abhängig vom Projektthema. - Die Checklisten sind auf die Belange der Projektdurchführung an Schulen zugeschnitten. Sie enthalten also wichtige Informationen für das organisatorische und methodische Vorgehen des Lehrers. Wichtig ist in diesem Zusammenhang auch das Kapitel über Projektmanagement, Kap.2.1.3.

Allerdings wird dringend empfohlen, sorgfältig zu prüfen, welche der Checklisten-punkte beachtet werden können, da sich bei Berücksichtigung aller Punkte leicht Zeitprobleme bei der Projektdurchführung einstellen können.

Außerdem wird ausdrücklich darauf hingewiesen, daß bei der Darstellung des Projekts "Trabrennbahn" in Kapitel 3 aus Platzgründen nicht alle Punkte der Checklisten - insbesondere, was die Dokumentation angeht- dargestellt werden können.

Phase 0: Projektbeginn
Der Beginn eines Projektes zur Softwareentwicklung aus Schulsicht

CHECKLISTE C-0

- *O Wie viele Schüler sind in dem Kurs, wieviele Mitarbeiter stehen zur Verfügung?*
- *O Wie ist die Leistungsstärke der Schüler?*
- *O Sind die fachlichen Kenntnisse für ein Projekt vorhanden oder sind noch grundsätzliche Vorbereitungen nötig?*
- *O Soll es um ein neues Projekt gehen, soll ein vorhandenes Produkt gewartet werden, soll ein ein angefangenes Projekt fortgeführt werden?*
- *O Wie ist die maschinelle und räumliche Ausstattung für ein Projekt?*
- *O Wieviel Zeit steht zur Verfügung?*

Phase I: Problemanalyse
(Anforderungsdefinition)

CHECKLISTE C-I

Erfassen des Istzustands

- *O Die bislang bestehende Organisationsform erfaßt?*
- *O Vorhandene Arbeitsabläufe festgestellt?*
- *O Ablauf der Prozesse analysiert?*

- *Datenflüsse im Arbeitsablauf aufgefunden?*
- *Wer erzeugt, wer benutzt die Daten?*
- *Datenstruktur erkannt?*
- *Verwendete Formulare erfaßt?*
- *Welche Qualitätsmerkmale sind vorhanden?*
- *Welches sind die Schnittstellen in der Organisation und in den einzelnen Arbeitsabläufen?*

Erfassen der Bedürfnisse

O Was soll sich ändern?

Formulierung der Anforderungsdefinition (Schriftstück)

O Einsatzzweck dargestellt?
O Gewünschter Leistungsumfang formuliert?
O Erwartete Arbeitsweise des Systems dargelegt?
O Über die vorgesehenen Benutzer informiert?
O Qualitätsanforderungen formuliert?
O Forderungen an den Datenschutz mitgeteilt?
O Ansprüche an die Datensicherheit notiert?
O Zeitvorstellungen mitgeteilt?

O Übergeben der Anforderungsdefinition an den Softwarehersteller

Phase II: Funktionelle Spezifikation (Pflichtenheft)

Der Auftraggeber hat die Anforderungsdefinition übergeben. Gestützt auf diese Vorstellungen muß nun detailliert beschrieben werden, was das Gesamtsystem und seine Teilsysteme leisten werden.

CHECKLISTE C-II

O Sind die personellen und maschinellen Voraussetzungen gegeben (siehe Checkliste C-0)

Formulierung des Pflichtenhefts (Schriftstück)

Man beachte bei der Dokumentation, daß graphische Darstellungen besonders aussagekräftig und übersichtlich sein können.

O Ist der Einsatzbereiches des Systems beschrieben?
O Werden die für den Systembetrieb notwendigen Hardware- und Software-Voraussetzungen genannt?
O Werden die organisatorischen Voraussetzungen beschrieben?
O Werden Angebote für die Einarbeitung der späteren Benutzer unterbreitet?
O Werden die Bedingungen zur Systemabnahme formuliert?
O Wird die Organisation der Inbetriebnahme dargestellt?
O Wird ein Terminplan vorgelegt?

- O *Sind Überblicke über die Systemfunktionen und Teilsystemfunktionen vorhanden?*
- O *Werden die Benutzerschnittstellen deutlich gemacht?*
- O *Wird die vorgesehene Benutzeroberfläche beschrieben?*
- O *Werden die Hilfefunktionen genannt?*
- O *Ist ersichtlich, an welchen Stellen und mit welchen Mitteln Daten eingegeben werden und um welche Daten es sich handelt?*
- O *Ist beschrieben, welche Daten an welchen Stellen ausgegeben werden? Werden die Ausgabegeräte genannt?*
- O *Werden die vorgesehenen Testdaten aufgeführt?*
- O *Wird die Reaktion auf Fehleingaben beschrieben?*

- O *Werden die Datensicherheitsaspekte dargestellt?*
- O *Werden die Datenschutzprobleme aufgezeigt?*

Phase III: Entwurf, Modularisierung (Entwurfsspezifikation)

CHECKLISTE C-III

An dieser Stelle soll noch einmal auf ein wichtiges Prinzip hingewiesen werden; siehe auch Kapitel 2.1.2 "Prototyping".

1. ***Die Gestaltung der Benutzungsoberfläche sollte vollkommen getrennt von dem eigentlichen Programm erfolgen (mit Menü- und Maskengeneratoren).***

2. ***Das eigentliche Anwenderprogramm sollte keinen (oder möglichst wenig) Code enthalten, der sich auf Präsentation und Interaktion bezieht. Damit wird seine Struktur übersichtlicher, und notwendige Wartungsarbeiten gestalten sich leichter.***

Entwurfsentscheidungen checken:

- O *Wurde das Gesamtsystem problemangemessen (in der Regel entsprechend den gewünschten Funktionen) in Teilsysteme zerlegt?*
- O *Wurden die Vorgaben für die Benutzeroberfläche eingehalten?*
- O *Wurden die Prinzipien des strukturierten Entwurfs eingehalten?*
- O *Können die gewählten Teilsysteme unabhängig von anderen Teilsystemen entworfen werden? Kann arbeitsteilig vorgegangen werden?*
- O *Wurden die Schnittstellen zwischen den Teilsystemen richtig definiert?*
- O *Ist die Einhaltung des Geheimnisprinzips möglich?*
- O *Ermöglicht die Zerlegung den Einsatz von vorhandenen Hilfsprogrammen, z.B. Maskengeneratoren?*

O Kann (aus arbeitsökonomischen Gründen) auf vorhandene Softwarebausteine zurückgegriffen werden?

Formulierung der Entwurfsspezifikation (Schriftstück)

O Ist die Zerlegung des Gesamtsystems in Einzelbausteine (Modulhierarchie) dokumentiert (auch graphisch)?
O Sind die Import- und Exportschnittstellen beschrieben?
O Sind die Module spezifiziert?
O Wurden die Testdaten notiert?
O Wird die Benutzeroberfläche dargestellt?
O Sind die Arbeitsaufträge an die einzelnen Teams eindeutig beschrieben?
O Ist ein Zeitplan für die Bearbeitung der Module erstellt? Sind dabei informelle Gespräche zwischen den Teams bedacht?

Phase IV: Modulprogrammierung, Modultest (Teilprogramme; Testprotokolle)

CHECKLISTE C-IV

Beachten die Bearbeiter die Methoden des strukturierten Programmierens ?

O Werden die zur Verfügung stehenden Hilfsprogramme für die Oberflächengestaltung eingesetzt?
O Werden die vorliegenden Programmbausteine benutzt?
O Halten sich die Teams an die Schnittstellenvereinbarung?
O Erfolgt eine angemessene Integration der Dokumentation in den Quelltext?
O Achten die Bearbeiter auf übersichtliche Gestaltung des Programmtextes?
O Werden aussagekrätige Bezeichner benutzt?
O Leisten die Module die verabredeten Funktionen?
O Wird der Zeitplan eingehalten?

Kontrolle beim Modultest:

O Werden die vorher festgelegten Testdaten verwendet?
O Werden Testprotokolle angelegt?
O Sind alle Unterlagen für das Wartungshandbuch gesammelt? Erlauben diese Daten eine spätere leichte Wartung der Programmteile?

Phase V: Systemintegration; Systemtest (Vollständiges Programm; Testprotokoll)

CHECKLISTE C-V

- O *Wird bei der Systemintegration schrittweise vorangegangen?*
- O *Werden die jeweils integrierten Teilsysteme getestet?*
- O *Wird das Gesamtsystem getestet?*
- O *Werden die vorher festgelegten Testdaten verwendet?*
- O *Werden Testprotokolle angelegt?*
- O *Sind alle Unterlagen für das Wartungshandbuch gesammelt? Erlauben diese Daten eine spätere leichte Wartung der Programmteile?*
- O *Entspricht die Funktionalität des Gesamtsystems den im Pflichtenheft festgelegten Anforderungen?*

Phase VI: Installierung des Systems (Abnahmetest)

CHECKLISTE C-VI

Hersteller und Auftraggeber:

- O *Werden bei der Installierung die Anforderungen des Benutzers beachtet?*
- O *Wird auf die Sicherheitsbestimmungen geachtet?*
- O *Erfolgt der Test des Gesamtsystems mit echten Daten?*
- O *Sind die geforderten Funktionalitäten und Leistungsmerkmale erfüllt?*
- O *Wurde das System vollständig geliefert?*
- O *Wurde die Systemdokumentation (Wartungshandbuch, Benutzerhandbuch) übergeben?*
- O *Ist der Übergang von der Projektorganisation zur Benutzerorganisation organisiert?*
- O *Wurde die Benutzerschulung eingeleitet?*

Mit dieser Phase endet die Projektorganisation. Es folgt die Phase des Betriebs und der Wartung, d.h. der Auftraggeber bzw. Benutzer übernimmt die Regie. Selbstverständlich wird auch hier der Hersteller verschiedentlich aktiv werden müssen, doch handelt es sich nun um die Betreuung des Softwareprodukts und nicht mehr um ein Projekt. Umfangreichere Wartungsarbeiten können aber möglicherweise später erneut zu einem projektartigen Vorgehen führen (Reengineering), sofern ein entsprechender Auftrag vorliegt.

3. Das Projekt "Simulation einer Trabrennbahn"

Auf den folgenden Seiten wird der Ablauf eines Projekts geschildert. Es handelt sich um ein Projekt, das mit 8 Schülern der Klasse 13 im Schuljahr 1988/89 durchgeführt wurde. **Das Thema "Simulation einer Trabrennbahn" erwies sich aus verschiedenen Gründen als attraktiv:**

- Realitätsnah - Besuch einer Trabrennbahn möglich
- anfangs offene Themenstellung - Thematik sehr umfassend, Passendes kann ausgewählt werden, Beschränkung auf Teilbereiche ist möglich
- interessante Algorithmen aus verschiedenen Bereichen
- Möglichkeit des Einsatzes von Grafik
- umfassendes Datenmaterial - Arbeit mit Dateien
- Notwendigkeit guter Benutzeroberflächen ein verschiedenen Teilbereichen
- Notwendigkeit der Konstruktion von Eingabemasken

Die Darstellung des Projektablaufs soll den tatsächlichen Ablauf möglichst gut widerspiegeln, u.a. wird verschiedentlich auf Schülerbeiträge zurückgegriffen. In weiten Teilen hat die Dokumentation daher den Charakter einer **Entwicklungsdokumentation,** siehe Kapitel 2.3.

Hinweis: Allerdings können hier aus Platzgründen nicht alle im Verlauf des Projekts erstellten Schriften abgedruckt werden. Es handelt sich um eine repräsentative Auswahl.

Bei der Realisierung der Entwürfe wurde bereits mit Bausteinen und Masken gearbeitet, allerdings in einer Erprobungsform gegenüber den in den Kapiteln 2.2.6 und 2.2.7 beschriebenen Möglichkeiten. Außerdem wird schon hier darauf hingewiesen, daß das entstandene Softwareprodukt im Schuljahr 1994/95 mit einem weiteren Informatikkurs überarbeitet, also gewartet wurde. **Auf die Wartungsarbeiten wird in Kapitel 4 näher eingegangen.**

Beachten Sie für die folgende Schilderung u.a. die Checklisten von Kapitel 2.4!

Original-Wettslip der Trabrennbahn und Quittung für einen Wetter

3.1 Sammeln und Auswerten von Informationen - Problemanalyse - Anforderungsdefinition

Der Beginn des Projekts fand auf der Trabrennbahn in Berlin-Mariendorf statt. Vorabsprachen mit der Verwaltung des Trabrennvereins ermöglichten uns die kostenlose Teilnahme an einem Renntag und den Besuch des Rechenzentrums der Trabrennbahn.

Aufgabenstellung 1
Der Besuch galt dem Sammeln von Informationen. Diese wurden dann im Unterricht in verschiedenen Arbeitsgruppen ausgewertet und teilweise schriftlich erfaßt (Dateien mit dem Namen DOK-PA*.TXT (PA : Problemanalyse)). Hierzu benutzten die Schüler ein Textverarbeitungsproblem. Die Abläufe auf der Trabrennbahn wurden analysiert.

Schülergruppe 1 schreibt:

Datei DOK-PA1.TXT
Ideensammlung zum Projekt Trabrennbahn *Problemanalyse - 19.09.1988*

(1) Organisation der Verwaltung
Die Organisation eines Rennens umfaßt nicht nur die für die Besucher sichtbaren Vorgänge, sondern auch die Kalkulation der Start- und Nenngelder sowie der Gewinnprämien. Außerdem muß die Organisation der Einzelrennen, die die Besetzung der Pferde, den Einsatz der Jockeys und die Belegung der Ställe der Trabrennbahn umfaßt, übersichtlich für die Teilnehmer (Besucher, Jockeys, Besitzer) gestaltet werden.

(2) Informationen für die Besucher
Um den Besucher über den Ablauf der Rennen zu informieren, werden Service-Leistungen angeboten. Die Informationen für die Besucher lassen sich aufteilen in folgende Punkte:
a) Informationen über die Wettregeln,
b) Informationen über die Rennvorgänge (Startbesetzungen, Anzahl der Rennen)
c) weitere Informationen sind im Programmheft enthalten.

(3) Wetten aus der Sicht des Kunden
Hat sich der Besucher für eine Wette entschieden, hat er sich einen Wettschein zu besorgen, ihn auszufüllen und am Schalter abzugeben (siehe 4). Die Auswertung des Wettscheins (zum Beispiel Ausgabe einer Quittung) und eine Auszahlung eines möglichen Gewinnes wird aus der Sicht des Kunden dargestellt.

(4) Wetten aus der Sicht des Bearbeiters
Für den Besucher nicht sichtbar werden die Daten des Wettscheins z. B. zur Bestimmung der Wettquoten oder hochgerechneter Rennergebnisse durch die Bearbeiter über ein Terminal in den Großrechner des Rechenzentrums eingegeben.

(5) Das Rennen
Das Rennen wird durch eine Schautafel über das aktuelle Renngeschehen informiert. Sie zeigt folgendes an:
1) Pferdenamen, 2) Jockeybesetzung, 3) Startnummern, 4) Wettquoten.
Die Schautafel wird zentral vom Rechenzentrum mit Informationen gespeist. Außerdem wird während des Rennablaufes ständig die Einhaltung der Rennregeln über Kameras und Monitore vom Schiedskommitée überwacht. Die Schiedsrichter haben die Möglichkeit, sich die einzelnen Bilder nachträglich in verschiedenster Weise anzusehen.

** Kommentar des Lehrers **
In dem Beitrag wird in erster Linie der sog. "Istzustand" geschildert, eine für die Einschätzung des Projekts wichtige Dokumentation. Eine Ideensammlung konnte hier auch verstanden werden als Zusammenstellung von durch den Computer zu bearbeitenden Sachverhalten.

An Informationsmöglichkeiten zur Istanalyse (Problemanalyse) des komplexen Systems "Trabrennbahn" standen zur Verfügung:

- Die Zeitschrift "Traber aktuell" mit Informationen zu den einzelnen Rennen, mit Handlungsanweisungen für das Wetten und mit Artikeln zum Anwendungsbereich
- Wettscheine (Mariendorf-Wettslip) mit Quittung (Beleg)
- Führung durch das Rechenzentrum mit dem RZ-Leiter
- Ansagen des Stadionsprechers, Anzeigetafel, Monitorbild vom Rennverlauf
- staatliche Wettvorschriften (Wettsteuer usw.)
- Beobachtungen "vor Ort" u.a. Befragungen von Angestellten und Wettern

In der Ideensammlung zum Projektthema wurden zunächst alle Einfälle notiert, so daß eine umfangreiche (zunächst ungeordnete) Liste entstand. Nach einer (mühsamen) Einordnung in größere Bereiche entstand schließlich ein übersichtliches Bild über die anfallenden Problemstellungen, siehe Abbildung 3.1.a.

Diese umfangreiche Zusammenstellung zeigt, daß die Bearbeitung all dieser Punkte in einem Projekt, das bestenfalls ein halbes Jahr (mit 3 Stunden Informatik pro Woche, 8 Schüler) dauern darf, unmöglich ist. Die Zielsetzungen müssen stark eingeschränkt werden. Die **Anforderungsdefinition** (das abschließende Dokument zur Phase "Problemanalyse /Anforderungsdefinition", das der Auftraggebers dem Software-Hersteller vorlegt - hier jedoch mangels eines echten Auftraggebers vom Kurs erstellt) könnte zwar ein solches um-fangreiches System fordern, jedoch müßte der Software-

Hersteller nach einer sorgfältigen funktionellen Analyse in dem von ihm vorgelegten **Pflichtenheft** vermutlich starke Ein-schränkungen machen oder den Auftrag ablehnen.

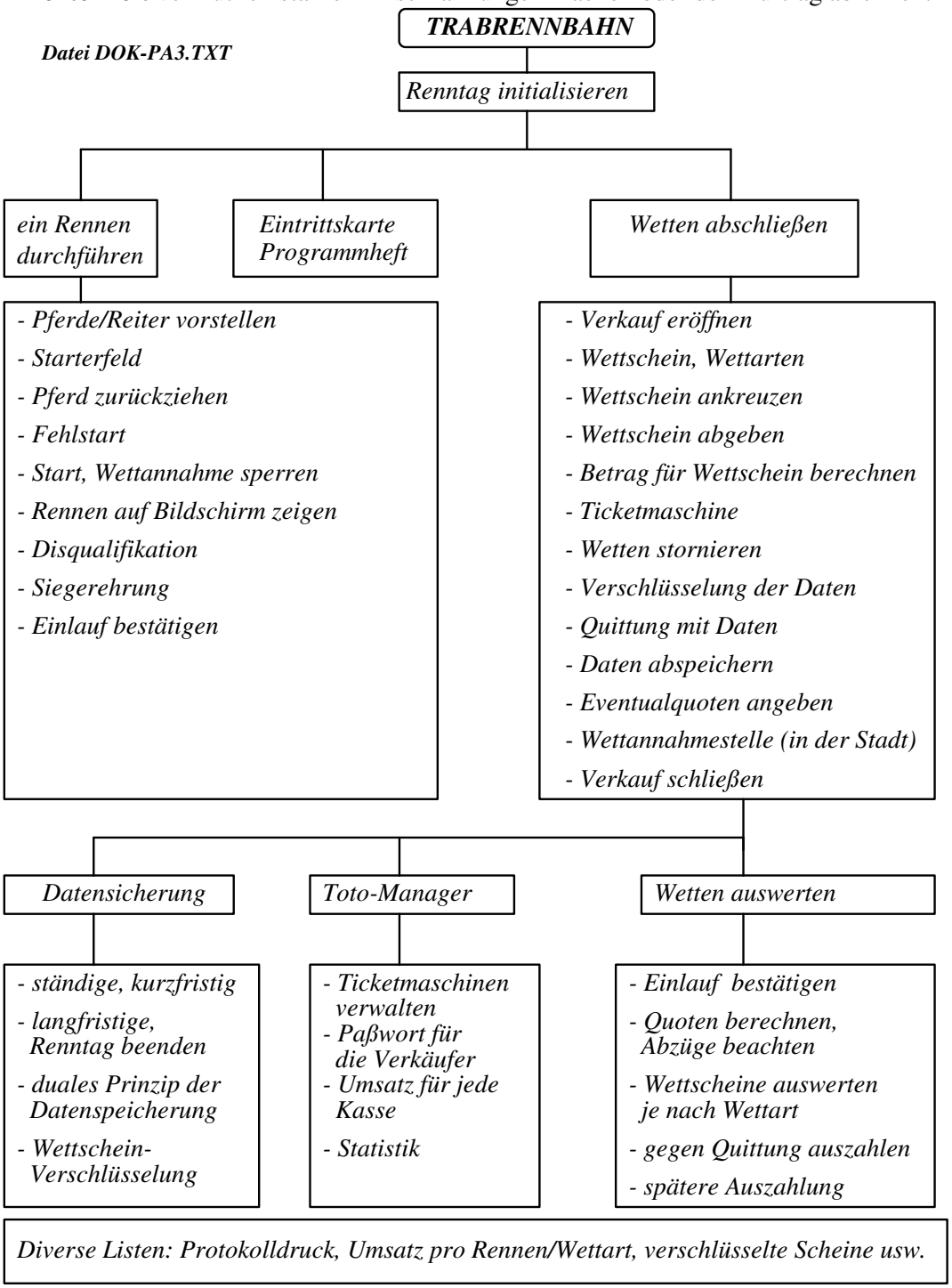


Abb.3.1.a: Ideensammlung, schon geordnet
Hinweis: Bei einem Projekt, das sich ein Informatikkurs selbst stellt, vermischen sich die Phasen "Problemanalyse/Anforderungsdefinition" und "Funktionelle Analyse /Pflichtenheft". Es gibt jedoch auch für die Schule zahlreiche Themenstellungen, bei denen ein echter Auftraggeber auftritt. Beispielsweise kann ein Erdkundelehrer ein Lernprogramm zur Topographie von Europa bestellen und dabei seine Wünsche (Anforderungen) in Form einer Anforderungsdefinition schriftlich formulieren (siehe auch Kapitel 1.3). - Wir gehen in der Projektdarstellung daher gleich zur nächsten Phase "Funktionelle Spezifikation / Pflichtenheft" über!

3.2 Funktionelle Spezifikation und Pflichtenheft

Wir haben uns nun aus Herstellersicht mit den Funktionen zu beschäftigen, die das geplante System ausführen soll und diese dem Auftraggeber in Form eines Pflichtenheftes verbindlich mitzuteilen.
Beachten Sie aber die Ausführungen in Kapitel 2.1.2 über die Methode des Prototyping! Bei einem derartigen Vorgehen werden Auftraggeber und Hersteller eng zusammenarbeiten, so daß die Anforderungsdefinition letztlich entfallen kann. Wichtig bleibt das Pflichtenheft, auf dessen Aussagen sich der Auftraggeber später auch rechtlich berufen kann.

Aufgabenstellung 2
Nach den Ausführungen in Kapitel 3.1 geht es nun um eine Festlegung des in dem Projekt Machbaren. Wir gehen dazu von der graphischen Darstellung der Ideensammlung in Kapitel 3.1 aus und reduzieren die genannten Punkte auf das, was uns möglich erscheint. So entsteht Abbildung 3.2.a.

Aufgabenstellung 3
Beschreiben Sie die geplanten Programmsystem-Funktionen (in grober Form) und die vorgesehene Benutzeroberfläche (Hauptmenü).

Datei DOK-PF0.TXT
Bei Aufruf des Programms wird das Hauptmenü erscheinen:

```
Hauptmenü          TRABRENNBAHN SCHÖNEBERG
                        1: Renntage organisieren
                        2: Rennen starten
                        3: Wetten annehmen
                        4: Gewinne auszahlen
                        5: Informationen
                        6: Hilfe
```

7: Ende des Programms
Auswahl einer Option mit den Cursortasten ↑↓, Fortsetzung mit ↵

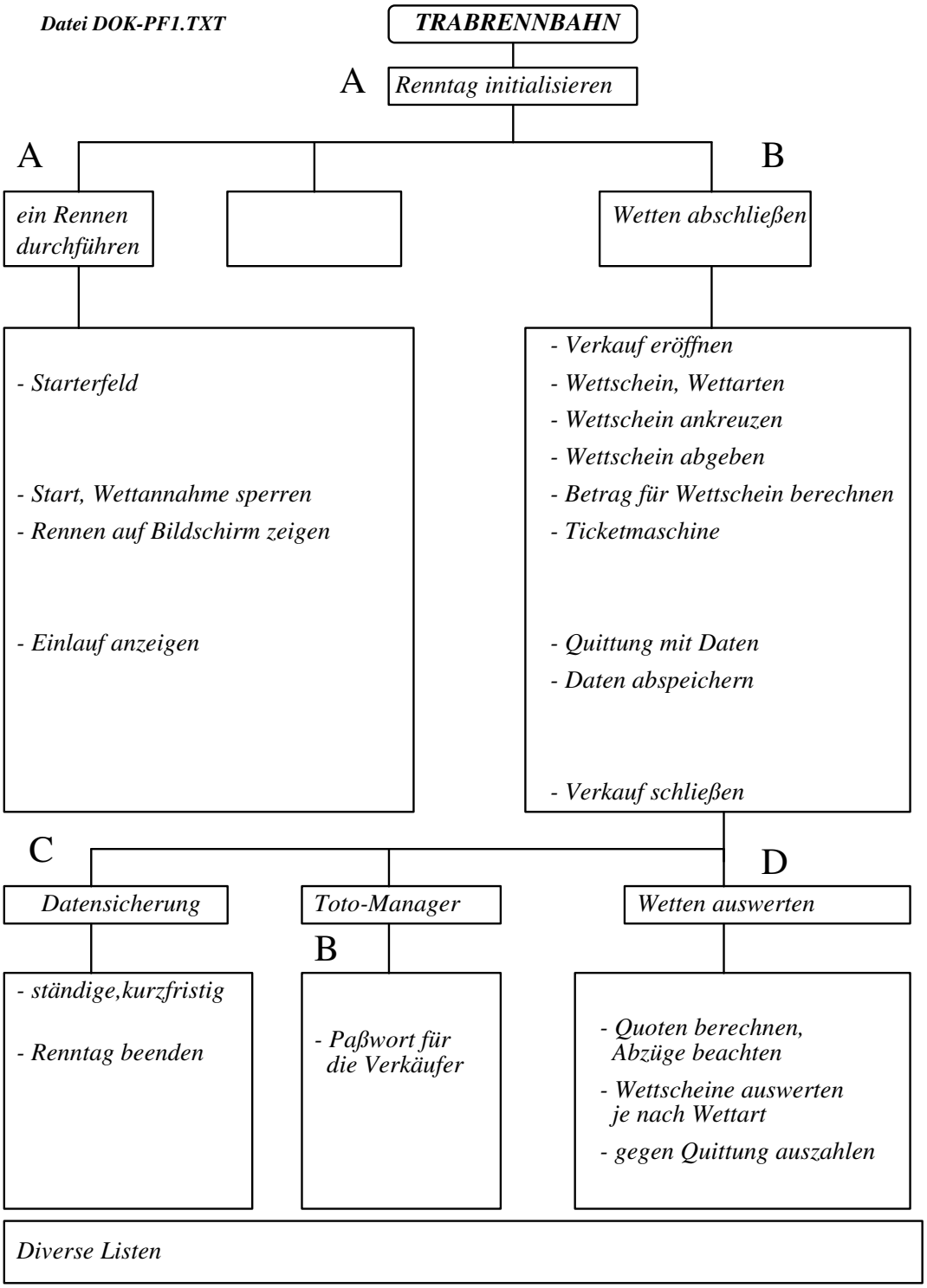


Abb. 3.2.a: "Bereinigte" Ideensammlung - Einengung der Problemstellung

Das Programmsystem "Trabrennbahn" enthält also im Hauptmenü sieben Optionen.

***Option 1** ermöglicht die Eingabe der grundlegenden Daten für einen Renntag. Das sind Tag und Namen der für ein Rennen startenden Pferde.*

***Option 2** steuert die eigentlichen Rennen. Diese können auf Wunsch gestartet werden und werden auf dem Bildschirm simuliert. Ein Rennen endet mit der Ausgabe des Einlaufs.*

***Option 3** ermöglicht die Abgabe von Wettscheinen. Ein Rennen eines Renntags wird aufgerufen. In die Bildschirmmaske können nun per Handeingabe die Wettart, der erwartete Einlauf und der Einsatz eingegeben werden. Der Wettschein erhält eine Nummer. Um für die Auswertung mehr Wettscheine zur Verfügung zu haben, kann man außer der Handeingabe auch Wettscheine in großer Anzahl zufällig erzeugen lassen.*

***Option 4** übernimmt die Auswertung der Wettscheine, nachdem ein Rennen durchgeführt wurde (siehe Modul 2). Hierbei werden die vorgeschriebenen Abzüge von der für dieses Rennen eingenommenen Geldsumme vorgenommen. Die Gewinn-Wettscheine werden markiert und zur Auszahlung freigegeben.*

***Option 5** wird Informationen zum Programmsystem enthalten. Im zweiten Teil dieses Aufrufs ist dann ein "Film" vorgesehen, der weitere Informationen, wie z.B. die Eingabemasken bringt.*

***Option 6** soll den Aufruf einer Hilfedatei ermöglichen, in der Hilfen auf Stichworteingabe möglich sind.*

***Option 7** beendet das Programm.*

*An verschiedenen Stellen des Programms wird man zur Eingabe eines Datums aufgefordert. Zuvor wird in der Regel eine Liste ausgegeben, in der die **Daten der Renntage** enthalten sind. Die **Auswahl von Menüpunkten** (Optionen) erfolgt durch die Cursor (auf, ab) mit anschließendem Return oder durch Eingabe bestimmter Zeichen. Für die Eingabe der Teilnehmer eines Rennens und für die Ausfüllung von Wettscheinen werden **Masken** vorgegeben, die man schrittweise ausfüllen kann. Die Optionen des Hauptmenüs führen meistens weiter zu **Untermenüs**, auf die in dieser groben Beschreibung nicht weiter eingegangen wird (siehe die Detailbeschreibungen zum Pflichtenheft). Im Übrigen soll das Programm so geschrieben werden, daß es sich für den Benutzer möglichst selbst dokumentiert.*

Das Programm wird für zwei unterschiedliche Gruppen angelegt:
*- Für den **Spieler**, der die Trabrennbahn-Simulation durchführt.*
*- Für den **Verwalter**, der rennspezifische Daten eingeben darf, also mehr Rechte hat als ein Spieler.*

Für das Pflichtenheft sind nun die Funktionen der weiteren Menüs und Eingabemasken zu beschreiben.

Aufgabenstellung 4
Abbildung 3.2.a zeigt die diversen Einschränkungen und gibt gleichzeitig mit den Buchstaben A,B,C,D eine mögliche Zerlegung der Probleme an, die sich in dem Hauptmenü widerspiegelt und die an verschiedene Arbeitsgruppen übergeben werden kann. - Jede Gruppe kann nun eine Beschreibung der einzelnen auszuführenden Funktionen für ihren Bereich durchführen.

Als Beispiel wird hier der Schülertext von Gruppe B dokumentiert:

PHASE: FUNKTIONELLE ANALYSE - PFLICHTENHEFT / ***Datei DOK-PF3.TXT***
Gruppe B: (Schülernamen) *17.10.88*
==========
WETTEN-ANNAHME

Folgende Eingaben bzw. Ausgaben werden erfragt bzw. durchgeführt:

Vorgang	*Eingaben*	*Ausgaben*
Wettschein ausfüllen	*1) Nummer des Rennens* *2) Gewählte Startnummern ankreuzen* *3) Art der Wette bestimmen* *a) Siegwette* *b) Platzwette* *c) kleiner Einlauf* *d) großer Einlauf* *4) Grundbetrag in DM*	
Ticketmaschinen starten	*Paßworteingabe des Verkäufers* *Kontrolle* *Eröffnung des Verkaufs*	 *Bestätigung* *Bestätigung*
Wettscheine in Ticketmaschinen einlesen und Preis berechnen, Quittung erstellen (Daten intern speichern)	*Daten des Wettscheins ein-lesen, Kontrolle*	*Bestätigung* *Preis* *Quittung*

Ticketmaschine/
Verkauf schließen *Ende-Kommando* *Bestätigung*

Die funktionelle Analyse muß auch zu den Menüs und Eingabemasken führen, mit denen der Anwender arbeiten soll. Insofern sind hier zumindest schon erste Entwurfsarbeiten nötig, die insbesondere die in der Systemoberfläche sichtbaren Schnittstellen betreffen.

Hier werden noch 2 Menüs bzw. Masken vorgelegt:

R E N N E N I N I T I A L I S I E R E N Menü1-1.mas

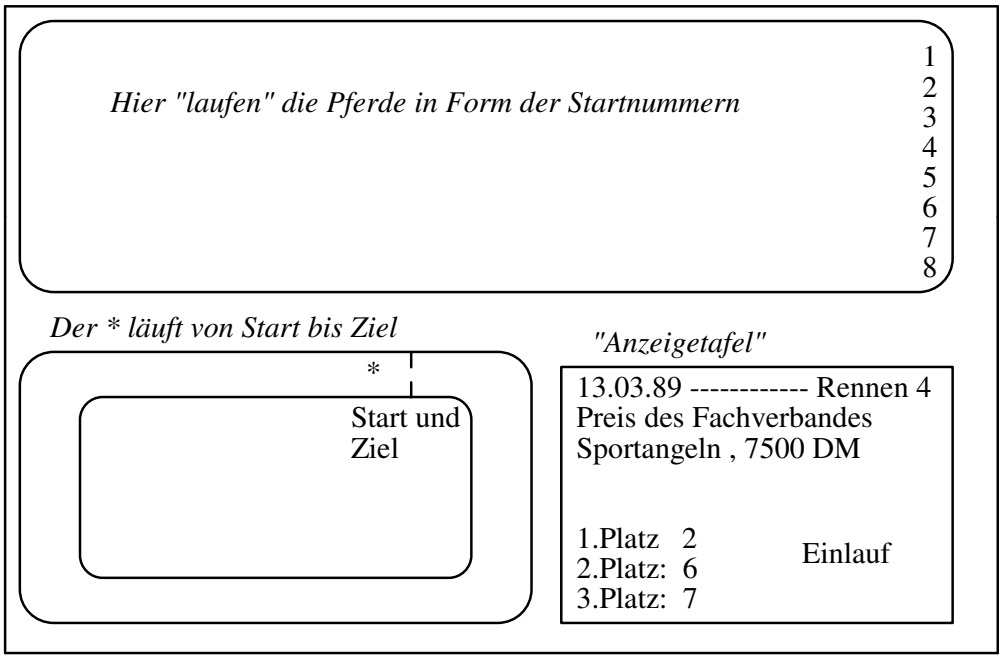
1: * Eingabe von Datensätzen
2: * Löschen von Datensätzen
3: Listen von Datensätzen
4: Informationen
5: Zurück zum Hauptmenü

Auswahl einer Option
mit den Cursortasten "auf,ab".

* Die Nummern 1 und 2 sind nur für Operateure zulässig

Das Trabrennbahnbild (Bildschirmanzeige) wurde folgendermaßen entworfen:

T R A B R E N N B A H N



Hinweis: Die anderen Menüs und Masken werden in Kapitel 3.3 (Entwurfsphase) abgebildet. Weitere Beschreibungen zum Pflichtenheft finden Sie auf der Diskette: Es sind die Dateien DOK-PF?.TXT (mit WORD5 erstellt).

3.3 Entwurf, Modularisierung - Entwurfsspezifikation

Prototyping

In Kapitel 2.1.2 werden die Grundzüge des Prototyping dargestellt. Im Prototyping vermischen sich die Phasen Problemanalyse, funktionelle Analyse und Entwurf. Im vorliegenden Projekt wären in der Entwurfsphase nach Konstruktion der Menüs und Eingabemasken Rücksprachen mit dem Anwender nötig, um sich davon zu überzeugen, ob dieser mit den konstruierten Schnittstellen (der Programmoberfläche) einverstanden ist oder Änderungswünsche hat. Dafür würden die einzelnen Aufrufe und Eingaben durch vorläufige Festlegungen zu simulieren sein.

In die Phase der Entwürfe fällt auch die Angabe von Testdaten für die einzelnen Module. Mit den Testdaten kann einerseits bereits hier ein "Trockentest" erfolgen (Programmcode liegt ja noch nicht vor), andererseits kann man aber beim Prototyping (und erst recht in der Schule) die Tests auch an den Programmteilen durchführen, zumal insbesondere die kleineren Programmteile heutzutage nach einer kurzen Entwurfsüberlegung sofort am Rechner konstruiert werden können.

Datenmodul: Datenstruktur und Schnittstellen (Modul T_DS_U.PAS)

Die Überlegungen in den vorhergehenden Phasen haben gezeigt, daß vier große Bereiche zu unterscheiden sind:

I Renntag initialisieren	III Wetten-Annahme
II Ein Rennen durchführen	IV Wetten auswerten, Gewinne auszahlen

Zwischen diesen Bereichen müssen Daten ausgetauscht werden (**Schnittstellen**)! Bevor wir also an die Bearbeitung der Module gehen, müssen wir uns mit der Datenstruktur des Systems beschäftigen. - Die von uns erarbeitete Datenstruktur für das System TRABREN-NEN faßt die Daten folgendermaßen zusammen:

Bereiche I und II Datei ***renntage : FILE OF rennen;***
Bereiche III und IV Datei ***wetten : FILE OF wettschein;***

Die Dateistrukturen bestehen aus geeignet zusammengestellten Verbunden:

```
TYPE rennen  = RECORD
                 tag                 : STRING[8];
                 rennen_nummer       : BYTE;
                 kommentar           : ARRAY[1..3] OF STRING[40];
                 anzahl_pferde       : BYTE;
                 pferde_namen   : ARRAY[1..8] OF STRING[12];
                 einlauf             : ARRAY[1..8] OF BYTE;
                 zustand             : (vorher,laeuft,beendet);
               END;
```

Beispiel für einen Datensatz vom Typ "rennen"

21.11.88	3	Preis der Sparkasse Preisgeld 25000 DM Favorit: Hamsterboy	8	Hamsterboy Williams Adonie Karoline Goldfisch Halali Goldjunge Silberfee	0 0 0 0 0 0 0 0	vorher

```
TYPE wettschein  = RECORD
                     tag                 : STRING[8];
                     rennen_nummer       : BYTE;
                     satz_nummer         : INTEGER;
                     { hierdurch wird der schnelle Zugriff auf den zum Wettschein gehörenden
                     Satz von "file of rennen" ermöglicht }
                     wettart             : (sieg,platz,kew,gew);
                     platz1,platz2,platz3 : BYTE;
                     einsatz             : REAL;
                     schein_nummer       : INTEGER;
                     gewinn              : REAL;
                     ausgezahlt          : BOOLEAN;
                   END;
```

Beispiel für einen Datensatz vom Typ "wettschein"

27.11.88	3	15	kew	4 1 7	10	722	120	false

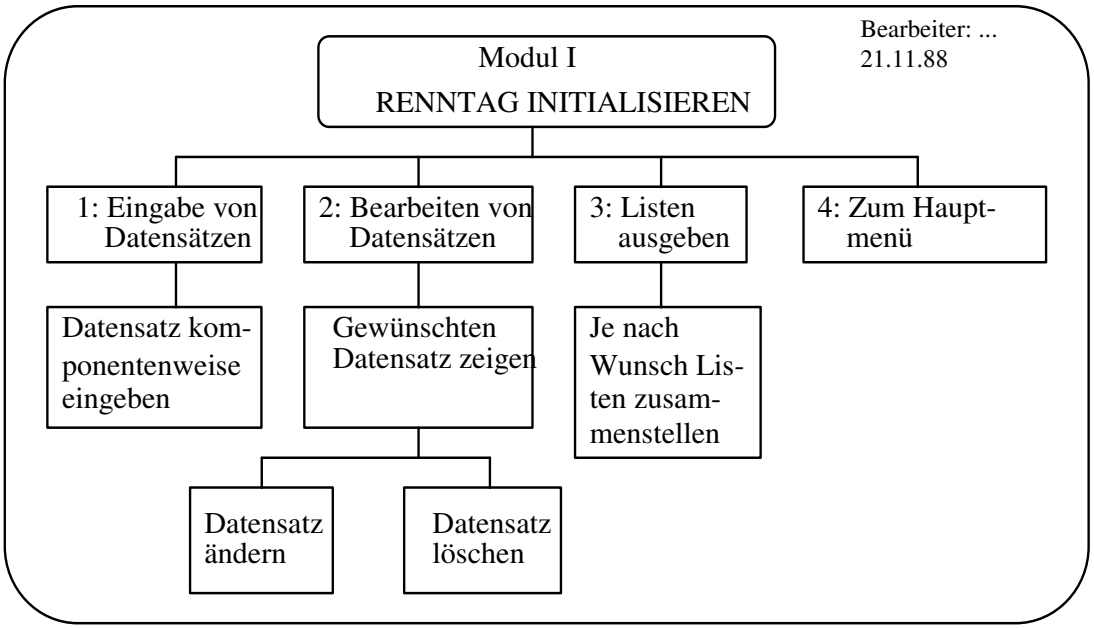
Der Wetter trägt in seinen Wettschein u.a. Tag (sofort prüfen!) und die Nummer des Rennens ein. Falls beide Eingaben vernünftig sind, wird der zugehörige Satz in "file of rennen" gesucht und die "satz_nummer" des Wettschein erhält ihren Wert. Damit ist bei der Auswertung ein sofortiger Zugriff auf "file of rennen" möglich.

Entwürfe der Module - Menüs

Arbeitsauftrag

Die für die einzelnen Module verantwortlichen Arbeitsgruppen entwerfen eine Überblicksdarstellung (Hierarchie) für ihr Modul, zerlegen es also weiter in Teile. Außerdem sind die Schnittstellen zu konstruieren (Menüs, Eingabemasken) .

MODUL 1: RENNTAG INITIALISIEREN



Dieser Entwurf bestätigt das bereits im Pflichtenheft vorgelegte MENÜ1-1.

Menü1-1.mas

R E N N E N I N I T I A L I S I E R E N

*1: * Eingabe von Datensätzen*
*2: * Löschen von Datensätzen*
3: Listen von Datensätzen
4: Informationen
5: Zurück zum Hauptmenü

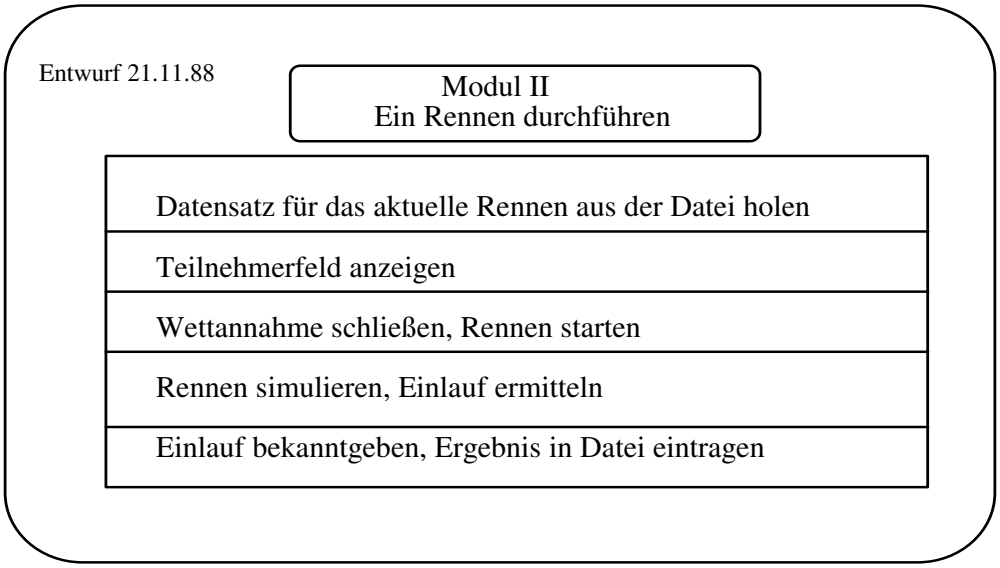
Auswahl einer Option mit den Cursortasten "auf,ab".

** Die Nummern 1 und 2 sind nur für Operateure zulässig*

Nun wird noch eine Maske benötigt, um die Datensätze eingeben zu können. Diese soll folgendermaßen aussehen:

```
                        EINGABE  VON  DATENSÄTZEN                 MENÜ1-2
------------------------------------------------------------------------------
 Datum :                Rennennummer :          Anzahl der Pferde :
 [z.B.   01.04.89]

 Pferdenamen :  1.                 Kommentar :
                2.                ----------------------------------------
                3.                |                                      |
                4.                ----------------------------------------
                5                 |                                      |
                6.                ----------------------------------------
                7.                |                                      |
                8.                ----------------------------------------

                                  DATENSATZ übernehmen (j,n) ?

                                  Weitere Eingaben     (j,n) ?
```

MODUL 2: EIN RENNEN DURCHFÜHREN



Beispiel für die Anzeige des Teilnehmerfeldes:

S T A R T E R F E L D
13.03.89 4.Rennen

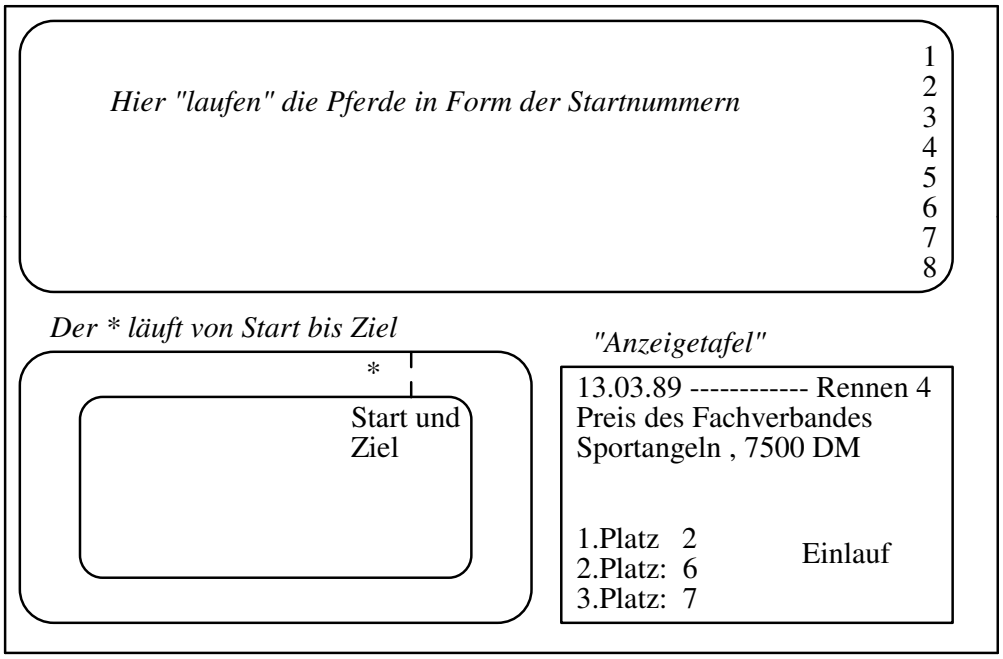
Preis des Fachverbandes Sportangeln über 7500 DM
(3750 DM, 1875 DM, 900 DM, 600 DM, 375 DM)

1 Chiantis Boy	2 Otsche
3 Ilo Lojbjerg	4 Gral
5 Galadys	6 Fasching
7 Vario	8 Vajan

Weiter mit RETURN

Das Trabrennbahnbild (Bildschirmanzeige) wurde folgendermaßen entworfen:

T R A B R E N N B A H N



Das Bild zeigt den Endstand eines Rennens. Auf eine bessere Grafik haben die Schüler hier verzichtet. Ihr Interesse galt seinerzeit insbesondere dem Wetten.

MODUL 3: WETTEN ABSCHLIEßEN

Diese Arbeitsgruppe hat zunächst das Menü entworfen, durch deren Funktionen die Wettscheine verwaltet werden können:

```
                    W E T T E N   A N N E H M E N          MENÜ3-1.MAS

        1:    WETTSCHEINE VON HAND AUSFÜLLEN

        2:    WETTSCHEINE ZUFÄLLIG AUSFÜLLEN LASSEN (erst in die Maske)

        3: *  WETTSCHEINE ZUFÄLLIG AUSFÜLLEN LASSEN (sofort in Datei)

        4: *  WETTSCHEINE LÖSCHEN
-------------------------------------------------------------------------
        L:    WETTSCHEIN-LISTEN

        I:    INFORMATIONEN
-------------------------------------------------------------------------
        #:    ZURÜCK ZUM HAUPTMENÜ
-------------------------------------------------------------------------
                                          Achtung! Die mit * markierten
                                          Menüpunkte sind nur für System-
Auswahl mit ↑↓ <↵                         Operateure erlaubt!
```

Der Bildschirm-Wettschein wurde dem tatsächlichen Wettschein der Trabrennbahn nachempfunden:

```
                          W E T T S C H E I N          MENÜ3-2

Datum    : 13.03.89
-------------------------------------
Rennen   : 1 2 3 4 5 6 7 8 9 10        RENNENNUMMER : 05
-------------------------------------
Wettart : S      P      KE     GE     WETTART      : s
-------------------------------------
1.Platz : 1 2 3 4 5 6 7 8              1.Platz : 3
2.Platz : 1 2 3 4 5 6 7 8
3.Platz : 1 2 3 4 5 6 7 8
-------------------------------------
DM       :  2.50    5.00    7.50      EINSATZ :   5.00 DM
           10.00   20.00   50.00
          100.00  200.00  500.00
-------------------------------------
```

Dieses Modul soll hier außerdem als Beispiel einer Modulbeschreibung herangezogen wer-den:

MODULSPEZIFIKATION

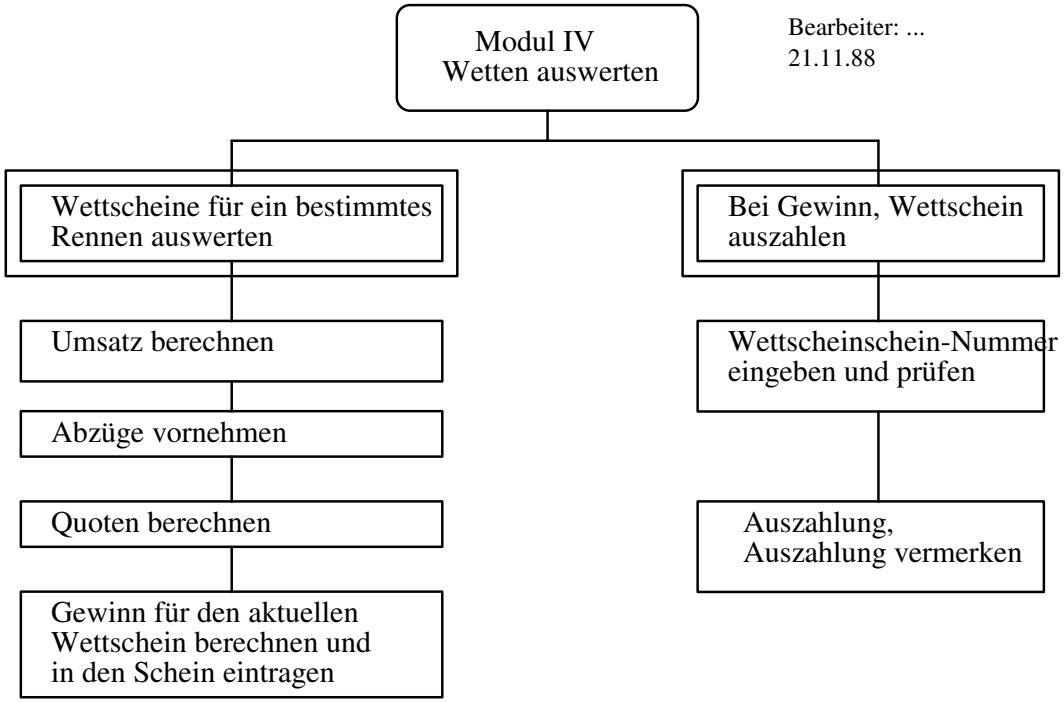
Modulname Wetten annehmen (Modul 3 des Programmsystems)
Letztes Update / Autor: 28.11.88, Gruppe 3
Modulfunktionen: Dieses Modul ermöglicht das Ausfüllen eines Wettscheines von Hand oder per Zufall. Die Eingaben erfolgen in eine Wettscheinmaske. Der Wettschein wird abgespeichert. Eine weitere Option erzeugt gleich 10 zufällige Wettscheine auf einmaDie Wettscheindatei kann ausgegeben werden.
Schnittstellen (Import): Diverse Prozeduren aus den allgemein verwendbaren Modulen Crt, Ino_u Balk_u, Bild_u und dem programmspezifischen Modul t_hi_u **Schnittstellen (Export):** PROCEDURE Wetten_abschließen (renntag:STRING8; renntag_feld: feld_of_renntag; VAR datei:file_of_wettschein; VAR schein_anzahl: INTEGER); PROCEDURE scheine_datei_aus; (ohne Parameter) **Datentypen, Datenstrukturen** Diverse aus dem programmspezifischen Datenmodul t_ds_u
Aufrufbeispiel: wetten_abschließen(renntag,renntag_feld, wett_datei,schein_anzahl)
Sonstiges:

Schließlich legte die Gruppe einen Algorithmus für das Abgeben eines Wettscheines vor:

Modul III: Teilalgorithmus "Wettscheine ausfüllen" 21.11.94

Verkauf eröffnen	
solange noch Wettscheine ausgefüllt werden sollen	
	Wettschein auf den Bildschirm bringen
	Datum und Rennen-Nummer und prüfen
	Weitere Wettscheindaten eingeben und prüfen (Art der Wette, Einsatz)
	Quittung erstellen
	Wettschein in die Wettscheindatei übernehmen
Verkauf schließen	

MODUL 4: WETTEN AUSWERTEN



Für jedes Modul sind Testdaten bereitzustellen.

Am Ende der Entwurfsphase kann nun die Checkliste von Kapitel 2.4 eingesetzt werden. Falls keine Nachbesserungen nötig sind, kann die Phase beendet werden. Es wird aber erneut darauf hingewiesen, daß aus der nun folgenden Programmierphase heraus möglicherweise Änderungen an den Entwürfen notwendig werden!

Auf die Beschreibung weiterer Detailalgorithmen wird verzichtet. Sie spiegeln sich im Quellcode des Programms (auf der Diskette SOFTENGI) wider.

3.4 Modulprogrammierung - Modultests

Die vollständigen Modulprogramme finden sich auf der Diskette unter den Namen

T_DS_U.PAS bzw. T_DS_U.TPU, Datenmodul (Trabrennen, Datenstruktur, Unit)
T_M1_U.PAS bzw. T_M1_U.TPU, Modul 1, Rennen initialisieren
T_M2_U.PAS bzw. T_M2_U.TPU, Modul 2, Rennen starten
T_M3_U.PAS bzw. T_M3_U.TPU, Modul 3, Wettscheine abgeben
T_M4_U.PAS bzw. T_M4_U.TPU, Modul 4, Wettscheine auswerten
T_HI_U.PAS bzw. T_M4_U.TPU, Hilfsmodul mit mehrfach benötigten Prozeduren
T_HP0.PAS bzw. T_HP0.EXE, Vorläufiges Hauptprogramm

Diese Module wurden getrennt getestet, ggf. wurden kleine Testprogramme konstruiert (dummies). Die Testdaten mußten dazu geeignet ausgewählt werden (siehe Kapitel 2.2.9, Testmethoden). Da später zahlreiche Daten aus den Dateien für die Renntage/ Rennen und für die Wettscheine kommen und dorthin übergeben werden, mußten einige Datensätze auf elementarere Weise bereitgestellt werden. So stehen zum Beispiel erst mit der Programmierung von Modul 1 Daten (Pferde, Renntage, Rennen) für die Wettscheine zur Verfügung.

Datenmodul

```
UNIT t_ds_u;
INTERFACE
CONST anz=40;
TYPE string8    = STRING[8];
   einlauftyp   = ARRAY [1..8] OF BYTE;
TYPE rennen = RECORD
          tag                : STRING8;
          rennen_nummer      : BYTE;
          kommentar          : ARRAY[1..3] OF STRING[40];
          anzahl_pferde      : BYTE;
          pferde_namen       : ARRAY[1..8] OF STRING[12];
          einlauf            : einlauftyp;
          zustand      : (vorher,laeuft,beendet);
        end;
TYPE wettschein = RECORD
            tag              : string8;
            rennen_nummer    : BYTE;
            satz_nummer      : INTEGER;
            wettart          : (sieg,platz,kew,gew);
            platz1,platz2.
```

```
            platz3          : BYTE;
            einsatz         : REAL;
            schein_nummer   : INTEGER;
            gewinn          : REAL;
            ausgezahlt      : BOOLEAN;
          end;
TYPE menge_der_menuepunkte = SET OF CHAR;
  feld_of_renntag  = ARRAY [1..10] OF RECORD
            tag             : STRING8;
            rennen_nummer   : BYTE;
            satznummer      : INTEGER;
            stand           : 0..2;          {0: vorher, 1: läuft, 2: nachher }
          end;
  file_of_rennen      = FILE OF rennen;
  file_of_wettschein  = FILE OF wettschein;
  feld                = ARRAY[1..anz] OF rennen;
VAR
   optionen           : menge_der_menuepunkte;
   menuepunkt         : CHAR;
   renndatei          : file_of_rennen;
VAR renntag           : string8;
     tag_ok           : BOOLEAN;
  renntag_feld        : feld_of_renntag;
  trab_datei          : file_of_rennen;
  wett_datei          : file_of_wettschein;
IMPLEMENTATION

BEGIN
END.
```

Hinweis: Die vielen globalen Variablen ließen sich sicher teilweise vermeiden. Hierin würde beispielsweise eine Aufgabe der Wartung des Systems bestehen.

Modul HI - Hilfsprozeduren

```
UNIT T_HI_U;
INTERFACE
USES Ino_u, Crt, Bild_u, t_ds_u;
PROCEDURE menue_freie_maske
       (dateiname: STRING; zu_x,zu_y: INTEGER;
        optionen : menge_der_menuepunkte; var menuepunkt: char);
{"dateiname" ist der der freien Maske,zu_x,zu_y ist die Cursorposition, an der die
 Eingabe erfolgt,"optionen"  ist die Menge der auswählbaren Menüpunkte,
menuepunkt ist der ausgewählte Menüpunkt, der weitergereicht wird.}
```

```
PROCEDURE menue_freie_maske_ohne_echo
      (dateiname: STRING; zu_x,zu_y: INTEGER;
       optionen : menge_der_menuepunkte; var menuepunkt: char);

PROCEDURE datei_check_1 (VAR datei: file_of_rennen);
PROCEDURE datei_check_2 (VAR datei: file_of_wettschein);
PROCEDURE renntag_datum(VAR renntag: string8);
PROCEDURE vorhandene_daten_nennen(VAR datei: file_of_rennen);
PROCEDURE renntag_in_feld( VAR renntag       : string8;
                           VAR tag_ok        : boolean;
                           VAR datei         : file_of_rennen;
                           VAR renntag_feld  : feld_of_renntag);
PROCEDURE passwort(var pass:BOOLEAN);
IMPLEMENTATION
```

Dieses Modul benötigt noch die Hilfsdatei TRABRAND.MAS

Modul 1: Renntag initialisieren

```
UNIT t_m1_u;
INTERFACE
USES Balk_u; Ino_u, Crt, t_ds_u, t_hi_u, Bild_U;
VAR dateiname: string;

PROCEDURE trab1_dat_liste;
PROCEDURE menue_1_maske(var datei: file_of_rennen);

IMPLEMENTATION
```

Dieses Modul benötigt noch die Hilfsdateien MENÜ1-2.MAS; MENÜ1-1.MAS und MENÜ1-1.BAL

Modul 2 : Rennen durchführen

```
UNIT t_m2_u;
INTERFACE
USES Crt, Bild_u,t_ds_u;

PROCEDURE rennen_durchfuehren (renntag: string8);

IMPLEMENTATION
```

Dieses Modul benötigt noch die Hilfsdatei MENÜ2-1.MAS

Modul 3 : Wetten annehmen

MODULSPEZIFIKATION

Modulname WETTEN ANNEHMEN (T_M3_U, Modul 3 des Programmsystems)

Letztes Update / Autor: 28.11.88, Gruppe 3

Modulfunktionen: Dieses Modul ermöglicht das Ausfüllen eines Wettscheines von Hand oder per Zufall. Die Eingaben erfolgen in eine Wettscheinmaske. Der Wettschein wird abgespeichert. Eine weitere Option erzeugt gleich 10 zufällige Wettscheine auf einmal. Die Wettscheindatei kann ausgegeben werden.

Schnittstellen (Import/Export)
Benutze Module: Diverse Prozeduren aus den allgemein verwendbaren Modulen Crt, Ino_u, Balk_u, Bild_u und dem programmspezifischen Modul t_hi_u, Datentypen aus Datenmodul.
Benutzte globale Variable: Keine

Prozeduren

Wetten_abschließen (renntag:STRING8;	Eingabeparameter, z.B. 13.08.94
renntag_feld: feld_of_renntag;	Eingabeparameter, Renndaten (ein Renntag)
VAR datei:file_of_wettschein;	Ausgabeparameter, Datei mit Wettscheinen
VAR schein_anzahl);	Anzahl der eingegebenen Scheine
scheine_datei_aus;	keine Parameter

Sonstiges:

```
UNIT t_m3_u;
{Dieses Modul M3 ermöglicht die Wettschein-Eingabe über eine Wettschein-Maske und
das Abspeichern des Wettscheins. }
INTERFACE
USES  t_ds_u,          (* Datenmodul von "Trabrennen"*)
      t_hi_u,          (* mehrfach verwendete Hilfsprozeduren von Trabrennen*)
     Ino_u,            (* Absicherung Zahleneingabe, Stringeingabe usw.*)
      Balk_u;          (* Menübalken-Unit *)
      Bild_u,          (* Maskenverwendung *)
      Crt;             (* Turbo-Pascal-Unit *)

PROCEDURE wetten_abschliessen  (renntag:string8; renntag_feld: feld_of_renntag;
                        VAR datei: file_of_wettschein; VAR schein_anzahl: INTEGER);
PROCEDURE scheine_datei_aus;

IMPLEMENTATION
```

Dieses Modul benötigt noch die Hilfsdateien MENÜ3-2.MAS; MENÜ3-1.MAS und MENÜ3-1.BAL

Modul 4 : Wettscheine auswerten

```
UNIT t_m4_u;
{ Unit für Modul 4, Wetten auswerten und Gewinne auszahlen }
INTERFACE
USES Crt,Ino_u, Bild_u, Balk_u,t_ds_u, t_hi_u;

PROCEDURE wetten_auswerten(    VAR wetten  : file_of_wettschein;
                               VAR renntage: file_of_rennen);

IMPLEMENTATION
```

Dieses Modul benötigt noch die Hilfsdatei MENÜ4-1.MAS

Modul 0 : "Film" zum Projekt "Trabrennbahn"

```
UNIT t_m0_u;
INTERFACE
        USES Crt,Bild_u,Balk_u;
        PROCEDURE trabfilm;
IMPLEMENTATION
```

Dieses Modul benötigt noch die Hilfsdateien
MENÜ1-1.MAS; MENÜ1-2.MAS; MENÜ2-1.MAS; MENÜ3-1.MAS;
MENÜ3-2.MAS;
TRAB-B1.MAS; TRAB-B2.MAS; TRAB-B3.MAS; TRAB-B4.MAS;
TRAB-B5.MAS; TRAB-B02.MAS und TRAB-HM.BAL.

Das vorläufige Hauptprogramm T_HP0.PAS

```
PROGRAM trabrennen;
USES Crt, Bild_u;
VAR weiter: CHAR;
(*-------------------------------------------------------------------------*)
PROCEDURE hauptmenue;
VAR fall: CHAR;
BEGIN   Clrscr;
  dateiname:='trab-hm0.mas';        {gespeicherte Hauptmenümaske }
  Textbild_aus(dateiname);
  REPEAT
```

```
  GOTOXY(64,22);   fall:=READKEY;
  UNTIL fall IN ['1','2','3','4','i','I','#'];      { # ist alt 35 }
  Clrscr;

  CASE fall OF
  '1': BEGIN
         WRITELN('Modul "Renntage organisieren", Test ok');
         READLN;
       END;
  '2': BEGIN
         WRITELN('Modul "Rennen starten", Test ok');
         READLN;
       END;
  '3': BEGIN
         WRITELN('Modul "Wetten annehmen", Test ok');
         READLN;
       END;
  '4': BEGIN
         WRITELN('Modul "Gewinne auszahlen", Test ok');
         READLN;
       END;
  'i','I': BEGIN
             WRITELN('Modul "Informationen", Test ok');
             READLN;
           END;
  '#': BEGIN
         WRITELN('Modul "Ende", Test ok');
         READLN;
         EXIT;
       END;
  END;    {von CASE}
END;      { Prozedur Hauptmenue beendet }
(*-------------------------------------------------------------------------*)
BEGIN
  weiter:='j';
  WHILE weiter IN ['j','J']  DO
  BEGIN
    hauptmenue; CLRSCR;
    write('Weiter  (j,n) ? '); readln(weiter);
  END;
END.
```

Zum Abschluß der Programmierphase lagen zahlreiche Dateien unterschiedlicher Funktion vor:

Quellprogramme	*.pas	
Compilierte Programme	*.exe	
Menühilfen	*.inf	
Dokumentationsteile	*.txt	
Daten: Wettscheine,Rennen	*.dat	
Eingabemasken	*.mas	erstellt mit dem Tool BILD_DEF (siehe Kap.2.2.6)
Menübalken	*.bal	erstellt mit dem Tool MENUE (siehe Kap.2.2.6)

Hilfen für die Menübenutzung

MENÜ-0 .INF	MENÜ1-1 . INF	MENÜ3-1 .INF	MENÜ4-1 .INF

Menümasken und andere Eingabemasken sowie Menübalken

H-MENUE .MAS	MENÜ1-1 .MAS	MENÜ1-2 .MAS	MENÜ2-1 . MAS
MENÜ3-1 .MAS	MENÜ3-2 .MAS	MENÜ4-1 .MAS	
RENNBAH1 .MAS	TRABRAND.MAS	WETTSCH.MAS	
TRAB-B01 .MAS	TRAB-B02 .MAS	TRAB-B1 .MAS	TRAB-B 2 .MAS
TRAB-B3 .MAS	TRAB-B4 .MAS	TRAB-B5 .MAS	TRAB-HM0 .MAS
MENUE1-1 .BAL	MENUE3-1 .BAL		

Dokumentationsteile aus der Problemanalyse und für die Anforderungsdefinition

DOK-PA3 .TXT	DOK-PA2 .TXT	DOK-PA4 .TXT	DOK-AD1 .TXT
DOK-AD2 .TXT	DOK-PA1 .TXT		

Dokumentationsteile für das Pflichtenheft

DOK-PF1 .TXT	DOK-PF2 .TXT	DOK-PF3 .TXT	DOK-PF4 .TXT

Dokumentationsteile aus der Entwurfsphase

ENTW-M11 .TXT	ENTW-M21 .TXT	
ENTW-M31 .TXT	ENTW-M41 .TXT	ENTWU-0 . TXT

Gespeicherte Wettscheine und Rennen, Quellprogramme, Hauptprogramme

SCHEINE.DAT	SCHEINE2 DAT	TRAB1.DAT TRAB2.DAT
T_DS_U .PAS	T_DS_U .TPU	Datenmodul
T_HI_U .PAS	T_HI_U .TPU	Modul mit problemspezifischen Hilfen
T_M1_U .PAS	T_M1_U .TPU	Aus der Problemzerlegung heraus
T_M2_U .PAS	T_M2_U .TPU	entstandene Programmmodule
T_M3_U .PAS	T_M3_U .TPU	
T_M4_U .PAS	T_M4_U .TPU	
T_M0_U .PAS	T_M0_U .TPU	Modul zum "Film"
T_HP0.PAS	T_HP0.EXE	Vorläufiges Hauptprogramm
T1234.PAS	T1234.EXE	Endgültiges Hauptprogramm

3.5 Systemintegration

Nachdem nun jedes Team nachgewiesen hat, daß das von ihm bearbeitete Modul richtig arbeitet, sollte man meinen, daß die Integration der Module zu einem Gesamtsystem keine besonderen Schwierigkeiten macht. Leider ist diese Integration häufig nicht so problemlos! Über die Vorgehensweise kann man sich in den Kapiteln 2.1.1 und 2.4 informieren.

In jedem Fall ist es günstig, sich noch einmal einen Überblick über die Schnittstellen zwischen den verwendeten Modulen zu verschaffen. In dem Überblick wird die Turbo-Pascal-Unit CRT nicht berücksichtigt. Auch das Modul T_M0_U wird hier nicht betrachet, da der Film schon ohne Berücksichtigung der anderen Module erstellt wurde. Für ihn sind insbesondere die Dateien *.MAS wichtig. Diese ruft er als Bildschirmmasken jeweils als Ganzes auf den Bildschirm.

			Projektspezifische Module				Projektunabhängige Module		
Modul-name	***t_ds_u***	***t_hi_u***	***t_m1_u***	***t_m2_u***	***t_m3_u***	***t_m4_u***	Ino_u	Balk_u	Bild_u
t_ds_u Daten-modul.									
t_hi_u Hilfs-modul	X						X		X
t_m1_u Renntag initialis.	X	X					X	X	X
t_m2_u Rennen	X								X
t_m3_u Wettsch abgeben	X	X					X	X	X
t_m4_u Wetten auswer-ten	X	X					X	X	X
Baustein Ino_u									
Baustein Balk_u									
Baustein Bild_u									

X: Das in der 1.Spalte der Tabelle stehende Modul benutzt den in der 1.Zeile stehenden Modul.
Beispiel: t_m1_u benutzt unter anderem t_ds_u (und noch vier weitere Module). Es wird unterschieden zwischen den **projektspezifischen Modulen** (fett und kursiv gedruckt) und den **projektunabhängigen Modulen**, die **als wiederverwendbare Bausteine** zur Verfü-gung stehen und um deren Test man sich nicht kümmern mußte (siehe Kapitel 2.2.7). Mit diesen Bausteinen brauchen wir uns hier nicht zu beschäftigen.

Auswertung: Zunächst ist beruhigend zu sehen, daß sich die projektspezifischen Module nicht gegenseitig aufrufen. Das zeigt, daß die gewählte Zerlegung des Problems vernünftig war. Häufig benutzt werden das Datenmodul und das Hilfsmodul (diese hätte man auch zusammenführen können zu einem abstrakten Datentyp). Von Seiten der Modulzerlegung sind also keine Probleme zu erwarten.

Wir analysieren nun noch einmal die in der Entwurfsspezifikation in den Modulen angegebenen Prozeduren und ihre Parameter, insbesondere die Variablenparameter. Wir erkennen, daß wir als Schnittstellen insbesondere die beiden Dateien (Datei mit den Renn-tage-Daten, Datei mit den Wettscheinen) sowie das Renntagsdatum zu beachten haben.

Die Systemintegration erfolgt nun schrittweise so:

1) Einklinken von Modul 1 (Renntag initialisieren, T_M1_U.TPU) in das vorläufige Hauptprogramm T_HP0.PAS. Neues vorläufiges Hauptprogramm ist T_HP1.PAS. Testen!

2) Einklinken von Modul 2 (Rennen durchführen, T_M2_U.TPU) in das vorläufige Hauptprogramm T_HP1.PAS. Neues vorläufiges Hauptprogramm ist T_HP2.PAS. Testen!

3) Einklinken von Modul 3 (Wettscheine annehmen, T_M3_U.TPU) in das vorläufige Hauptprogramm T_HP2.PAS. Neues vorläufiges Hauptprogramm ist T_HP3.PAS. Testen!

4) Einklinken von Modul 4 (Wettscheine auswerten, T_M4_U.TPU) in das vorläufige Hauptprogramm T_HP3.PAS. Neues vorläufiges Hauptprogramm ist T_HP4.PAS. Testen!

5) Einklinken von Modul 0 (Film, T_M0_U.TPU) in das vorläufige Hauptprogramm T_HP4.PAS. **Das endgültige Hauptprogramm ist T1234.PAS.** Testen!

Das Hauptprogramm T1234 hat schließlich folgende Struktur:

```
PROGRAM trabrennen;

USES Ino_u, Balk_u, Bild_u, Crt,  (* projektunabhängige Module*)
    t_ds_u,      (* Datenstruktur *)
    t_hi_u,      (* Hilfsprozeduren *)
    t_m1_u,      (* Modul 1, Renntag initialisieren, integriert 15.3.*)
    t_m2_u,      (* Modul 2, Rennen durchführen, integriert 3.4. *)
    t_m3_u,      (* Modul 3, Wetten abschließen, integriert 18.3*)
    t_m4_u,      (* Modul 4, Wetten auswerten, Gewinn auszahlen, integriert 24.4.*)
    t_m0_u       (* Modul 0, Demo-Film, integriert 24.4.*)
VAR schein_anzahl: INTEGER;
   neuer_renntag: char;

PROCEDURE renntag_waehlen;
BEGIN
    REPEAT
{P}     vorhandene_daten_nennen(trab_datei);
{P}     renntag_datum(renntag);
{P}     renntag_in_feld(renntag, tag_ok, trab_datei, renntag_feld);
    UNTIL tag_ok;
END;

PROCEDURE hauptmenue(VAR option: INTEGER);
VAR pass : BOOLEAN;
   c : char;
BEGIN
  clrscr;
  rahmen_zeichnen2(6,2,71,25,'—','|',magenta,magenta);
  textcolor(lightgreen);
  gotoxy(14,5);write('T R A B R E N N B A H N    S C H Ö N E B E R G');
  textcolor(7);
  gotoxy(10,24);write('Auswahl mit ',#24,' ',#25,' ↵');
  gotoxy(10,3);write('Hauptmen□ ');
  dateiname:='trab-hm.bal';     (* das spezielle Menü *)
  menue_wahl(lightgreen,magenta,dateiname,option);
```

```
  CASE option of
    1:    menue_1_maske(trab_datei);
    2:    BEGIN
            renntag_waehlen(renntag);
            rennen_durchfuehren (renntag);
          END;
    3:    BEGIN
            renntag_waehlen(renntag);
           wetten_abschliessen(renntag,renntag_feld,
                        wett_datei,schein_anzahl);
         END;
    4:    BEGIN
            renntag_waehlen(renntag);
            wetten_auswerten(wett_datei,trab_datei);
          END;
    5:    BEGIN
            output_textdatei(1,1,80,25,'menü-0.inf','');
            trabfilm;
          END;
    6:    BEGIN
                  writeln('Hilfe noch nicht integriert.'); c:=readkey;
          END;
    7:    ;
  END;          {von CASE}
END;

BEGIN
  assign(trab_datei,'trab1.dat');
  assign(wett_datei,'scheine.dat');
  {P}    datei_check_1(trab_datei);
  REPEAT
    {P}   hauptmenue(option);
  UNTIL option = 7;
  Clrscr;
END.
```

3.6 Installierung des Systems

Betrieb und Wartung

Wir fassen hier zwei Phasen zusammen, da hierzu in diesem Zusammenhang nur wenig zu sagen ist.

INSTALLIERUNG

Bei der Installierung wird das Produkt in die Systemumgebung und die spezielle betriebliche Organisation des Auftraggebers eingebettet. Der Auftraggeber führt einen Abnahmetest durch. Außerdem erfolgt eine Benutzerschulung durch den Hersteller. Dabei werden auch Benutzerhandbuch und Wartungshandbuch übergeben. - Da in diesem Fall das System am Herstellungsplatz benutzt werden sollte, waren derartige Arbeiten nicht nötig.

BETRIEB UND WARTUNG

Das fertige System wurde später im Unterricht zur Informationstechnischen Grundbildung ITG in Klasse 9 eingesetzt. Für die Benutzung standen verschiedene Dokumentationsteile zur Verfügung, insbesondere wurden die integrierten Hilfen benutzt. Dabei wurden einige Systemschwächen deutlich, so daß jetzt geplant ist, das System "Trabrenn-bahn" in einem neuen Informatik-Projekt aufzugreifen (1994/95) und es zu "warten". Beispielsweise könnten die im folgenden genannten Wartungsarbeiten vorgenommen wer-den.

EINIGE WARTUNGSARBEITEN

- Verbesserung der Menüführung, etwa durch Mausklick
- Benutzung einer grafischen Oberfläche bei der Darstellung eines Rennens
- Verbesserung der Datumseingabe
- Auswahl eines vorhandenen Rennens durch Mausklick

Auf die Wartung von Software (Reengineering) werden wir in Kapitel 4 zurückkommen.

4. Softwarewartung - Reengineering

4.1 Grundlagen

Unter der Wartung eines Softwareprodukts versteht man u.a.

- die Beseitigung von Fehlern, die möglicherweise bei der Benutzung des Produkts aufgetreten sind,
- Änderungen am Programm, die aufgrund neuer Anforderungen an den Anwendungsbereich notwendig geworden sind.

Diese Gründe sind es auch, die bei vielen professionellen Produkten auftreten und zu immer neuen Versionen eines Programms führen. Sie kennen dieses Phänomen wahrscheinlich für die verschiedenen Versionen von Turbo-PASCAL (3,4,5,6,7). Möglicherweise haben Sie auch schon bei eigenen Programmen gemerkt, daß sie bald nicht mehr mit der vorliegenden Fassung zufrieden sind.

Software-Wartung nimmt also in der Praxis einen großen Stellenwert ein. Sie erfordert häufig einen erheblichen Arbeitsaufwand und kostet entsprechend viel. Um Aufwand und Kosten zu minimieren, muß man sich Gedanken machen, wie man eine derartige Wartung am besten durchführt, sofern man selbst die Möglichkeit dazu hat. Für diesen Fall muß einem selbstverständlich die Programmquelle vorliegen.

Wenn ein Projekt zur Softwarewartung begonnen wird, ist es unerläßlich, sich vorher klarzumachen, wie Software überhaupt hergestellt wird. Die umfangreichen Methoden hierzu sind in den Kapiteln 1-3 des Buches dargetellt. Als Grundlage auch für das Reengineering kann der bekannte Software-Life-Cycle dienen, angereichert mit den Methoden des Prototyping, siehe Kapitel 2.1.1 und 2.1.2.

Die uns hier interessierende Phase der Wartung des Softwareprodukts tritt in der Abbildung zum Software-Life-Cycle unter der Nummer (7) auf. Man kann sich leicht vorstellen, daß zumindest ein größerer Änderungsauftrag wieder einen Ablauf einleitet, der durch die Abfolge (1),(2),..., aber auch durch zusätzliche Überlegungen bestimmt ist. **Denn nun setzt die Arbeit auf eine fertige Systemversion auf, die bei der Systemerweiterung mehr oder weniger stark zu berücksichtigen ist.** Die Zeitdauer der Phasen wird daher sicher anders sein, als bei der erstmaligen Systementwicklung und auch die Vorgehensweise wird sich teilweise davon unterscheiden,

Wir wollen bei unseren Überlegungen zur Softwarewartung davon ausgehen, daß der Anwender der Software zu einer Softwarefirma mit dem Wunsch kommt, Änderungen an dem Softwareprodukt vorzunehmen, und daß die Mitarbeiter der Softwarefirma dieses Softwareprodukt noch nicht kennen. - Eine vergleichbare Situation liegt vor, wenn

Schüler eines Informatikkurses ein Softwareprodukt warten sollen, das mehrere Jahre zuvor von anderen Schülern erstellt wurde.

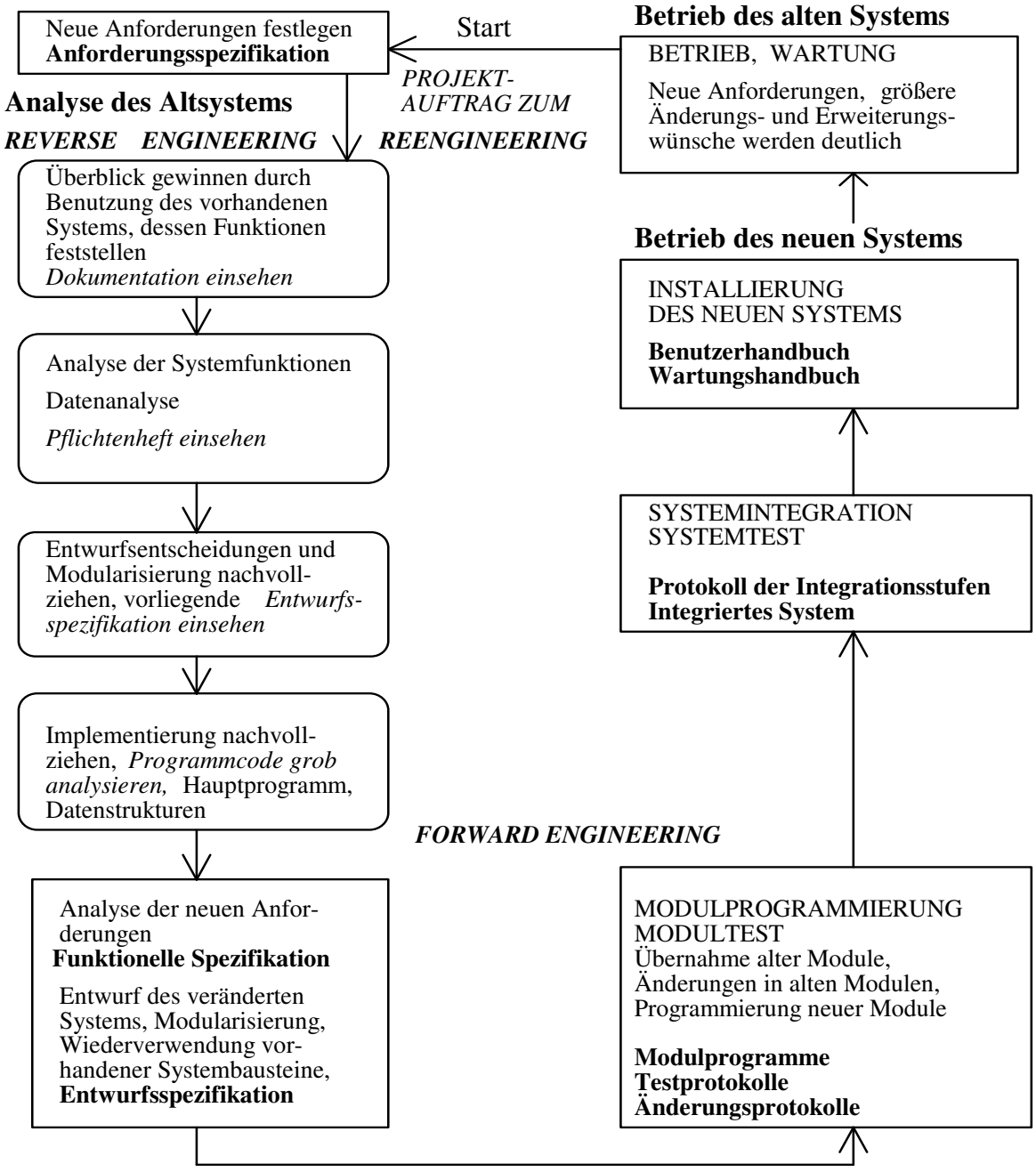


Abb. 4.1.a: Ablauf eines Reengineering-Projekts
Reengineering = Reverse Engineering + Forward Engineering

Wie ist die Vorgehensweise bei der Software-Wartung?

Abbildung 4.1.a zeigt, welche Vorgänge durch den Wunsch nach einer umfangreicheren Änderung eines Softwareproduktes in Gang gesetzt werden. Auch hier ist wieder zu berücksichtigen, daß es immer wieder Rückläufe zu vorhergehenden Phasen geben wird. Außerdem ist die Art des Durchlaufs stark von der vorliegenden Software, der Güte ihrer Dokumentation und dem Umfang der Änderungswünsche abhängig. Daher lassen sich keine eindeutigen Regeln aufstellen.

(1) Die Wartung eines Softwareprodukts beginnt in der Regel mit der **Analyse der vorliegenden Software**: Programmbenutzung, Funktionsumfang erkennen, Dokumentation (Handbücher) inspizieren, Datenstruktur verstehen, Schnittstellen erkennen, Quellprogramm soweit benötigt verstehen, änderungsrelevante Programmteile ausfindig machen.

(2) Nun erfolgt die **Veränderung des Systems gemäß den neuen Anforderungen.** Im folgenden werden einige solcher Anforderungen genannt. Diese treten in der Regel kombiniert miteinander auf und führen zu einer neuen Programmversion.

(3) Ein Wartungsdurchgang endet mit der **Implementierung des überarbeiteten Systems** in einer neuen Form oder einer neuen Umgebung. Dazu gehören auch die notwendigen Dokumentationen und vor allem die Schulung der Mitarbeiter.

Datenanalyse

Das Reengineering darf nicht losgelöst von der jeweiligen Datenbasis gesehen werden. Immer dann, wenn das Softwaresystem auch auf zu erweiternden oder neuen Datenbeständen arbeiten soll, kommt der Analyse der verwendeten Daten eine besondere Bedeutung zu. In diesem Fall kann nach Abbildung 4.b vorgegangen werden.

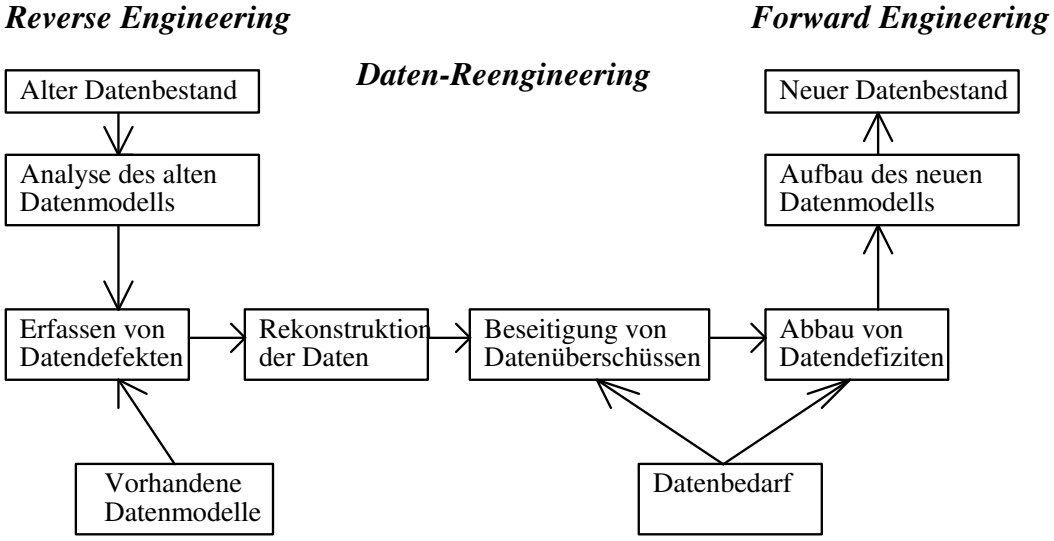


Abb.4.1.b: Daten-Reengineering

Anlässe für einen Wartungsfall

An solchen Anlässen mangelt es nicht! Wie bei einem völlig neuen Projekt, sind aber auch hier Warnungen abgebracht, die sich zum einen auf die Zeiteinschätzung, zum anderen auf den Schwierigkeitsgrad beziehen. Insbesondere sollte man den Aufwand für die Analyse des Altsystems nicht unterschätzen. Mancher Änderungswunsch führt zu erheblichen Schnittstellenproblemen und Seiteneffekten. Beachten Sie auch für das Reengineering die Ausführungen in Kapitel 2.4 über die Projektkontrolle.

Im folgenden werden Anlässe für Wartungsarbeiten zusammengestellt, wie sie auch in der Schule immer wieder auftreten können:

a) Beseitigung von Fehlern; dabei können die Fehler ganz unterschiedliche Ursachen haben: Fehlerhafte Analyse, unvollständige oder widersprüchliche funktionale Spezifikation, ungeeignete oder fehlerhafte Modularisierung, Programmierfehler,...)
b) Erweiterung des Funktionsumfangs
- Hinzufügen neuer Module
- Ersetzen von Modulen durch andere mit vergrößertem (verkleinertem) Funktionsumfang
- Einfügen graphischer Elemente

c) Verbesserung der Benutzerfreundlichkeit
- Veränderung der Benutzeroberfläche
- integrierte Hilfen, Einrichten von Hilfedateien
- Einfügen graphischer Elemente
- Umstellung von Handbedienung auf Mausbedienung

d) Nachprogrammierung eines bekannten Systems mit gesteigerten Anforderungen
- an die interne Programmstruktur (Modularisierung)
- unter Verwendung von Standardmodulen

e) Verbesserung und Ergänzung der Dokumentation (Handbücher)
f) Standardisierung
g) Erhöhung der Sicherheit und Zuverlässigkeit
h) Steigerung der Effizienz
i) Verbesserung der Portabilität
- Umstellung auf Betrieb im Netz

j) Beseitigung überflüssigen Codes, bessere Strukturierung des Codes, Verbesserung der Parametrisierung
k) Ergänzung von Kommentaren im Quellcode oder während des Programmlaufs, Wahl eines geeigneten Sprachniveaus (für die Benutzer, für die Systembetreuer)
n) Wartungsarbeiten unter Beachtung der Wiederverwendbarkeit
o) Einbettung in neue Umgebungen

Beispiel eines Wartungsfalls

Von einem älteren Informatikkurs ist ein Softwareprodukt für den Einsatz im Schulfach Erdkunde konstruiert worden. Das System wurde inzwischen von einem Fachlehrer im Unterricht benutzt. Dabei sind einige Schwächen in der Benutzerführung, Dokumentationsfehler und Programmierfehler festgestellt und notiert worden.
Der Fachlehrer wendet sich mit diesen Unterlagen an den Informatiklehrer und bittet um Nachbesserung. Der Informatikkurs soll diese Aufgabe im Rahmen von Projektarbeit bearbeiten. - In diesem Wartungsfall werden vermutlich folgende oben genannte Punkte relevant: a), c), e), g). Der Informatikkurs muß bei der Realisierung der Wünsche alle Phasen der Projektarbeit von der Systemanalyse bis zur neuen einsatzfähigen Version abarbeiten, jedoch in sehr eingeschränktem Umfang, da im wesentlichem auf das vorliegende System aufzubauen ist.

Wiederverwendbarkeit von Komponenten eines Software-Systems

Vorhandene Softwaresysteme enthalten eine Menge von Wissen, das bei der Vernichtung des Systems oder bei Systemänderungen leicht verloren gehen kann! Man sollte sich daher darum bemühen, das in der Software abgebildete Anwendungswissen und das Know-how weiterzunutzen, siehe Abbildung 4.b.

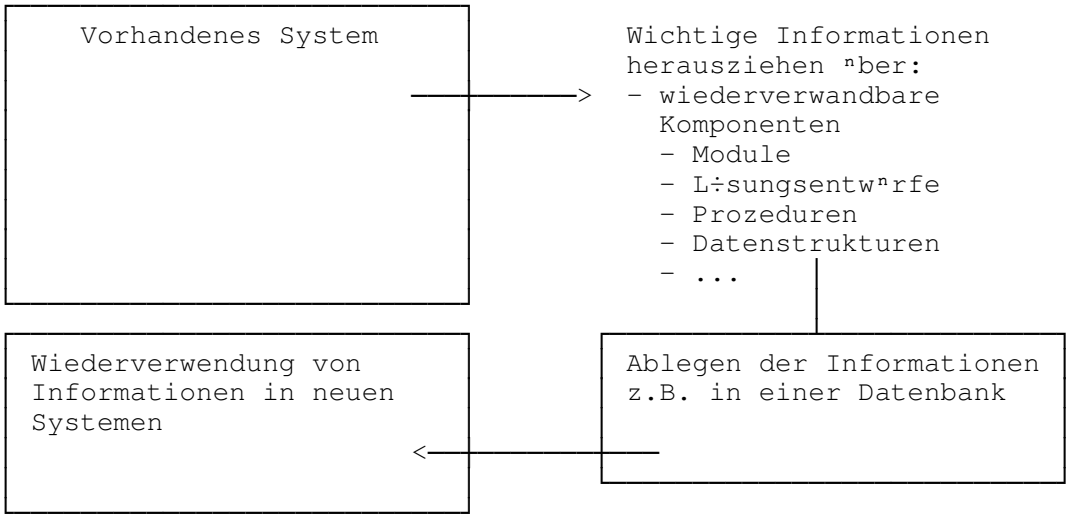


Abb. 4.1.c: Weiterverwendung von in einem Softwaresystem enthaltenen Wissen

Hinweise zu ausgewählten Phasen des Reengineering

Programmanalyse

Selbstverständlich kommt es bei der angewandten Analysemethode auf die Art der vorher verfolgten Programmierstrategie an. Ist das Programm gut strukturiert, in welchem Umfang gibt es erläuternde Kommentare, wie lang ist das Programm? In der Regel läßt sich jedoch sagen: Das häufig zu beobachtende Verfahren, ein vorgelegtes

längeres Programm linear von oben nach unten zu lesen und zu verstehen, ist i.a. wenig geeignet.
Vielmehr sollte man etwa so vorgehen:

- Analyse des Hauptprogramms
- Analyse der globalen Daten
- Analyse ausgewählter Prozeduren, die im Hauptprogramm aufgerufen werden

In jedem Fall sind vorhergehende Programmläufe nützlich, um sich mit der Funktionalität des Programms, die besonders in den Menüs und den Eingaben/Ausgaben deutlich wird, vertraut zu machen.

Um Zusammenhänge zwischen Programmteilen aufzuspüren, kann man sich Prozedurtabellen anfertigen, in die man die Prozeduraufrufe einträgt, siehe hierzu "**Cross-Referenzen**" im Glossar.

4.2 Beispielablauf eines Software-Wartung-Projekts

Wie bereits in Kapitel 3 angekündigt, wird hier über ein Reengineering-Projekt berichtet, das im Jahr 1994 mit Schülern der 13.Jahrgangsstufe durchgeführt wurde. Als Softwareprodukt stand das in Kapitel 3 dargestellte TRABRENNBAHN-System zur Verfügung. Die Vorgehensweise richtete sich im wesentlichen nach dem in 4.1 dargestellten Modell (Abb. 4.1.a).

Hinweise:
- Aus Platzgründen kann hier nur ein kurzer Abriß des Projektverlaufs gegeben werden.
- Bei der Beurteilung der Systemschwächen bedenken Sie bitte, daß das vorliegende Softwareprodukt in Schüler-Projektarbeit entstanden ist und vom Lehrer nicht verändert wurde. Daher treten mehr Schwächen auf, als sie professionale Software aufweisen würde. Allerdings ist es bei den vorliegenden Unterrichtsintentionen sogar erwünscht, daß solche Schwächen leicht feststellbar und zumindest einige von ihnen auch leicht zu beseitigen sind.

Reverse Engineering

Arbeitsaufträge 1 und 2
Benutzen Sie das TRABRENNBAHN-System mit folgender Zielsetzung
1. Vertrautmachen mit der Funktionalität des Systems

2. Notieren von Systemschwächen

Die Funktionen des Systems sind in Kapitel 3 beschrieben. Den Schülern lag außerdem der Text dieses Kapitels als Dokumentation vor!

Zur Feststellung der Systemschwächen wurden die sich in der Programmoberfläche widerspiegelnden 4 Module von 4 Gruppen bearbeitet. Dabei ergaben sich die folgenden Mängelisten (jeweils im Editor dokumentiert):

Menüpunkt 1: Rennen initialisieren

- Eingabe von Datensätzen: Keine Korrekturmöglichkeit von Tippfehlern nach Drüc??ken der Eingabetaste. Man gelangt sofort in das nächste Feld der Maske und kann nicht zurück. Es ist lediglich ein Neubeginn mit leerer Maske möglich, nachdem die Eingabemaske zuvor vollständig bearbeitet wurde.
- Bei den Zeichenketteneingaben werden keine Sonderzeichen zugelassen (ä, ß,,- usw.).
- Löschen von Datensätzen: Keine Absicherung gegen Tippfehler bei der Eingabe der zu löschenden Satznummer. Kein zusätzliches Darstellen des zu löschenden Datensatzes.
- Menüpunkte: Irreführende Beschriftung der Menüpunkte. Der Benutzer wird dazu verleitet, angegebene Buchstaben zu drücken, was jedoch keine Wirkung hat. Vielmehr wird ja mit dem Cursor ausgewählt.

Mögliche Abhilfen bei der Eingabe von Datensätzen:
- Bei Neubeginn die vorhergehenden Einträge anzeigen oder
- Neuprogrammierung (unter Beibehaltung der Maskentexte), so daß man in der Maske mit Hilfe der Cursortasten und der Tab-Taste hin- und herfahren kann.

Menüpunkt 2: Rennen starten

Der Ablauf des Rennens ist mit einer sehr primitiven Blockgrafik auf dem Textbildschirm dargestellt. Hier wäre eine Benutzung des Grafikbildschirms zu empfehlen.

Menüpunkt 3: Wetten annehmen

- Wenn man eine 1 eingeben will, muß man 01 eingeben.
- Beim Datum müssen die Punkte mit eingegeben werden.
- Beim Tippen sieht man nur die Nummern der Pferde, nicht die Namen.
- Beim Löschen eines Wettscheins wird dieser nicht vorher angezeigt.
- Man kann nicht alle Scheine zu einem Renntag löschen.

Menüpunkt 4: Gewinne auszahlen

- Die Liste der vorhandenen Renntage ist unübersichtlich.
- Die Datumseingabe erfolgt unnötigerweise doppelt.

- Bei Eingabe einer falschen Rennen-Nummer gelangt man sofort wieder ins Hauptmenü.
- Die Fehlermeldung ist blinkend und daher schlecht zu lesen.
- Bei ausführlicher Auswertung der Wettscheine ist kein Abbruch möglich.
- Die Informationen erhält man nur mit "I", nicht mit "i".

- *Von allen Gruppen wurden die häufigen Paßwortabfragen für die Nutzungsberechtigung verschiedener Menüpunkte besonders bemängelt .*

Forward Engineering

Arbeitsauftrag 3:
Notieren Sie die von Ihnen gewünschten Änderungen und schätzen Sie den Aufwand und die Priorität der Änderung ab. Benutzen Sie hierzu die vorgegebene Schablone.

Änderungsvorschlag
Bearbeiter Datum
Änderung in Modul / Prozedur
Art der Änderung
Grund
Aufwand (sehr gering, gering, mittel, hoch, sehr hoch)
Priorität (sehr gering, gering, mittel, hoch, sehr hoch)

Hinweis: Vor dem Beginn Entwurfs- und Programmierphase für die Änderungen wurde das eigentliche Reengineering-Projekt unterbrochen, um **Qualitätsanforderungen** an Software zu erarbeiten, siehe Kapitel 2.2.8.

Arbeitsauftrag 4
Führen Sie folgende Programmänderungen durch:
1. Ersetzen Sie die häufigen Paßwortabfragen für die Nutzungsberechtigung verschiedener Menüpunkte durch eine generelle Abfrage beim Programmstart.
2. Beim Löschen von Wettscheinen soll der jeweilige Schein vorher angezeigt werden.
3. Die Eingabe von Datensätzen für ein Pferderennen soll unter Beibehaltung des Maskenaufbaus so abgeändert werden, daß Eingabekorrekturen in den einzelnen Feldern durch Cursorwanderung möglich werden.
4. Die primitive Blockgrafik ist durch eine Simulation der Rennen auf dem Grafikbildschirm zu ersetzen!

5. Die Änderungen sind in Änderungsprotokollen zu dokumentieren.

Wichtige Hinweise:
Zunächst sind Entwürfe anzufertigen und die Schnittstellen und möglichen Seiteneffekte zu untersuchen. Erst dann kann zur Programmierung übergegangen werden. Wenn möglich, sind dabei Bausteine zu verwenden. Die Programmierung erfolgt nach den bekannten Grundsätzen für strukturierte Programme! Außerdem ist so wenig wie möglich in die vorhandene Programmstruktur einzugreifen - es soll also möglichst viel vom Altsystem übernommen werden!

Zunächst wurden die ersten drei Arbeitsaufträge von je einer Schülergruppe bearbeitet und erfolgreich abgeschlossen. Daraufhin erfolgte eine Integration zu einem neuen System (Hauptprogramm S1234.PAS). Sie finden das gesamte neue System unter den Dateinamen S_*.* auf der Diskette zu diesem Buch. Die Namen der Hilfsdateien (Menüs, Masken, Hilfetexte usw.) wurden gegenüber dem Altsystem nicht verändert.

Hinweis: Vor dem Beginn Grafikprogrammierung wurde das eigentliche Reengineering-Projekt unterbrochen, um grundlegende Kenntnisse über Grafikprogrammierung zu erarbeiten. Weitere Kenntnisse dazu wurden von den Schülern immer dann erworben, wenn es bei ihrer Arbeit vonnöten war.

Änderungsprotokoll Datum
18.10.94
Bearbeiter: Schüler A und B
Aufgabe: Wettscheine vor Löschvorgang erscheinen lassen und Kontrollabfrage einbauen

Unit	T_M3_U.PAS
Prozedur	Scheine_saetze_löschen
Art der Änderung	Ein Teil der Prozedur Scheine_Datei_Aus wurde übernommen und eine Kontrollabfrage wurde eingebaut
Globale Variablen	keine

Der nächste Änderungsschritt bestand darin, daß alle drei Schülergruppen die Aufgabe 4 getrennt voneinander bearbeiteten. Eine der entstandenen Grafik-Lösungen wurde dann in die obige neue Version integriert (auf der Diskette). Selbstverständlich wurden auch die anderen Lösungen erprobt. An dieser Stelle ist es sehr lehrreich zu sehen, wie man eine der drei Teillösungen ohne besondere Probleme durch eine andere Teillösung ersetzen kann (modular arbeiten!).

Mit Beendigung dieser Arbeiten konnte das neue System in Betrieb gehen. Das Reengineering-Projekt wurde einschließlich dem Erwerb von projektunabhängigen und projek-

tabhängigen Kenntnissen über Softwarequalität, Grafikprogrammierung und Methoden des Reengineering von Software in ca. 30 Schulstunden durchgeführt.

4.3 Weitere Software zum Zweck der Wartung

Für die Durchführung anderer Software-Wartungsprojekte oder auch für neue Projekte stehen folgende Produkte zur Verfügung:

(A) Das Spiel GOBANG:

Dieses Programm wird ausführlich beschrieben in dem *Dümmler-Buch*:
Software-Analyse im Informatik-Anfangsunterricht, 1994, und steht auf der Diskette GOBANG zur Verfügung. Die Quelldateien und alle Hilfsdateien sind auf der Diskette vorhanden. Bei der Benutzung des Spiels werden sich verschiedene Wünsche zu kleineren Änderungen ergeben. - Dieses Programm ist ebenfalls in Projektarbeit entstanden. Es kann in zwei Versionen gespielt werden, die sich durch die Art der Zugeingabe unterscheiden - Eingabe der gewünschten Spielfeldkoordinaten oder Eingabe durch Mausklick.

DIE SPIELREGELN

Gobang wird auf einem quadratischen Spielfeld mit z.B. 11 x 11 Kästchen von zwei Spielern gespielt. Spieler 1 spielt mit roten, Spieler 2 mit blauen Steinen. Die Spieler setzen abwechselnd jeweils einen ihrer Steine auf ein leeres Feld. Jeder der Spieler versucht, als erster eine Fünferreihe zu erzeugen bzw. seinen Gegenspieler dabei zu behindern. Eine Fünferreihe besteht aus 5 gleichfarbigen Steinen, die horizontal, vertikal oder diagonal nebeneinander angeordnet sind. - Wer zuerst eine Fünferreihe zustandebringt, hat gewonnen, und ein neues Spiel kann beginnen. Ein Unentschieden entsteht, wenn alle Felder besetzt sind, ohne daß ein Spieler 5 Zeichen nebeneinander plazieren konnte. - Der Reiz des Spieles erhöht sich, wenn schnell gespielt wird. Interessant ist auch die Organisation eines Turniers "Jeder gegen Jeden" oder im "Ko-System"!

(B) Das Räuber-Beute-System MAUS-WIESEL

Literatur und Diskette siehe (A). Die Funktionen des Systems werden deutlich bei Aufruf des Hauptmenüs:

SIMULATION EINER RÄUBER-BEUTE-BEZIEHUNG
Ein Programm für die ITG und den Informatikunterricht

1 SIMULATION - Mäuse und Wiesel
2 Aufgaben zu 1
3 Informationen zum Räuber-Beute-Modell
4 Hilfe zur Programmbenutzung

```
5 Programmdokumentation (für Informatikkurse)
6 Hinweise für den Lehrer (Didaktik und Methodik)
7 Programmende
Auswahl einer Option mit den Cursortasten "auf/ab", anschließend "Enter"
```

Jede Simulation erfordert gewisse Startwerte für die Population. Diese Werte kann man in der folgenden Maske schnell ändern. Der jeweils aktuelle Stand einer Simulation wird in einer Tabelle angezeigt.

```
Anfangs-Mausanzahl auf der Wiese               (10...100)  a=
Anfangs-Wieselanzahl auf der Wiese             ( 1... 10)  b=
Maximal erlaubte Fangzahl an Mäusen pro Wiesel ( 1... 10)  c=
Mindestanzahl gefangener Mäuse zum Überleben   ( 1...  c)  d=
Maus-Vermehrungsfaktor                         (0.5... 3)  e=
Folgende Startwerte sind voreingestellt:
Anfangs-Mausanzahl    =  30
Anfangs-Wieselanzahl  =   3
Maximale Fangzahl     =  10
Überlebenszahl        =   5
Maus-Vermehrungsfaktor=  1.8
Startwerte übernehmen (j,n) = ?
```

(C) Das Karteiverwaltungssystem ZINSY

Dieses befindet sich in zwei Versionen für unterschiedliche Wartungsaufgaben auf der Diskette des Dümmler-Verlages mit dem Titel SOFTWARE-WARTUNG. Die zweite Version benutzt fertige Bausteine und Menüs, die mit einem Tool erstellt wurden (Disket-te LEH-TOOLS), die erste Version ist in allen Teilen ausprogrammiert. Auch hier sagt schon das Hauptmenü Wesentliches über die Funktionen des Systems.

```
                    HAUPTMENÜ

 1: Autor bekannt                 --> weitere Daten suchen
 2: Stichwort bekannt             --> weitere Daten suchen
 3: Aufsatz- (Buch)titel bekannt  --> weitere Daten suchen
 4: Zeitschrift- (Buch)Nr.bekannt --> weitere Daten suchen
 ---------------------------------------------------------
 5: Ausgabe einer Stichwortliste (sortiert)
 6: Ausgabe einer Autorenliste   (sortiert)
---------------------------------------------------------
 7: Anzahl der Datensätze in der Datei angeben
 8: Ausgabe aller Datensätze
 9: Ausgabe eines bestimmten Datensatzes
---------------------------------------------------------
 E: Eingabe von Aufsätzen  (Büchern)
```

```
A: Änderungen der Datei
P: Datei packen (aktualisieren / Leersätze löschen)
-----------------------------------------------------
F: Zum Info-Menü / zum Dateinamen / zum Programmende
-----------------------------------------------------

Auswahl einer Option mit den Cursortasten AUF/AB,
danach <┘
```

Die grundlegende Datenstruktur kann man aus der folgenden Information ablesen:

```
ZINSY arbeitet mit Datensätzen,          Man kann aber auch z.B.die
folgenden Aufbau haben                   Schallplatten katalogisieren.
                                         Man denkt sich dazu:

1) | Zeitschrift-(Buch-) Nummer |  =  1) | Schallplatten-Nummer    |
2) | Autor                     |  =  2) | Komponist               |
3) | Aufsatztitel              |  =  3) | Titel des Musikstücks   |
4) | Stichwörter               |  =  4) | Stichwörter             |
```

(D) Das Multiple-Choice-System MUCHO

Mit diesem System kann man Multiple-Choice-Tests auswerten. Das System kann direkt beim Buchautor bestellt werden (Preise auf Nachfrage). Zwei der Menüs geben einen ersten Eindruck von den Möglichkeiten:

```
                    MENÜ 1
V: Testvorbereitungen, grundlegende Listen
S: Statistik für den Testleiter
F: Formulare für Testpersonen ausgeben
?: Erläuterungen zu den Menüpunkten V, S, F
E: Neue Dateien wählen oder Programmende
```

```
                    MENÜ 1.1
PERSONENLISTE.....................................
A: Erstellen,  B: Ändern ,  C: Ansehen.

LlSTE MIT RICHTIGEN LÖSUNGEN.....................
D: Erstellen,  F: Ändern ,  G: Ansehen.

LISTE MIT LÖSUNGEN DER TESTPERSONEN...............
H: Erstellen,  K: ändern ,  L: Ansehen.

W: Definieren eines eigenen Notenschlüssels, vor-
   definiert sind ein Sek1-  und ein Sek2-Schlüssel

Z: Zurück zum vorigen Menü, Menü 1
E: PROGRAMMENDE.
```

(E) Die Zahlenlotto-Simulation LOTTO

Mit dieser Simulation können Sie Lottotips erzeugen, eine Ziehung durchführen und die Tips auswerten lassen. Das System kann direkt beim Buchautor bestellt werden.

```
          H A U P T M E N U E
        1 Tippen,von Hand oder Computer
        2 Ziehung u. Vergleich mit Tip·
        3 Statistik·····················
        4 Anleitung, Hilfen·····
        5 Abbruch (Quit)···············
        Bitte mit dem Cursor auswählen!
```

(F) Münzspiel MUENZEN

Nach Einzahlung je eines Chips dreier Spieler werden zwei Münzen geworfen. Abhängig vom Ausfall des Wurfes kann der Gewinner die drei Chips für sich entnehmen. Geeignet als kürzeres neues Projekt oder zur Wartung. Das System kann direkt beim Buchautor bestellt werden (Preise auf Nachfrage).

```
          MÜNZSPIEL
         Spielen 1 (für Anfänger)
         Spielen 2 (Fortgeschrittene)
         Spielregeln
         Textdatei zur freien Verfügung
         Hinweise für den Lehrer
         Programmende
```

```
          SIMULATION EINES MÜNZSPIELS
-----------------------------------------------------
             Anzahl von Chips   Anzahl der Gewinnspiele
Spieler 1              000               000
Spieler 2              000               000
Spieler 3              000               000
-----------------------------------------------------
Der nächste Münzwurf (j,n) = ?     ---> ??

| WW: Spieler 1 gewinnt | ZZ: Spieler 2 gewinnt   |
| ZW oder WZ:Spieler 3 gewinnt                    |
-----------------------------------------------------
Spielprotokoll
Anzahl der Chips von Spieler 1 am Anfang = 10
Anzahl der Chips von Spieler 2 am Anfang = 10
Anzahl der Chips von Spieler 3 am Anfang = 10
```

Es wird noch darauf hingewiesen, daß die Softwareprodukte (A),(B),(C, eine Version), (E) und (F) mit Tools und Bausteinen erstellt wurden, die auf der Dümmler-Diskette LEH-TOOLS gespeichert sind. Zu dieser Diskette gibt es das Buch ("Formelsammlung") "Programmieren in Turbo-PASCAL mit Hilfe von Bausteinen", Dümmler-Verlag 1994.

5. Aufgaben zu den Kapiteln 1-4

Hinweis: Etliche der genannten Aufgaben lassen sich auch anderen Kapiteln zuordnen.

Aufgaben zu Kapitel 1.1 (Was sind komplexe Softwareprodukte?)

1) Sie kennen vermutlich einige größere Programmsysteme. Vergleichen Sie zwei von ihnen nach den in Kapitel 1.1 genannten **Komplexitätskriterien**.

Aufgaben zu Kapitel 1.2 (Was versteht man unter ist Software-Engineering?)

1) Sie haben im Verlauf ihres Informatikunterrichts diverse Programme geschrieben, können sich also als "Programmierer" bezeichnen! Welche **Fähigkeiten und Vorgehensweisen** sind Ihnen beim Schreiben der Programme besonders nützlich gewesen?

2) Erläutern Sie die Tabelle in Kapitel 1.2 mit den Vorgehensweisen in ausgewählten Berufszweigen. Ergänzen Sie weitere **für den jeweiligen Beruf notwendige Methoden.**

3) Oben werden **Methoden eines Software-Ingenieurs** aufgelistet. Welche dieser Metho-den haben Sie bereits angewendet? Bei welcher Gelegenheit?

4) Begründen Sie, warum die **Herstellung von komplexen Softwareprodukten** nicht von einer einzigen Person erfolgen sollte (kann). Denken Sie dabei an Ihnen bekannte umfang-reichere Softwareprodukte, wie z.B. an ein Textverarbeitungssystem.

5) Im Verlauf Ihrer Ausbildung haben Sie Programme allein erstellt! Wie würde sich die **Arbeitsweise bei der Programmerstellung** vermutlich ändern, wenn Sie mit einem ande-ren Schüler zusammenarbeiten würden?

6) Erläutern Sie die folgende **Begriffsbestimmung: "Software-Engineering** ist die syste-matische Verwendung von Methoden und Werkzeugen zur Herstellung von Software mit dem Ziel einer Rationalisierung des Herstellungsprozesses bei gleichzeitiger Qualitätssteigerung."

Aufgaben zu Kapitel 1.3 (Projektarbeit im Informatikunterricht)

1) Erläutern Sie den **Begriff der Teamarbeit.**

2) Erläutern Sie den für den Arbeitsmarkt und die Berufsausbildung wichtigen Begriff der ***Schlüsselqualifikation*.**

3) Bei Vorliegen eines Auftrags zur Softwareentwicklung wird man sich mit dem zugrundeliegenden **Anwendungsbereich** auseinandersetzen müssen. - Nennen Sie Informationen, die zu sammeln wären, wenn man ein Programmsystem
a) zur Verwaltung einer Patientendatei eines Arztes,
b) für den Ausleihverkehr einer Bücherei,
c) für ein Vokabel-Lernprogramm erstellen soll.

Aufgaben zu Kapitel 2.1.1 (Software-Life-Cycle)

1) Welche Vorteile bringt die **Zerlegung** eines komplexen Systems in Teilsysteme?

Aufgaben zu Kapitel 2.1.2 (Prototyping)

1) Welche **Probleme bei der Softwareherstellung** haben zur Idee des Prototyping geführt?

2) Warum ist der **Software-Life-Cycle** in der in Abbildung von Kapitel 2.1.1. dargestellten Form unrealistisch?

3) Erläutern Sie die Abbildungen 2.1.2.a,b zum **Prototyping**.

4) Auch in der Industrie kennt man den Begriff des **Prototyp**s! Nennen Sie Beispiele. Wa-rum ist der Begriff nicht eindeutig übertragbar auf die Softwareherstellung?

Aufgaben zu Kapitel 2.1.3 (Projektmanagement, Teamarbeit)

1) Nennen Sie Aspekte, in denen sich ***Teamarbeit*** von der ***Einzelarbeit***, aber auch von der Zusammenarbeit z.B. nur zweier Personen unterscheidet.

2) Erläutern Sie: Jedes **Team einer Projektgruppe** ist *Empfänger und Zulieferer* von Informationen.

3) Welche Eigenschaften sollte Ihrer Meinung nach ein Schüler haben, der Sie und andere Mitschüler in der Funktion eines **Projektmanager**s "leitet"?

4) Entwerfen Sie für kleine Projekte geeignete (sehr begrenzte) **Checklisten** für den Projektleiter. Hinweis: Wegen des möglichen Umfangs solch einer Liste sollten Sie sich die Arbeit teilen, etwa gemäß den einzelnen Phasen der Softwareentwicklung.

5) Nennen Sie **Unterschiede in der Projektleitung** an einer Schule und in einem Betrieb!

6) Die folgende Tabelle gibt einen Überblick über **Softwareentwicklung** und Softwarewartung in aktuellen Projekten.
a) Erläutern Sie die in den einzelnen Punkten der Anwendungsentwicklungen auftretenden Begriffe.

b) Bei welchen Punkten handelt es sich um neue Projekte?
c) Warum wird für Wartung und Weiterentwicklung soviel Kapazität benötigt?

Aufteilung der Kapazität für die Anwendungsentwicklung	
Projekte zur Rationalisierung	30.6%
wettbewerbsorientierte Projekte	12.4%
Wartung	20.1%
Weiterentwicklung	25.7%
Interne Projekte zur Datenverarbeitung (DV-Infrastruktur)	11.2%

5) Eine **Befragung** zu 1990 und 1991 abgeschlossenen DV-Projekten hat u.a. ergeben:
- In fast 40% der Projekte wurde keinerlei Projektmanagement eingesetzt.
- Fast jedes zweite Projekt hatte keine konkreten "Meilensteine".
- Nur ca. 47% hielten die Projektlaufzeit ein.
- Die Hälfte der Projekte hielt sich nicht an die Aufwandsschätzung.

Erläutern Sie mögliche Gründe und Folgen dieser Feststellungen.

6) Für die **Wartung von Software** haben sowohl das Alter der Software als auch die bei der Entwicklung verwendete Methodik eine große Bedeutung! - Erläutern Sie diese Aussage - benutzen Sie auch die folgende Statistik. Eine Untersuchung hat 1992 ergeben:

Anwendungen insgesamt		Dialog-Anwendungen
Mehr als 10 Jahre alt	15.5%	7.3%
5 bis 10 Jahre alt	32.2%	30.5%
bis 5 Jahre alt	52.3%	62.2%
Altersstruktur der Anwendungen		

Aufgaben zu Kapitel 2.2.2 (Strukturiertes Programmieren ...)

1) Beleuchten Sie die von Ihnen bisher verwendete **Programmierpraxis** unter den Aspek-ten von Aufgabe 2, also u.a. bzgl. der Begriffe "strukturiertes Programmieren", "Bottom-Up-Methode", "Top-Down-Methode" usw.

2) Ein Beispiel aus der Mathematik: Wir betrachten das allgemein bekannte Lösungsverfahren zur **Lösung einer quadratischen Gleichung** mit der p,q-Formel. Das Verfahren soll so programmiert werden, daß nach Eingabe der Werte p und q die Lösungen der quadratischen Gleichungen ausgegeben werden. - Die Aufgabe wird von zwei Schülern unterschiedlich bearbeitet.

a) Vergleichen Sie die beiden Lösungen unter den Aspekten Arbeitsweg, Programmierung, Strukturierung! Berücksichtigen Sie die Begriffe "Top-Down" und "Bottom-up".

Schüler 1 setzt sich an den Computer und schreibt das PASCAL-Programm. Da ihm die Formel gut bekannt ist, beginnt er mit ihrer Übersetzung in die Programmiersprache. Dann programmiert er die Eingabe, zunächst noch ohne Eingabetext und die Ausgabe. Ein erster Test zeigt ihm, daß das Programm für manche p,q-Werte abstürzt. Die IF-Anweisung entsteht. Nun fügt er die Texte ein. Nach etlichen Tests erhält er schließlich:

```
PROGRAM quadratische_gleichung;
VAR p,q,d,x1,x2: REAL;
BEGIN
        Clrscr;
        WRITELN(´Qudratische Gleichung lösen:´);
        WRITE(´p=  ´); READLN(p); WRITE(´q=  ´); READLN(q);
        d:=p*p/4-q;
        IF d<0 THEN WRITELN(´Keine Lösung, da negative Radikand.´)
        ELSE
        BEGIN
                WRITELN(´x1= ´, -p/2+SQRT(d));
                WRITELN(´x2= ´, -p/2-SQRT(d));
        END;
END.
```

Schüler 2 überlegt sich, daß das Programm aus drei Teilen besteht:

1. Eingabe von p und q
2. Berechnung von x1 und x2, wobei eventuell ein negativer Radikand auftritt
3. Ausgabe von x1 und x2

Er programmiert daraufhin so:

```
PROGRAM quadratische_gleichung;
VAR p,q,d,x1,x2: REAL;
{*}
BEGIN {Hauptprogramm}
        eingabe;
        rechnung;
        ausgabe;
END.
```

und ergänzt dann bei {*}

```
PROCEDURE  eingabe;
BEGIN
        Clrscr;
        WRITELN(´Qudratische Gleichung lösen:´);
        WRITE(´p=  ´); READLN(p); WRITE(´q=  ´); READLN(q);
```

```
END;
PROCEDURE rechnung;
        d:=p*p/4-q;
        IF d>=0 THEN
        BEGIN
                x1:=-p/2+SQRT(d); x2:=-p/2-SQRT(d);
        END;
END;
PROCEDURE ausgabe;
BEGIN
        IF  d<0 THEN WRITELN(´Keine Lösung, da negative Radikand.´)
        ELSE WRITELN(x1,x2);
END;
```

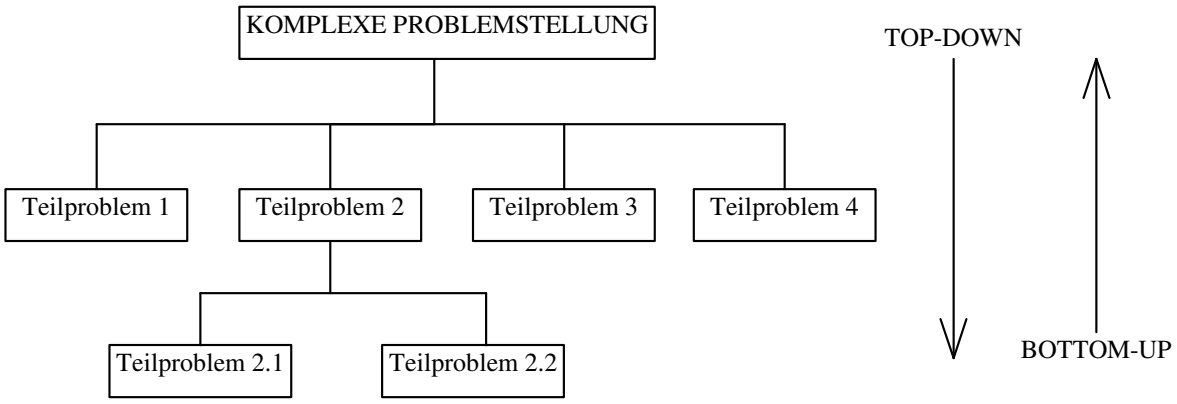
Auch Schüler 2 testet zwischendurch sein schrittweise entstehendes Programm: Erst das Hauptprogramm, wobei er die Prozeduraufrufe zunächst so formuliert:

```
PROCEDURE EINGABE;
BEGIN
        WRITELN(´Hier werden später p und q eingegeben.´); READLN;
END;
```

(ensprechend für die anderen Prozeduren).
Danach programmiert er diese Prozeduren nacheinander aus, testet sie jeweils, bis schließlich das gesamte Programm fertig ist.

b) Welche der beiden Arbeitswege ist besser geeignet für die Bearbeitung umfangreicherer Probleme? Begründung!
c) Formulieren Sie eine dritte Lösung unter Verwendung der Lösung 2, indem Sie die Prozeduren mit geeigneten Parametern versehen.

3) Erläutern Sie die folgende Abbildung



AUFGABEN zu Kapitel 2.2.3 (Das Prozedurkonzept)

1) Erläutern Sie die **Begriffe** Prozedur, Prozedurkopf, Prozedurrumpf, formaler Parameter, aktueller Parameter, Eingangsparameter, Ausgangsparameter.

2) Bei vielen Aufgabenstellungen hat man Tabellen (Matrizen) einzugeben. Man kann hierzu eine **Eingabemaske** verwenden, die man als wiederverwendbaren Baustein konstruieren sollte. Für die Eingabe einer (3,5)-Tabelle (3 Zeilen, 5 Spalten) ist z.B. die folgende Maske geeignet.

```
        1        2        3        4        5
 1 ..........  ..........  ..........  ..........  ..........
 2 ..........  ..........  ..........  ..........  ..........
 3 ..........  ..........  ..........  ..........  ..........
```

Schreiben Sie eine PASCAL-Prozedur, die Eingaben in eine Tabelle realisiert. Benutzen Sie Variablenparameter für die Zeilen- und Spaltenanzahl.

Aufgaben zu Kapitel 2.2.4 (Das Modulkonzept)

1) Nennen Sie Geräte, die aus verschiedenen **Bauteile**n aufgebaut sind. Um welche Bauteile handelt es sich? Erläutern Sie dabei auch den Begriff der Normung!

2) Die **Zerlegung der Module in Prozeduren** folgt gewissen Regeln, die man aus verschiedenen Gründen beachten sollte. Nennen Sie solche Regeln, und begründen Sie ihren Sinn.

3) Betrachten Sie die folgende **Spezifikation einer Turbo-PASCAL-Unit**:

```
Unit char_adt;
INTERFACE
USES Crt,Ino_u;
TYPE zeichen_menge = SET OF CHAR;
     string12      = STRING[12];

PROCEDURE Eingabe_zeichen(x,y:INTEGER;
                  input_aufforderung:STRING;
                  VAR antwort:CHAR;
                  erlaubte_zeichen: zeichen_menge);
PROCEDURE Eingabe_string (x,y : INTEGER;
                  input_aufforderung : STRING;
                  VAR input : STRING;
                  stringlaenge :INTEGER;
                  erlaubt : Zeichen_menge);
PROCEDURE Eingabe_textdatei(var datei_name:string12;
                name_erfragen:boolean);
```

IMPLEMENTATION

Mit dieser Spezifikation wird ein **abstrakter Datentyp** definiert, bezeichnet mit CHAR_ADT. Der Datentyp heißt abstrakt, weil die interne Struktur für den Benutzer kei-ne Rolle spielt. Vielmehr wird in der Beschreibung des Datentyps von seiner Konstruktion abgesehen. Diese wird im Implementierungsteil versteckt. - Der Datentyp benötigt die

Importschnittstellen (Units) *Crt, Ino_u* und enthält als **Exportschnittstellen** die
Datenobjekte *zeichen_menge, string12* und die
Operationen *PROCEDURE Eingabe_zeichen*
PROCEDURE Eingabe_string
PROCEDURE Eingabe_textdatei
einschließlich ihrer Parameter.

Datenobjekte und Operationen auf ihnen sind die Schnittstellen, die durch Aufruf des abstrakten Datentyps CHAR_ADT exportiert werden können. Die detaillierte Beschreibung bzw. Programmierung der Prozeduren erfolgt in der Implementierung (siehe Diskette, Unit CHAR_ADT.TPU/PAS).

a) Das folgende Programm zeigt eine Anwendung des abstrakten Datentyps CHAR_ADT. Erläutern Sie das Programm!

```
PROGRAM adt_test;
USES Crt,Ino_u, Char_adt;
VAR dateiname: string12; {string12 wurde in Char_adt definiert}
        c: CHAR;
BEGIN
  dateiname:='versuch.tes';
  eingabe_textdatei(dateiname,true);
  {diese Prozedur ist eine Operation aus dem ADT (der Unit) Char_adt}
  WRITELN('Mit der folgenden Prozedur aus der Unit Ino_u erfolgt');
  WRITELN('die Ausgabe der eben eingegebenen Textes.');
  c:=Readkey;
  {die Prozedur stammt aus der Unit Crt}
  output_textdatei(1,1,80,25,dateiname,'');
  {diese Prozedur ist eine Operation aus der Unit Ino_u}
END.
```

b) Die folgende Prozedur gehört zum Implementierungsteil des abstrakten Datentyps CHAR_ADT und steht normalerweise dem Benutzer der Unit nicht zur Verfügung.. Erläu-tern Sie den Algorithmus.

```
PROCEDURE Eingabe_zeichen(x,y:INTEGER; input_aufforderung:STRING;
                  VAR antwort:CHAR; erlaubte_zeichen: zeichen_menge);
BEGIN
  Gotoxy(x,y);
  WRITE (input_aufforderung);
  x := x + LENGTH(input_aufforderung);
  TEXTBACKGROUND (magenta); TEXTCOLOR (white);
  WRITE(antwort);
  REPEAT
    Gotoxy(x,y);
    antwort:=Readkey;
    WRITE(antwort);
  UNTIL antwort IN erlaubte_zeichen;
  TEXTBACKGROUND (black); TEXTCOLOR (lightgray);
END;
```

4) Welche Vorteile bringt die **Zerlegung** eines komplexen Systems in Moduln?

Aufgaben zu Kapitel 2.2.5 (Grafische Darstellungsmittel ...)

1) In Kapitel 2.2.5 wird ein Programmablaufplan für ein **Trainingsprogramm zur Prozentrechnung** angegeben.
a) Erläutern Sie den Entwurf.
b) Der Entwurf ist geeignet, um eine (wenig umfangreiche) Projektarbeit durchzuführen! Wie könnte man hierzu das Problem in Teilprobleme zerlegen, die dann arbeitsteilig bearbeitet und anschließend zusammengefügt werden könnten?

2) Nennen Sie **Vor- und Nachteile** der beiden Darstellungsformen Struktogramm und Programmablaufplan. Vergleichen Sie.

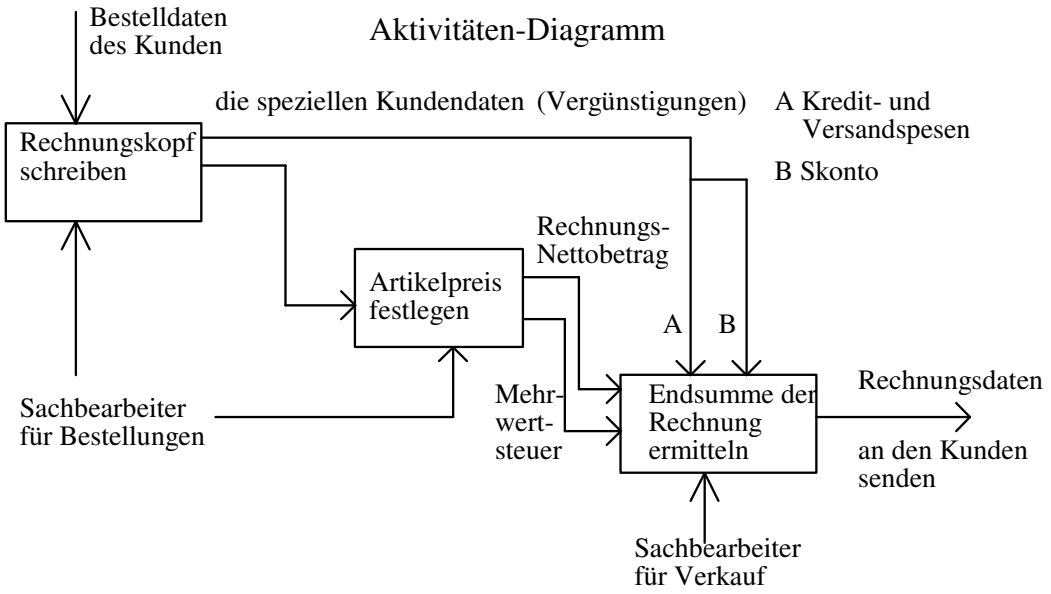
3) Nennen Sie **Vor- und Nachteile** der beiden Darstellungen (SADT, HIPO). Vergleichen Sie!

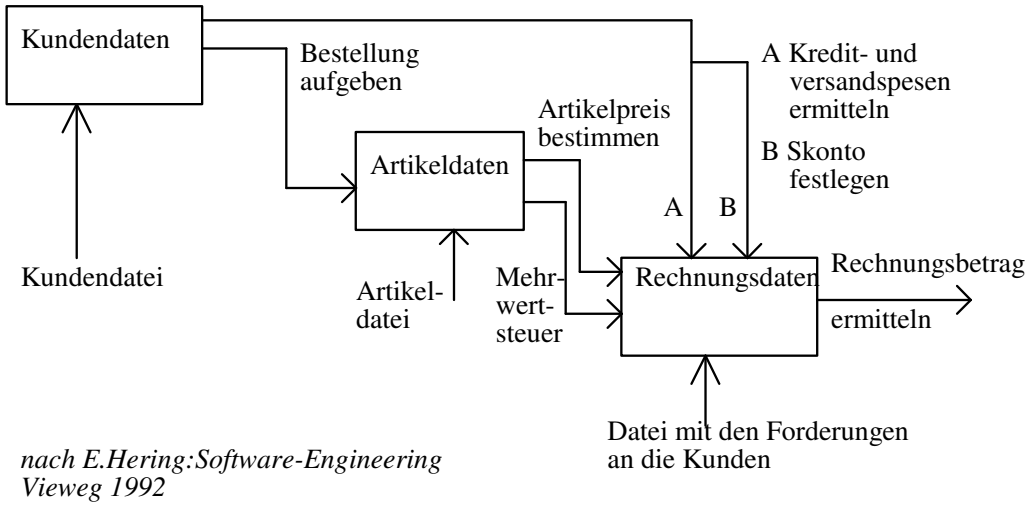
4) Entwerfen Sie ein Struktogramm für die **Ermittlung des Rechnungsendbetrages** für einen Kunden. Der Kunde habe 3 Artikel zum Preis von 45 DM, 89 DM und 124 DM bestellt. Er erhält 5% Skonto (keine weiteren Vergünstigungen). - Übersetzen Sie das Struktogramm in einen Programmablaufplan.

5) Eine **Telefonrechnung** soll erstellt werden. Konstruieren Sie ein Aktivitätendiagramm.

6) In Kapitel 2.2.5 finden Sie ein **SADT-Diagramm für eine Weinhandlung.** Die folgenden beiden Diagramme beschreiben ein Teilproblem. - Erläutern Sie jede

Abbildung. Vergleichen Sie die Darstellungen! Gehen Sie dabei insbesondere auf die funktionale Sicht im Vergleich zur datenbezogenen Sicht ein.





nach E.Hering:Software-Engineering
Vieweg 1992

7) Erstellen Sie HIPO-Diagramme zu den Abbildungen von Aufgabe 6

Aufgaben zu Kapitel 2.2.6 (Schnittstellen-Design)

1) Unter den in unserem Informatikkurs zur Verfügung stehenden Tools befinden sich auch solche, die die Entwicklung von Benutzungsoberflächen ermöglichen.
a) Nennen Sie solche Tools, und beschreiben Sie ihre Möglichkeiten.
b) Entwerfen Sie (auf Papier) ein Hauptmenü (höchstens 7 Menüpunkte) für ein Programmsystem, das die im gesamten Kurssystem (Klasse 11-13) besprochenen Mathematikaufgaben verwaltet.
c) Erläutern Sie dabei auch, warum Sie sich gerade für dieses Menü entschieden haben.
d) Realisieren Sie dieses Menü am Computer! Speichern Sie es unter dem Namen "ihrname.bal".

2) Äußern Sie sich zu den folgenden Aussagen:
a) Die Verwendung von Werkzeugen gibt den Arbeitsablauf und auch die Arbeitsergebnisse in erheblichem Umfang vor.
b) Werkzeuge beschränken die Ausdrucksmöglichkeiten ihrer Anwender.
c) Werkzeugeinsatz gewährleistet die syntaktische Korrektheit von Ergebnissen, jedoch nicht die semantische Korrektheit ihrer Resultate.

Aufgaben zu Kapitel 2.2.8 (Qualitätsanforderungen)

1) **Ein Rollenspiel**: Ein Schüler Ihres Kurses wird zum Käufer einer Software, z.B. eines mathematischen Programms (Funktionenplotter) ernannt, ein anderer Schüler tritt als Verkäufer der Software auf.
a) Formulieren Sie in einer Arbeitsgruppe die Ansprüche des Käufers an die Software.
b) Überlegen Sie sich in einer weiteren Arbeitsgruppe eine Strategie für das Verhalten des Verkäufers.
c) Starten Sie das Rollenspiel, also das Gespräch zwischen Käufer und Verkäufer. Die an-deren Schüler notieren sich die Argumente der Gesprächspartner.
d) Beurteilen Sie den Verlauf des Gesprächs

- nach Verhalten der Gesprächspartner
- nach der Güte ihrer Argumente.

2) Bringen Sie verschiedene Masken auf den Bildschirm (oder betrachten Sie entsprechende Kopien auf Papier). **Beurteilen Sie die Qualität dieser Masken** nach folgenden Gesichtspunkten:

Optische Wirkung, Überschaubarkeit, Verständlichkeit, Benutzerfreundlichkeit, Bildschirmaufteilung, Anordnung der Texte, Problemangemessenheit. Fügen Sie ggf. weitere Beurteilungskriterien hinzu.

3) In Kapitel 2.2.8 finden Sie eine Zusammenstellung von **Qualitätsmerkmale**n aus der Sicht des Benutzers. Erstellen Sie eine ähnliche Tabelle aus der Sicht eines Programmie-

rers, der die Software möglicherweise später auf Grund geänderter Anforderungen "warten" soll.

4) Nennen Sie Merkmale, die Sie am **Kauf einer Software** hindern würden.

Aufgaben zu Kapitel 2.2.9 (Das Testen von Programmen, Testmethoden)

1) Begründen Sie: Wenn innerhalb eines größeren Systems **ein Modul getestet** werden soll, so sollte man nicht den Konstrukteur des Systems einsetzen, sondern einen anderen Tester wählen!

2) Erläutern Sie die Aussage: "Man kann beim **Testen** immer nur die Anwesenheit von Fehlern, nicht jedoch ihre Abwesenheit nachweisen!"

3) Oben wird vom Test fertiger Softwareprodukte gesprochen! Sie erwägen den **Kauf eines Softwareprodukts!** - Welchen Lieferumfang würden Sie von dem Softwareprodukt erwarten? Formulieren Sie Gütebedingungen an das Softwareprodukt, die für Ihre Kaufentscheidung maßgeblich sein können.

Aufgaben zu Kapitel 2.3 (Dokumentation, Dokumentationshilfsmittel)

1) Nennen und erläutern Sie die wesentlichen **Gründe für die Anfertigung einer Dokumentation eines Softwareprodukts.** Welche Schwierigkeiten ergeben sich bei der Dokumentation?

2) Warum muß man zwischen **Benutzer- und Wartungshandbuch** unterscheiden? Was bedeutet in diesem Zusammenhang der Begriff "Wartung"? Nennen Sie Bestandteile der Dokumentation, die nur in das Wartungshandbuch gehören.

3) Schreiben Sie einen **Werbeprospekt** für das gerade in Arbeit befindliche Softwareprodukt bzw. für ein Ihnen bekanntes Softwareprodukt.

4) Kann man die **Dokumentation** einer Software schon vor ihrer Herstellung anfertigen?

5) Bezüglich der Dateien wird in DIN 66230 (siehe Aufg.6) ausführlicher gefordert: Zusammenstellung nach Dateiname, Datenträger, Dateiorganisation, Zugriffsart, Speicherbedarf, Datenflußplan. - Formulieren und erläutern Sie die einzelnen **Anforderungen** am Beispiel ihres aktuellen Projektes oder einer Ihnen bekannten Software.

6) Nach DIN 66230 sollte eine **Programmdokumentation** u.a. aus den unten aufgelisteten Teilen bestehen. Erläutern Sie die einzelnen Punkte!

1. Programm-Kenndaten	
Programmname und Variante	Versionsnummer und Freigabedatum
Aufgabenbeschreibung und Vorschriften	Gerätekonstellation
Programmgröße (Speicherplatzbedarf	benötigtes Betriebssystem, Hilfsprogramme
Programmiersprachen	Betriebsarten (Stapel-,Dialog- und Echtzeitbetrieb)
Dateien (Dateinamen, Zugriffsart, ...)	Zuständigkeit
2. Programm-Funktion und Aufbau	
Aufgabenstellung	Aufgabenlösung, z.B. Algorithmen
Programmaufbau	
Schnittstellenbeschreibung (Cross-Referenzen)	
Programmablauf (z.B. Struktogramme)	
Daten- und Datenflüsse (Datenflußplan)	
Anwendungsgrenzen	Datensicherung
3. Installierung und Test	
Übergabedokument	Testverfahren und Testergebnisse
4. Programmbetrieb	
Bedienungshinweise	Unterbrechungen und Wiederanlauf (Datensicherung)

Aufgaben zu Kapitel 2.4 (Projektkontrolle)

1) In einer Informationsstunde über den Informatikunterricht Ihrer Schule sollen Sie Ihren jüngeren Schülern (Klasse 10) die **unterschiedlichen Rollen des Lehrers** bei normalem Informatikunterricht und bei Projektarbeit erläutern. - Welche Gesichtspunkte würden Sie nennen?

2) Könnte auch einer Ihrer **Schüler als Projektleiter** fungieren?

3) Welche der in Kapitel 2.4.1 ausgesprochenen **Warnungen** halten Sie für besonders wichtig? Welche Warnungen entsprechen Ihren bisherigen Erfahrungen? Ergänzen Sie Warnungen aus Ihrer eigenen Sicht.

4) Kann man den **Projektleiter mit einem Diskussionsleiter vergleichen**?

5) a) Erläutern Sie das folgende **Planungsschema zur Projektkontrolle** in der Phase der Modulprogrammierung:
b) Entwerfen Sie eine ähnliches Formular für die Dokumentation in der Entwurfsphase.

Projektkontrolle "Modulprogrammierung" Datum: 7.6.1992

Modul/Prozedur	Gruppe 1	Gruppe 2	Planung in Stunden ( ) verbrauchte Std.		Bemerkungen
M1.1	O		4	(3)	braucht Hilfe
M1.2	∅		2		
M2.1		#	3	(2)	
M2.2		⊕	1	(2)	
Informationsaustausch aller Gruppen			1 am 14.6.92		

∅ noch nicht angefangen, O in Arbeit, # fast fertig, ⊕ Arbeit abgeschlossen

Aufgaben zu Kapitel 3 (Das Projekt Trabrennbahn ...)

1) Wir arbeiten zur Zeit am Entwurf und der Programmierung der 4 Module unseres Projekts "Trabrennbahn" und stehen kurz vor dem Abschluß dieser Arbeiten.
a) Sie haben Ihre Prozeduren verschiedentlich getestet und sich dafür eigene "**Testumgebungen**" geschaffen. - Erläutern Sie diesen Begriff. Wie sind Sie vorgegangen?
b) Demnächst muß die Systemintegration der Module erfolgen. Die Module liegen als Units vor. Stellen Sie einen übersichtlichen **Arbeitsplan** für die dann notwendigen Arbeiten auf. Welche Schwierigkeiten erwarten Sie? Verdeutlichen Sie diese auch in Ihrem Ablaufplan.

2) Modul 4 (Wettschein auswerten) benötigt zum Testen eine größere Anzahl von Wettscheinen. Die folgende Demonstration am Bildschirm zeigt Ihnen eine Möglichkeit, relativ **schnell viele Scheine zu erzeugen.**
a) Nach welchen Grundideen arbeitet das Programm?
b) Schreiben Sie eine PASCAL-Prozedur , die Ihre Ideen nach a) realisiert.

3) Die Option "Renntag initialisieren" soll **nur für Operateure zulässig** sein. Welche programmtechnische Realisierung schlagen Sie vor? (Algorithmus in Worten angeben).

4) Das Programmsystem enthält bekanntlich eine **Zugangskontrolle** zu einigen Optionen.
a) Welchen Sinn hat die Zugangskontrolle bei den jeweiligen Optionen? -
b) Schreiben Sie ein Struktogramm zu dem anliegenden Programm K-PASS.PAS
(nur für die Zugangskontrolle in engerem Sinn).

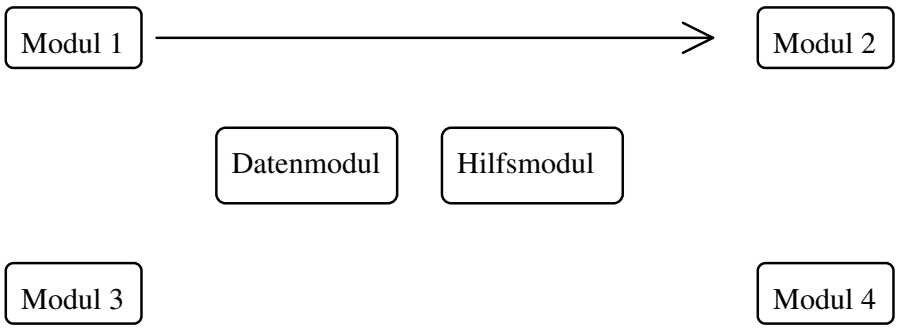
```
PROGRAM PASS;
USES Crt, Ino_u;
VAR ok: BOOLEAN;

PROCEDURE passwort_ein(VAR ok:BOOLEAN);
VAR zahl,passzahl: INTEGER;
BEGIN
  Clrscr; Gotoxy(1,5);
  RANDOMIZE;
  zahl:=RANDOM(100)+1;
  WRITE(zahl);
  Input_integer(1,6,'   Dazu bitte Ihr Paßwort: ',passzahl,2);
  WRITELN;
  IF zahl MOD (passzahl+3) = 0 then
    BEGIN
      WRITELN('Sie dürfen mein Programm benutzen!');
      ok:=true;
    END
  ELSE
    WRITELN('Sie dürfen mein Programm nicht benutzen!');
  READLN;
END;
BEGIN
  ok:=false;  passwort_ein(ok);
END.
```

c) Äußern Sie sich zu dem in der Prozedur PASSWORT_EIN verwendeten Parameter für den Fall, daß die Prozedur im Trabrennbahnsystem benutzt wird.
d) In das vorhandene System soll eine Überprüfung eingebaut werden, die sicherstellt, daß nicht ein Pferd an einem Tag in mehr als einem Rennen startet.
d1) Formulieren Sie die Lösungsidee (einen groben Algorithmus) in Worten.,
d2) Realisieren Sie ihre Idee in PASCAL. Benutzen Sie dabei folgende Deklarationen:

```
Const anzahl_der_rennen = 3;
Const anzahl_der_pferde = 5;
var pferde:array[1..anzahl_der_rennen,1..anzahl_der_pferde] of integer;
```

5) Das oben vorgelegte Datenmodul enthält nur die Datenstrukturen und die zur Verfügung gestellten (globalen) Variablen. Die darauf definierten Operationen fehlen hier. Sie stehen in den einzelnen Modulen. - **Organisieren Sie das Programm** so **um**, daß die Operationen auf den Daten von T_DS_U auch in diesem Modul stehen! Hinweis: Beachten Sie zu dieser Aufgabe die Ausführungen über den Aufbau von abstrakten Datentypen (ADT) in Kapitel 2.2.4.

6) **Welche Schnittstellen** gibt es zwischen den Modulen des Programmsystems?



Aufgaben zu Kapitel 4 (Softwarewartung - Reengineering)

1) **Warum Wartung?** - Warum muß Software gewartet werden? Denken Sie dabei auch an andere, Ihnen bekannte Softwareprodukte. - Stellen Sie stichwortartig Gründe zusammen.

2) Die folgende Zusammenstellung beschreibt in grober Form einen möglichen **Ablauf eines Reengineering-Projekts.** Erläutern Sie die genannten Phasen und ergänzen Sie ggf. durch weitere Aspekte.

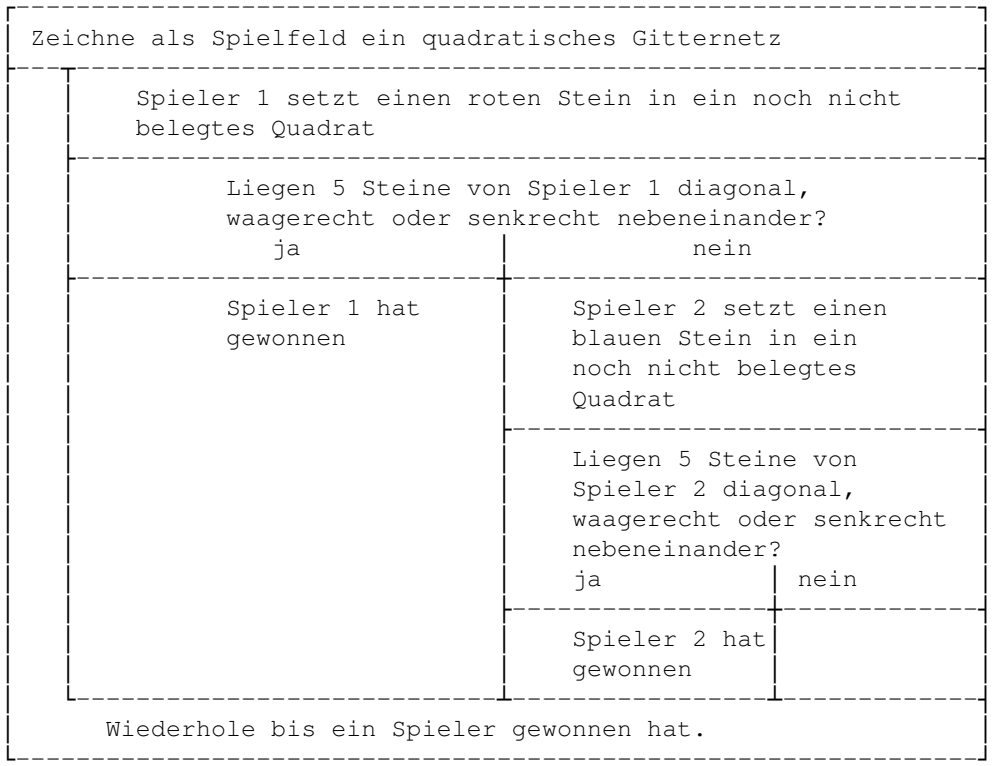
a) Formulierung der neuen Anforderungen
b) Analyse des vorhandenen Systems (program understanding)
c) Gegebenenfalls Beseitigung überflüssigen Codes und überflüssiger Dokumentationsteile im System
d) Erneutes Forward Engineering zur Realisierung der neuen Anforderungen
- Entwürfe, Programmierung, Dokumentation
e) Implementierung des neuen Systems

3) In einem Aufsatz über Reengineering findet sich die Frage:
"**Alt-Software: Problem oder Chance ?** " - Äußern Sie sich zu dieser Fragestellung. Beachten Sie dabei u.a. die Gesichtspunkte "Kosten, Wiederverwendbarkeit, Neuentwicklung, Bausteintechnik."

4) Betrachten Sie ein Ihnen zugängliches Softwareprodukt, und **beurteilen Sie die Benutzerfreundlichkeit** des Produkts (was ist gut, was ist schlecht?). Formulieren Sie entsprechende Wartungsaufträge an ein Softwarehaus.

5) Spielregeln als Struktogramm

Das folgende Struktogramm beschreibt die Spielregeln des "Gobang-Spiels". Zu diesem Spiel liegt ein Softwareprodukt vor, siehe Kapitel 4 und Buchanhang.
Die Spielregeln in Struktogrammform sollen nun auch vom Programm aus aufrufbar sein! Wählen Sie eine geeignete Stelle, an der Sie das Struktogramm aufrufen können.



6) Zu dem TRABRENNBAHN-System gehört die Unit T_DS_U mit der **Datenstruktur** des Systems. Hinweis: Entnehmen Sie die Datenstruktur dem Kapitel 3.
a) Warum empfiehlt es sich, für die Datenstruktur ein eigenes Datenmodul vorzusehen?
b) Notieren Sie je einen sinnvollen Datensatz für die Datentypen "rennen" und "wettschein".
c) Welche Gedanken haben vermutlich zu der vorgelegten Struktur von "wettschein" geführt?
d) Erläutern Sie den Datentyp "einlauftyp" und seine Verwendung im Programm.

7) Ein Softwarehaus erhält einen Reengineering-Auftrag für ein Softwareprodukt. - Vor welchen Problemen könnten die Angestellten beim Reverse Engineering stehen?

8) Erstellen Sie einen **Werbeprospekt** für das Progamm TRABRENNEN.
a) Dieser Prospekt soll aus einer Bildschirmseite bestehen, die Sie optisch ansprechend mit dem Tool BILD_DEF.EXE erstellen sollen.

b) Schreiben Sie ein kleines Programm, das diesen Bildschirm aufruft.
c) Im folgenden werden zwei mögliche Werbetexte angegeben (entworfen von einem Schüler in einer Klausur). Beurteilen Sie die beiden Bearbeitungen! Hinweis: Der Rahmen entspricht in etwa dem Bildschirm.

Bearbeitung 1:

T R A B R E N N E N (aktualisierte Version)
****

Vorteile und Funktionen:
- übersichtliche Programmierung
- Dokumentation für Benutzer und Programmierer
- erneuerter Grafikteil
- leicht verständliche Aufteilung und Handhabung
- der Natur nachempfunden:
- Rennen
- Wettscheine
- Gewinne
- hervoragendes Preis-Leistungs Verhältnis

FÜR WEITERE INFORMATIONEN STEHEN WIR JEDERZEIT ZUR VERFÜGUNG Ihr Rückert Softwareteam

Bearbeitung 2:

NEU T R A B R E N N E N : ÜBERARBEITETE VERSION

Funktionsumfang und Features:	- Organisation und Durchführung von Pferderennen - Verwaltung und Auswertung von Wettscheinen (HANDRECHNUNG WIRD ÜBERFLÜSSIG!) - Hohe Effizienz durch Computereinsatz - Automatische Auszahlung der Gewinne - 99.9% Sicherheit durch bewährtes Passwortsystem
Bestechend durch:	- Hohe Übersichtlichkeit und Ergonomie - Selbsterklärende Menüführung; nur wenige Verschachtelungen - Umfassende Handbücher für Anwender und Entwickler
Service-Leistungen:	- kostenloser Lehrgang inklusive - Telefonisches Sorgentelefon - Update-Service - Wunschgerechte Erweiterungen am Programmsystem möglich

10er Programmlizenz Optional erhältlich:

für nur DM 19.95	Wettscheinautomat Künftige Updates	DM 429.90 DM 8.95

Übergreifende Aufgabenstellungen

1) **Annonce aufgeben**

- Sie haben im Verlauf Ihres Informatikunterrichts an verschiedenen Projekten gearbeitet und dabei Software neu entwickelt oder auch ältere Software gewartet!
- Sie sind Personalchef eines Betriebes, in dem Software entwickelt oder gewartet wird!
- Sie wollen eine Annonce in einer DV-Zeitung aufgeben, in der Sie einen qualifizierten Mitarbeiter für diese Tätigkeiten suchen!

a) Welche Kenntnisse, Fähigkeiten und Eigenschaften eines zukünftigen Mitarbeiters würden Sie in der Annonce nennen? (stichwortartig, wie es in der Annonce stehen könnte).
b) Begründen Sie Ihre Auswahl (nur bzgl. der charakterlichen Eigenschaften des Bewerbers).

2) Bei der Softwareherstellung ist es von großem Vorteil, wenn man die **Gestaltung der Benutzungsoberfläche** vollkommen getrennt von dem sonstigen Programm durchführen kann!
a) Erläutern Sie den Begriff der Benutzungsoberfläche.
b) Begründen Sie die obige Feststellung.

3) Im folgenden finden Sie Auszüge aus einem Aufsatz über ein spezielles **Produktionsverfahren..**
a) Inwiefern sind viele der hier genannten Aspekte auch für die Arbeit an einem Informatik-Projekt zutreffend? Nennen Sie diese Aspekte.
b) Äußern Sie sich zu dem eingerahmten Text! (Jedes Team sieht ...)

Schlankmacher - Lean Production*: Die Unternehmensdiät für mehr Effizienz und Effektivität industrieller Produktionsverfahren (von H.Kargl)*
aus: Siemens-Nixdorf-Informationssysteme DIALOG, Juni 2/93

"... verspricht man sich auch bei der Lean Production eine kostengünstige Produktion bei hoher Qualität, kurze Lieferzeiten sowie ein flexibles und schnelles Reagieren auf Kundenwünsche. Und der ***Fehler! Verweisquelle konnte nicht gefunden werden.*** *enthält auf den ersten Blick Elemente, die schon aus anderen Fertigungsstrukturen bekannt sind:*
o strenge Ausrichtung auf den Materialfluß
o Bildung autonomer, objektorientierter Fertigungseinheiten ...
o ...
o weitgehende Gruppenarbeit in der Teile- und Baugruppenfertigung ...
o Gruppenarbeit an Montagebändern ...

o strenges Qualitätsmanagement

Was ist bei der Lean Production anders? Die wesentlichen Unterschiede zu bekannten Konzeptionen liegen in der Organisation der Gruppenarbeit am Band und in der Strenge des Qualitätsmanagements. Bereits in den 70er Jahren machte der Automobilhersteller Volvo erste Versuch mit Gruppenarbeit in Form ***Fehler! Verweisquelle konnte nicht gefunden werden.****, die die monotone Fließbandarbeit ablösen und so - über eine Humanisierung der industriellen Arbeit - die Mitarbeiter wieder motivieren sollte... Jeder dieser Arbeitsgruppen obliegt in eigener Verantwortung die Aufteilung, Durchführung und Qualitätssicherung ihrer Arbeit...Das Denken in Qualitätskategorien beginnt ... schon bei der Produktentwicklung...*
Der entscheidende Unterschied von Lean Production zu anderen Formen der industriellen Sereinfertigung liegt jedoch im Teamkonzept, das nicht nur das gesamte Unternehmen durchzieht, sondern das auch auf Kunden und Zulieferer ausgedehnt wird... Auslöser für die teamorientierte Arbeitsgestaltung war der bekannte Sachverhalt, daß weitgehende Arbeitsteilung wegen der damit verbundenen langen Informationswege, vielstufigen Entscheidungsprozesse und dem dazu erforderlichen hohen Kooridinationsaufwand ineffizient ... ist. Statt tayloristischer Funktionsspezialisierung dominiert deshalb heute die Objektorientierung als Gestaltungsprinzip der Organisation... Objekte in diesem Sinne können Entwicklungsvorhaben, zu fertigende Baugruppen, aber auch Leistungsprozesses wie etwa zwischen Unternehmen und Kunden oder Zulieferern sein.

Objektorientierte Arbeitsgestaltung verläuft zwangsläufig quer zur funktionalen Gliederung eines Unternehmens. Deshalb müssen die Träger dieser Organisationsform Arbeitsgruppen oder Teams sein... Die Produktentwicklung wird als Gemeinschaftsaufgabe in einem interdisziplinär zusammengesetzten Team durchgeführt, in dem vor Ort vorhandenes Fachwissen aus Konstruktion, Design, Marketing, Werkzeugbau, Fertigung, Beschaffung, Kostenrechnung und Wartung unmittelbar eingebracht wird.
Der Entwicklungsprozeß verläuft dadurch nicht mehr sequentiell, sondern vorwiegend parallel. So verläuft beispielsweise parallel zur Produktentwicklung die Verfahrensentwicklung oder die Planung der Qualitätssicherung und der Zulieferungen ab...

Jedes Team sieht sich dabei als Empfänger und Zulieferer zugleich und achtet deshalb nicht nur auf die Qualität der empfangenen Leistung, sondern auch auf die Qualität der weitergegebenen Leistung.

Außerhalb des Betriebs sind auch Kunden und Lieferanten in das Teamkonzept einbezogen... die Kunden sollen außerdem frühzeitig als Lieferanten von Informationen über Produktmängel, Kundenwünsche und veränderte Marktgegebenheiten gewonnen werden... Voraussetzung ist allerdings eine weitgehend offene Informationspolitik." (Ende des Zitats)

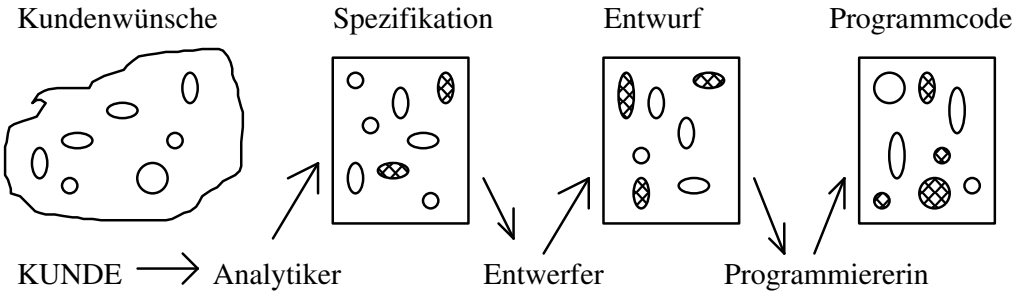
4) Entwerfen Sie einen **Werbeprospekt** für eine von Ihnen erstellt Software.

5) Erläutern Sie die Aussage: "Die **Programmiersprache** ist unerheblich. Sie spielt in den frühen Phasen der Softwareentwicklung keine Rolle!" - Oder doch?

6) **Typische Aufgaben eines Softwarehauses**:
- Durchführung von Entwicklungsprojekten, für Kunden (externe Auftraggeber) oder im Rahmen einer eigenen Produktentwicklung
- Erledigung von Beratungsaufträgen bei Kunden
- Durchführung von Kundenseminaren
- Vertrieb der eigenen Produkte

Welche Anforderungen an die Berufsausbildung eines Software-Ingenieurs lassen sich daraus ableiten?

7) Aus der Praxis der Softwareentwicklung: **Kundenwünsche und Programm**.
Äußern Sie sich zu der folgenden Abbildung, versuchen Sie, eigene Erfahrungen einzubringen!




geforderte Eigenschaften gehen teilweise verloren

unnötige Eigenschaften kommen hinzu

8) In einem Vortrag wurden von einem großen Software-Hersteller (Anwendungsentwicklung) u.a. folgende erwünschte **allgemeine Fähigkeiten von Informatik-Studenten** genannt.
- Fähigkeiten, sich in ein Anwendungsgebiet einzuarbeiten und dieses Wissen in technische Anforderungen umzusetzen
- Technische Randbedingungen für den Kunden verständlich darzustellen
- Techniken der Systemanalyse eines vorhandenen DV-Systems und Einarbeitung in die organisatorischen Rahmenbedingungen

Welchen Beitrag kann dabei die Schule durch Projektunterricht leisten?

9) In einem Fachbuch wird definiert: "**Software Engineering** ist eine Disziplin, die *mit ingenieurmäßigen Mitteln* und *ökonomischem Vorgehen* dem Entwicklser hilft, *qualitativ hochwertige Software* zu erstellen und zu *pflegen."*
Erläutern Sie die kursiv gesetzten Aspekte!

10) Sie haben gerade ein DV-Projekt in Arbeit oder planen es! Formulieren Sie **Ausbaustufen** für das geplante Produkt!

11) Sie wollen ein komplexes Software-Produkt verkaufen!
Welche Faktoren spielen für den Käufer bei seiner Kaufentscheidung eine besondere Rolle? Gehen Sie bei der Beantwortung besonders auf Qualitätseigenschaften ein! Andere Kriterien können kürzer dargestellt werden.

12) In einem Beitrag zur gymnasialen Schulbildung der Zukunft heißt es u.a.:
"In Tutzing sprechen die Kultusminister inzwischen über vergleichbare Leistungsstandards im Abitur und über die **Schlüsselqualifikationen**, die ein Abiturient für seinen weiteren Lebensweg haben sollte. Bei den Schlüsselqualifikationen sind sich die Kultusminister weitgehend einig: Arbeiten im Team, Aushalten von Frust, Bereitschaft zum lebenslangen Lernen, Verantwortungsbewußtsein, Durchhaltevermögen, geistige Neugier, Kritik und Urteilsfähigkeit, Erziehung zur Kreativität - das sind Eigenschaften, ohne die man im Berufsleben kaum Erfolg haben kann. Diese Schlüsselqualifikatioen sollen nach dem bewährtem Muster der beruflichen Bildung künftig auch intensiv an den Gymnasien vermittelt werden." (Uwe Schlicht: Zurück zum Klassenverband? Aus DER TAGESSPIEGEL vom 28.12.94).
Erläutern Sie, inwiefern die Durchführung eines Softwareprojektes oder die Wartung eines Softwaresystems viele der genannten Forderungen erfüllen kann!

13) In Kapitel 2.1.1 wird ein **Software-Life-Cycle** abgebildet, in Kapitel 4 finden Sie eine entsprechende Abbildung zum Software-Reengineering! - Vergleichen Sie die beiden Abbildungen und gehen Sie auf die Unterschiede ein.

14) Erläutern Sie den Begriff der **Software-Altlast!**

15) Der Fachbereich Sport einer Schule besitzt ein umfangreiches **Inventar** an Sportgeräten und Fachliteratur. Beides **soll mit Hilfe der Datenverarbeitung katalogisiert werden.**
a) Welche Daten sollte man speichern?
b) Geben Sie ein Beispiel für einen Datensatz.
c) Entwickeln Sie eine entsprechende Datenstruktur in PASCAL.
d) Welche Optionen sollte man vorsehen, um eine Aktualisierung der Daten zu ermöglichen?
e) Welche Vorteile sehen Sie für den Fachbereich Sport, wenn er einen derartigen Katalog auf einem Computer zur Verfügung hat?

16) Sie haben im gegenwärtigen Kurs **Projektarbeit** kennengelernt!
a) Worin unterscheidet sich diese Arbeitsform von den normalerweise in der Schule verwendeten Formen? Verfolgen Sie dabei auch die einzelnen Projektphasen!
b) Halten Sie es für wünschenswert in allen Fächern und ständig an Projekten zu arbeiten?

6. Glossar

Aktivitäten in einem Projekt (Rollen)

In einem Softwareprojekt gibt es zahlreiche unterschiedliche Arten von Tätigkeiten. Eine gewisse (kleine) Menge solcher Tätigkeiten wird zu einer Aufgabe zusammengefaßt, die von einer oder mehreren Personen bearbeitet wird und die man als eine **Fehler! Verweisquelle konnte nicht gefunden werden.** Aktivität bezeichnet.

Architektur eines Software-Systems

Das äußere Erscheinungsbild der Komponenten eines Softwaresystems. Für alle Eingabemöglichkeiten wird angegeben, was für Ausgaben zu erwarten sind.

Aufwandsschätzung bei Softwareprojekten

Schaffen wir das vorgesehene Softwareprojekt in der vorhandenen Zeit?

Expertenschätzung: Experten (möglichst aus verschiedenen Bereichen) schätzen und diskutieren den Aufwand solange, bis Einigkeit erzielt wird.

Analogieschluß: Man benutzt ein ähnlich gelagertes, schon abgeschlossenes Projekt zur Schätzung des Aufwands.

Bottom-Up-Schätzung: Bei Vorliegen einer zumindest groben Spezifikation kann man den Aufwand für jede Komponente des Systems (einschließlich der Systemintegration) schätzen und die Schätzwerte addieren. Dabei kann man z.B. folgende Größen betrachten: Anzahl der Eingaben und Ausgaben, der Abfragen, der Dateien, der Schnittstellen.

Als kaum geeignet für eine Aufwandsschätzung hat sich die Zuordnung von Prozentwerten zu den einzelnen Phasen des Software-Life-Cycle erwiesen! Schon in der Literatur variieren die Angaben erheblich!

Für die Schule, die in der Regel keinem Zwang unterliegt, ein bestimmtes angefordertes Produkt zu erstellen, hat es sich als beste Methode erwiesen, den Projektumfang von vornherein gering anzusetzen, die Thematik also entsprechend einzuschränken. Gegebenenfalls kann diese Minimalversion dann immer noch erweitert werden.

Bearbeitungsauftrag, Bearbeitungszustand

Im Verlauf des Projekts werden die Mitarbeiter i.a. zu Mitarbeitergruppen zusammengefaßt und diverse Bearbeitungsaufträge vergeben. Bearbeitungsaufträge bestehen aus der Aufgabenstellung, der Angabe der zu verwendenden und zu erstellenden Dokumente und Ressourcen. Jeder Bearbeitungsauftrag besitzt einen Bearbeitungszustand. Man kann z.B. unterscheiden zwischen den Zuständen

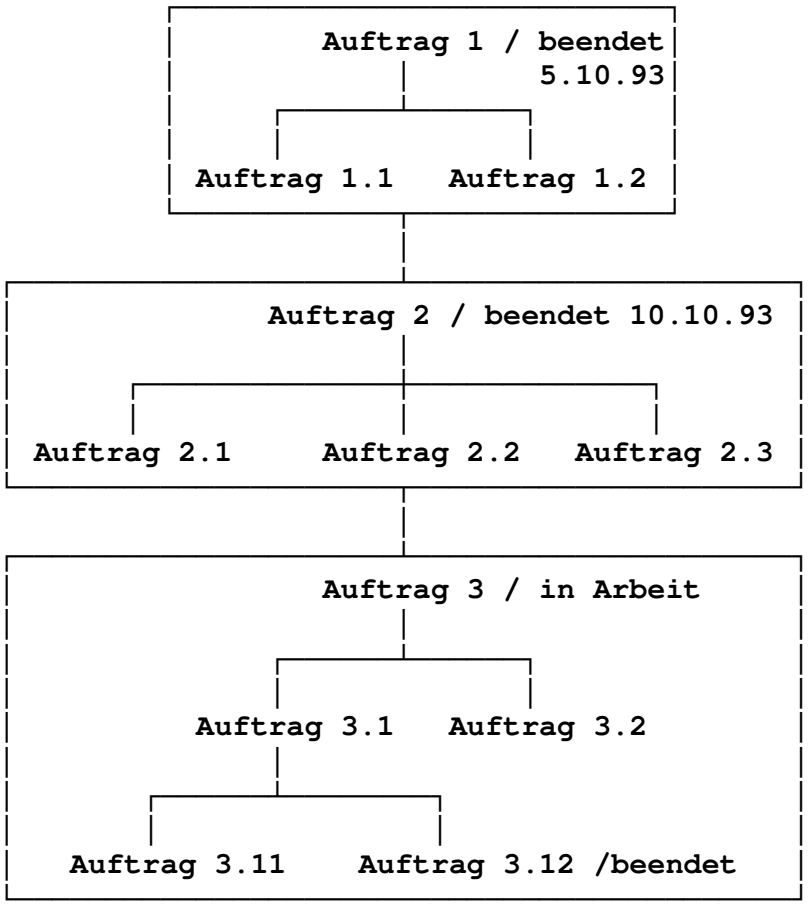
- Auftrag nicht existent
- Auftrag definiert
- Auftrag in Bearbeitung (am Anfang, etwa zur Hälfte bearbeitet, kurz vor Fertigstellung)
- Auftragsbearbeitung unterbrochen
- Auftrag abgebrochen

- Auftrag fertig

Der Bearbeitungszustand sollte vom Projektmanager jederzeit abgefragt werden können. Hierzu sollte von der Arbeitsgruppe jeweils ein Formular aktualisiert werden, etwa der folgenden Form:

Auftragsnummer: *3.2*	Auftragserteilung am : *5.11.93*
Auftraggeber: *Lehrer*	Aktuelles Datum : *11.11.93*
Ende der Bearbeitung:	
Bearbeiter : *Simon, Florian*	
Auftragsbeschreibung: *Entwurf einer Eingabemaske für die Daten der Skipisten.*	
Bearbeitungszustand: *Auftrag in Bearbeitung, kurz vor der Fertigstellung*	
Bemerkungen: *Die Positionen für die Eingabe der Datensätze müssen noch aufge - schrieben werden.*	

Die Aktualität und ständige Verfügbarkeit dieses Dokuments dient gleichzeitig der Information der anderen Projektgruppen. Im Projektverlauf ergibt sich eine zeitliche Abfolge von Aufträgen.



Benutzerschulung

Rechtzeitig vor dem Ende des Projekts bzw. der Fertigstellung der Software muß (bei breiterer Anwendung des Produkts, etwa in einem Betrieb) eine Benutzerschulung erfolgen. Diese wird i.a. in einem Lehrgang von einem Mitarbeiter des Software-Herstellers durchgeführt.

Bezeichnungen von Dateien

Während des Projektablaufs entstehen zahlreiche Dateien, so daß man leicht den Überblick über diesen Dateienbestand verliert. Hier empfiehlt sich eine passende Bezeichnung der entstandenen Dateien mit einer Kurzbeschreibung ihrer Funktion. Man kann z.B. unter-scheiden (etwa bei einem Projekt)

	Bezeichnungsvorschlag
Fehler! Verweisquelle konnte nicht gefunden werden.	
Programmquellen	*ski*.pas*
compilierte Programme	*ski*.exe*
Dokumentationstexte	*ski*.dok (*.txt)*
vom Programm benutzte Hilfsdateien	*ski*.bal (Menüs)*
	ski.mas (Bildschirmseiten)*
Module:	*ski_m1.pas, ski_m2.pas usw.*
Testumgebungen:	*ski-tm1.pas*
Units :	*ski_u1.pas*

CASE

Computer Aided Software-Engineering, CASE-Systeme enthalten Werkzeuge, mit denen Arbeiten innerhalb des Software-Engineering-Prozesses wesentlich unterstützt werden.
Manche Produkte unterstützen den gesamten Prozeß (integrierte CASE-Umgebung), erfordern allerdings viel Einarbeitungszeit. Derartige Software-Entwicklungsumgebungen können Systeme enthalten zur Projektplanung, Projektkontrolle, Maskenerzeugung, Übersetzung von Quellprogrammen, Verwaltung von Modulbibliotheken, Textverarbeitung usw. Siehe auch Kapitel 2.2.6. - Man kann unterscheiden zwischen zwei Anforderungsbereichen an CASE-Produkte:
"Upper CASE" findet vorwiegend auf der Workstationseite statt, auf der die Planungs-, Analyse- und Design-Komponenten angesiedelt sind. Die hier (lokal) erzielten Ergebnisse sollten aus den lokalen Data Dictionaries durch einen Koordinator zentral (z.B. in einer Entwicklungsdatenbank auf einem Server) gesammelt werden.
"Lower CASE" umfaßt die Generierungswerkzeuge für Programme und Datenbankschemata sowie die diversen Test- und Wartungshilfen.

Cross-Referenzen

Cross-Referenzen sind Beziehungen zwischen verschiedenen Programmteilen. Sie werden in **Crossreferenztabellen** zusammengefaßt. Beispiele sind
- Modul - Modul - Beziehungen (siehe Kapitel 3.5)
- Modul - Datei - Beziehungen
- Aufruftabellen globaler Variabler durch die Prozeduren des Programms
- Tabellen für Prozeduraufrufe in verschiedenen Programmteile

Derartige Tabellen verschaffen Überblick über die verwendeten Module, ihre Zusammenhänge und darüber, wie sie das Programm beeinflussen. Damit wird gleichzeitig die Wartung von Programmen erleichtert.

Beispiel von Modul-Datei-Beziehungen

Datei Modul	Datei 1	Datei 2	Datei 3	Datei 4	Datei 5	Datei 6	Datei 7
Modul 1	R	S	W				
Modul 2		R	W	R		U	
Modul 3			U	D			
Modul 4	D					D	D
Modul 5	W				U		

R read W write D delete U update S sort

Data-Dictionary

Ein Datenlexikon (Data-Dictionary) gibt eine Übersicht über alle bei der Datenverarbeitung eines Betriebes benutzten Daten und ihrer Strukturierung. Innerhalb eines Projektes kann auf diese Weise der Überblick über die an verschiedenen Geräten erzeugten Dateien (Pro-gramme, Masken, Menüs usw.) gewährleistet werden. Siehe auch CASE.

Design

Entwurf, Muster, Gestaltung; beispielsweise werden innerhalb des Software-Entwicklungsprozesses oft Masken als Teil der Programmoberflächengestaltung entworfen. Das Design (die Gestaltung) der Oberfläche ist wesentlich für die Akzeptanz eines Programmsystems durch die Benutzer. "Design ist funktionelle Attraktivität!" In weitem Sinn umfaßt das Design den gesamten Entwurfsprozeß von Software.

Ergonomie

siehe Software-Ergonomie und Kapitel 2.2.8

Informationsaustausch

zwischen den Mitarbeitergruppen ist eine Grundvoraussetzung für die kooperative Arbeit im Projekt. Zu diesem Austausch gehören insbesondere die gemeinsamen Sitzungen, in denen über den Projektstand berichtet und die weitere Arbeit beraten wird.

Konfigurationsmanagement

Systematische Überwachung aller Änderungen an der Konfiguration eines Softwaresystems über den gesamten Lebenszyklus

Kontextsensitive Hilfe

Innerhalb von Programmsystemen werden in der Regel Hilfefunktionen für den Benutzer angeboten. Im einfachsten Fall werden alle Hilfen in einer Datei zusammengefaßt, in der der Benutzer nach der gesuchten Information durch Blättern suchen kann. Bessere Hilfesysteme bieten die Hilfe bezogen auf den gerade vorhandenen Programmzustand an, sie beziehen sich auf den Kontext. In Kapitel 2.2.7 wird als ein mögliches Hilfsmittel die Prozedur "Output_textdatei" beschrieben.

Meilensteine

dienen der Terminplanung eines Projekts. Sie beschreiben das Ende einer Arbeitsphase und sind wichtig für die Kontrolle des Projektfortschritts. Die Terminplanung kann auch in Form von graphischen Darstellungen, z.B. mit Balkendiagrammen, erfolgen.

Messen und Bewerten von Prozessen und Produkten

ist für die Erstellung qualitätsorientierter Software unerläßlich! Siehe auch unter "Qualitätssicherung", Kapitel 2.2.8.

Objektorientierte Software-Entwicklung (OOS)

Man vermutet, daß der objektorientierte Ansatz für die Software-Entwicklung in den nächsten Jahren eine ähnliche Bedeutung erlangen wird, wie die strukturierte Programmierung in den achtziger Jahren. OOS ist ein Mittel, um Software übersichtlich zu strukturieren und sie aus wiederverwendbaren Bausteinen zu entwerfen. Das kann zu einer erheblichen Reduktion von Entwicklungs- und Wartungsaufwand führen und damit auch zu einer schnelleren Softwareproduktion von hoher Qualität.
Hier kann nur ansatzweise auf die Grundlagen des objektorientierten Entwerfens und Programmierens eingegangen werden. Für die Schule liegen noch kaum Erfahrungen vor. Allerdings werden in dem vorliegenden Buch bereits einige der Konzepte berücksichtigt (siehe Kapitel 2), insbesondere

- abstrakte Datentypen
- Zerlegung in voneinander unabhängige Module
- Geheimnisprinzip
- Benutzung wiederverwendbarer Bausteine

Die objektorientierte Sicht: Ein komplexes System besteht aus abgeschlossenen, voneinander unabhängigen Einheiten, den **Objekten** (auch **Instanzen** genannt). Gleichartige Objekte werden der gleichen **Klasse** zugeordnet. Aus dem Text in Kapitel 2 sind die abstrakten Datentypen und die Prinzipien der Datenkapselung und des Information Hiding

bekannt. Die Objekte bestehen aus Daten (**Instanzvariablen**), die den veränderlichen Zustand des Objekts beschreiben und aus den **Methoden,** den Aktionen auf den Daten. Während der Laufzeit werden Methoden ausgeführt, bei denen die Objekte **Nachrichten** an andere Objekte verschicken können.
Objekte sind also Ansammlungen von Daten, die auf Nachrichten durch Ausführung von Methoden reagieren.

Vorhandene Bausteine brauchen bei kleinen Änderungen nicht mehr neu geschrieben zu werden. Der objektorientierte Ansatz ermöglicht es, ähnliches Verhalten, ähnliche Baumuster auf einfache Weise auszudrücken. Die Konzepte der Klasse und der Vererbung dienen dazu, Gemeinsamkeiten und Verallgemeinerungen mit programmiersprachlichen Mitteln zu beschreiben.

Objektbasierte Softwaresysteme bestehen aus untereinander durch Nachrichten kommunizierenden Objekten. Herkömmliche Software dagegen ist prozedurorientiert, die Daten und die Operationen darauf sind i.a. nur lose gekoppelt. Unterprogramme übergeben anderen Unterprogrammen Daten zur Manipulation.

Präsentation von Projektergebnissen

Die Ergebnisse einer Projektarbeit können auf verschiedene Weisen präsentiert werden. Konstruktion eines "Films" (Demonstrationsprogramm), Handbücher, Referate mit Rechnereinsatz und Projektion auf einem LC-Display, Folien für den OH-Projektor, Wandzeitung. In der Schule können solche Präsentationen z.B. vor anderen Kursen oder Schulklassen stattfinden.

Projektmanagement

Bei der Planung und Abwicklung von Projekten besteht die Gefahr, daß der Überblick verloren geht. Hier muß eine geeignete Organisationsform - das Projektmanagement - helfen. Einzelheiten kann man in den Kapiteln 2.1.4 und 2.4 nachlesen.

Projektzustand

Ein Projektzustand umfaßt die Zustände aller Prozesse und Produkte eines Projekts zu einem bestimmten Zeitpunkt. Die Sichtbarmachung des aktuellen Projektzustands ermöglicht die Beobachtung des Projektfortschritts, z.B. das Einhalten von Terminen. Das Projektmanagement oder die Mitarbeiter (die Schüler) erkennen, welche umfassenderen Prozesse zu dem betrachteten Zeitpunkt gerade aktiv sind bzw. aktiviert werden können. - Für den Projektunterricht in der Schule bedeutet die Feststellung des aktuellen Projektzustands einen Haltepunkt, der den Schülern Überblick verschafft. Der Projektzustand kann von den einzelnen Gruppen z.B. in Referaten, auf Folien oder in einem Schriftstück beschrieben werden. Die Feststellung von Projektzuständen ist auch deswegen so wichtig, weil die verschiedenen Projektmitglieder unterschiedliche Interessen haben. Sie betrachten in der Regel nur die für sie wichtigen Teile und Aspekte eines Projekts. Diese unterschiedlichen Sichtweisen gilt es zusammenzuführen.

Qualitätssicherung

Das Pflichtenheft für eine Softwareproduktion enthält außer den funktionalen Anforderungen auch Qualitätsanforderungen wie Benutzerfreundlichkeit, Zuverlässigkeit, Wartbarkeit, Portabilität, Effizienz, Ergonomie (Gestaltung des Programms entsprechend den geistigen und körperlichen Bedürfnissen der Benutzer). Ausführliche Darstellung in Kapitel 2.2.8

Reengineering

Wartung eines Softwareprodukts, siehe unter Wartung. Das Reengineering beginnt mit dem Reverse Engineering und geht dann zum Forward Engineering über. Siehe Kapitel 4.

Reverse Engineering

Die Analyse eines bestehenden Systems mit den Zielen:

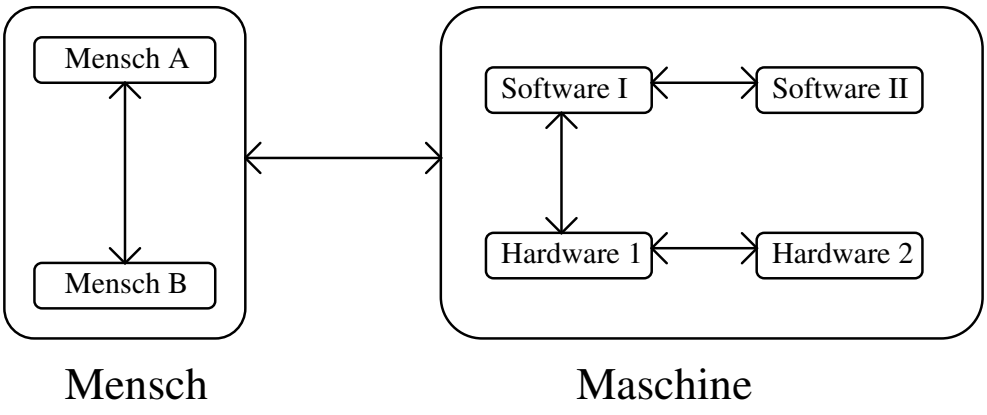
- Wiedergewinnung des Softwareentwurfs und der Anforderungsspezifikation, um die Systemkomponenten und ihre Beziehungen zu verstehen
- eine andere Form oder eine auf einer höheren Abstraktionsebene basierende Systembeschreibung zu erstellen.

Schablonentechnik

Diese Technik arbeitet mit einem (noch nicht ablauffähigen) Programmskelett. In diese vorgegebene Schablone können dann die nötigen Dateinamen, Variablennamen usw. eingetragen werden.

Schnittstellen

Es gibt verschiedene Arten von Schnittstellen, siehe Abbildung. Für das Buchvorhaben sind insbesondere die Schnittstellen Mensch-Maschine und Software-Software von Bedeutung, wobei hier auch die Schnittstellen zwischen verschiedenen Programmteilen gemeint sind, siehe Kapitel 2.2.6.



Schnittstellen: Mensch-Mensch
Mensch-Maschine
Software-Hardware
Hardware-Hardware
Software-Software

Software

Gesamtheit der für den Betrieb einer EDV-Anlage nötigen Programme und der Anwenderprogramme.

Software-Ergonomie

Ergonomie ist ein Aufgabengebiet, das sich mit der Anpassung von Arbeitsmitteln und Arbeitsumgebungen an die Eigenschaften des arbeitenden Menschen befaßt; siehe auch

Kapitel 2.2.8 (Qualitätsanforderungen an Software). Der Computer ist inzwischen zu einem weit verbreiteten Arbeitsmittel geworden, so daß die Anforderugen an eine menschengerechte Gestaltung der Arbeitsplätze am Computer gestiegen sind. Dabei geht es insbesondere darum, Kriterien und Methoden zur Gestaltung dialogorientierter Programmsysteme zu entwickeln. Zu den Kriterien zur Beurteilung von Softwaresystemen unter ergonomischen Gesichtspunkten gehört u.a. auch die Sozialverträglichkeit, wie das Vermeiden sozialer Isolierung am Bildschirm oder die Erhöhung der Qualifikation des Benutzers.

Software-Sanierung

Unter diesem Begriff faßt man alle Maßnahmen zusammen, mit denen eine in ihrer Funktionalität unveränderte, ggf. geringfügig erweitere Software erzeugt wird. Sanierungsprozesse haben das Ziel einer Verbesserung der Qualität und Produktivität von Alt-Software.

Spezifikation

Im Verlauf eines Projekts sind verschiedentlich Spezifikationen zu erstellen. Das sind genaue Beschreibungen (auch in graphischer Form) des Aufbaus, der Eigenschaften und des Verhaltens eines zu beschreibenden Modells oder Systems. Im Verlauf des Software-Life-Cycles (SLC) fallen z.B. an: Anforderungsspezifikation, Entwurfsspezifikation. In der An-forderungsspezifikation werden die Anforderungen an ein zu konstruierendes System be-schrieben (aus der Sicht des Auftraggebers bzw. späteren Anwenders). Näheres bei der Be-schreibung des SLC in Kapitel 2.1.1.

Standardisierung

(1) **Bei der Entwicklung** eines Softwareprodukts sind zahlreiche Personen beteiligt und in verschiedenen Phasen voneinander abhängig. Eine effektive Zusammenarbeit ist nur möglich, wenn individuelle Eigenheiten bei Entwurf, Programmierung und Dokumentation ausgeschlossen werden und standardisiert wird. Bei der Projektarbeit kommt es also darauf an, standardisierende Vereinbarungen zu treffen. Im Informatikunterricht ist eine gewisse Standardisierung bereits durch den Unterricht vor den Projektkursen erreicht, indem beispielsweise bestimmte Darstellungsformen von Algorithmen gemeinsam geübt oder Programme gemeinsam entworfen wurden. Angesichts des großen Umfangs eines Projekts erhält das Prinzip der Standardisierung für die Schüler nun eine neue Dimension und muß vor allen Dingen bewußt gemacht werden. Insbesondere sind für die Dokumentation Vereinbarungen nötig, wenn man die Entstehung eines zusammenhanglosen, schlecht lesbaren Konglomerats von Einzeldokumenten verhindern will.

(2) Ein Softwareprodukt wird in der Regel **von vielen Anwendern** benutzt. Die Anwender sind über gewisse Standardisierungen der Benutzeroberfläche dankbar. Insbesondere ist für sie eine Dokumentation erforderlich, die gut handhabbar ist, bei deren Erstellung also ebenfalls standardisierende Vereinbarungen getroffen wurden (s.o.).

Testen, Testumgebung

Komplexe Systeme bestehen aus mehreren Modulen, die später vom Hauptprogramm aufgerufen werden. Die Module werden getrennt entworfen und getrennt getestet. Dazu muß für einen Modul eine **Testumgebung Fehler! Verweisquelle konnte nicht gefunden werden.** konstruiert werden. Das Modul**Fehler! Verweisquelle konnte nicht gefunden werden.** wird von einem speziellen Testtreiber aufgerufen, der die Versorgung des Testlings mit Eingabeparametern übernimmt. Von außen benutzte Prozeduren werden durch **DummyroutinenFehler! Verweisquelle konnte nicht gefunden werden.** ersetzt. Diese simulieren die externen Originalprozeduren und stellen ggf. Rückgabewerte (unabhängig von den Eingabeparametern) zur Verfügung. Außerdem können sie**Fehler! Verweisquelle konnte nicht gefunden werden.** verwendet werden, um Werteveränderungen globaler Variablen (man sollte sie vermeiden!) zu verfolgen.

Zielsystem	Testdurchführung
Hauptprogramm	Testumgebung
Anwendung	Aufruf der Testumgebung
Modul	Modul
Anwendungsfunktion	Dummyroutine für die Anwendung

Abb.: Modultest

Die Definition der Testdaten kann in einer Textdatei erfolgen. Hier werden die Belegungen globaler Variablen, Eingabeparametern und Rückgabewerten für die Aufrufe zu den ein-zelnen Testfällen festgelegt. Siehe Kapitel 2.2.8

Validierung

Das Testen im dynamischen Ablauf mit Testdaten, siehe Test.

Wartung

a) Arbeiten, die nötig sind, um ein System bei unvorhergesehenen Vorfällen wieder lauffähig zu machen.
b) Anpassung eines Systems/Programms an veränderte Gegebenheiten (Reengineering), siehe auch Kapitel 4.

Zuverlässigkeit

Ein Softwareprodukt ist umso zuverlässiger, je höher der Grad der Korrektheit, Ausfallsicherheit, Robustheit gegenüber falscher Bedienung, Konsistenz (Entwicklung des Produkts nach vereinheitlichenden Gesichtspunkten) und Vollständigkeit bezüglich der angeforderten Leistungen ist.

7. Disketten

7.1 Trabrennbahn

SOFTWARE-ENGINEER

Die Diskette SOFTWARE-ENGINEER für DOS-Rechner gehört zu dem hier vorliegenden Buch. Sie ermöglicht

a) das Nachvollziehen der Ausführungen zum Projektablauf "Trabrennbahn" in Kapitel 3,
Hauptprogramm T1234.EXE.

b) die Durchführung eines Reengineering-Projekts "Trabrennbahn-Neu", siehe Kapitel 4.

c) das Nachvollziehen des Reengineering-Projekts aus Kapitel 4,
Hauptprogramm S1234.EXE.

Zu a) und b) dienen die Dateien T*.* und die dazugehörigen Hilfsdateien, während c) anhand eines durchgeführten Reegineering-Projekts mit der neuen Version S*.* und denselben Hilfsdateien verfolgt werden kann. Die Namen der Dateien zur T*.*-Version finden Sie auf den Seiten 111 und 117 von Kapitel 3.4.

Damit Sie das Reengineering-Projekt mit der T*.*-Version (oder auch der S*.* Version) durchführen können, werden die verwendeten Baustein-Bibliotheken auf der Diskette zur Verfügung gestellt. Es sind dies die Units

1) Ino_u.tpu (Turbo-Pascal 6), Ino_u.pas (Quelldatei)
2) Balk_u.tpu (Turbo-Pascal 6), Balk_u.pas (Quelldatei)
3) Bild_u.tpu (Turbo-Pascal 6), Bild_u.pas (Quelldatei)

Sie haben damit die Möglichkeit, auch in anderen Turbo-Pascal-Versionen zu arbeiten, so daß die vorliegende Diskette in sich abgeschlossen ist. Allerdinsgs fehlen auf dieser Diskette die ausführlichen Dokumentationen dieser Units, Beispielanwendungen sowie weitere Baustein-Bibliotheken und Hilfsprogramme zur Menü- und Maskenerzeugung (Programme MENUE und BILD_DEF). Diese und weitere Dateien befinden sich auf der Diskette LEH-TOOLS. - Wenn Sie also in Ihrem Informatikkurs konsequent mit Bausteinen und Menüerzeugungshilfen arbeiten wollen, empfiehlt sich die Anschaffung dieser Diskette und die des dazugehörigen Buches
Lehmann,E.: Programmieren in Turbo-Pascal mit Bausteinen, Dümmler-Verlag 1994, (siehe Kapitel 7.2).

7.2 Oberflächen und Bausteine

LEH-TOOLS

Diese Diskette gehört zu dem Schülerbuch
Lehmann,E.: Programmieren in Turbo-Pascal mit Bausteinen, Dümmler-Verlag Bonn 1993, Diskette LEH-TOOLS.
Die Schwerpunkte dieses Buches liegen bei den beiden Aspekten

- Entwerfen von Oberflächen (Menüs und Masken) unabhängig vom Programm
- Konstruieren mit Hilfe von Bausteinen.

INHALTSVERZEICHNIS DES BUCHES

INHALT DER DISKETTE LEH-TOOLS (für DOS-Rechner)

- Programme zur Oberflächengestaltung (Quellcode und compiliert) , Hilfsdateien
- Units (Toolsammlung) mit zahlreichen Prozeduren als Programmierhilfen (Quellcode und compiliert in Turbo-Pascal 6)
- Übungsaufgaben zur Arbeit mit Tools (Aufgabenlösungen ggf. mit Quelltext und in compilierter Form)

Da für die Programme und die Units der Quellcode zur Verfügung gestellt wird (in Turbo-Pascal 6), hat der Anwender die Möglichkeit, eigene Änderungen vorzunehmen oder die Quellen auch nur für folgenden Turbo-Pascal-Versionen fortzuschreiben und neu zu übersetzen. - Außerdem wird hingewiessen auf die Dateien *lehrer-1.** (Programm und Datei, mit didaktisch-methodischen Informationen für den Lehrer.

7.3 Spiel Gobang und Räuber-Beute-System

GOBANG - MAUS-WIESEL

Diese Diskette gehört zu dem Schülerbuch
Lehmann,E.: Software-Analyse im Informatik-Anfangsunterricht, Spiel GOBANG - Räuber-Beute-System MAUS-WIESEL ,Diskette, Dümmler-Verlag, Bonn 1993
Die Schwerpunkte dieses Buches liegen bei den beiden Aspekten

- Benutzen, Analysieren und Warten (in Ansätzen) komplexer Systeme sowie
- Konstruieren mit Hilfe von Bausteinen.

INHALTSVERZEICHNIS DES BUCHES

INHALT DER DISKETTE MAUS-WIESEL (für DOS-Rechner)

- Allgemeine Informationen (Dateien *readme.*, lehrer-1.*),*
- Programme und Dateien zum GOBANG-System u.a. Unterrichtsreihe,
- Programme und Dateien zum Räuber-Beute-System MAUS-WIESEL,
- Lösungen von Übungsaufgaben,

- Baustein-Bibliotheken (Units), die in den Programmen verwendet wurden,
- Informationen über weitere Literatur und Software des Autors (*lehsoft.**).

Sachverzeichnis

Aufgabenverzeichnis

Projekte im Informatik-Unterricht (Software-Engineering)

- ist entstanden aus den Erfahrungen bei vielen im Unterricht durchgeführten Projekten
- wendet sich an alle Personen, die umfangreichere Softwareprodukte erstellen wollen und insbesondere an Schüler und Lehrer von Informatikkursen, sowie Studenten und Dozenten
- eignet sich als Schülerbuch im Unterricht und zum Selbststudium
- zeigt den Ablauf eines Projektes anhand einer Entwicklungsdokumentation
- schildert beispielhaft die Wartung eines Softwareprodukts (Reengineering)
- bringt viele neuartige Übungsaufgaben zur Projektarbeit
- gibt viele Hinweise zur Unterrichtsmethodik bei Projektarbeit
- sammelt grundlegende Begriffe des Software-Engineering in einem Glossar

hat folgende Zielsetzungen

- Modellieren komplexer Problemstellungen
- Erlernen grundlegender Prinzipien der Konstruktion größerer Software-Systeme
- Kennenlernen und Benutzen von Methoden zur Organisation von Projektarbeit (Management)
- Anwenden von Dokumentationstechniken
- Kennenlernen grundlegender Begriffe des Software-Engineering
- Begreifen von Teamarbeit als einer Schlüsselqualifikation für viele Berufe

wird ergänzt durch die Diskette SOFTWARE-ENGINEER

- für MS-DOS-Computer
- mit zwei Versionen des Softwareprodukts "Trabrennen" (vor und nach der Wartung) in Turbo-Pascal 6

und durch die Diskette SOFTWARE-WARTUNG

- für MS-DOS-Computer
- mit zwei Versionen des Softwareprodukts "Zinsy" für Software-Wartungsprojekte (Zeitschriften-Informationssystem/Kartaeiverwaltung)