



# Uni Wien

**Business Informatics Group**

Institut für Softwaretechnik und Interaktive Systeme

---

## **Proseminar Grundlagen wissenschaftlichen Arbeitens Softwaretechnik**

Leitung: Prof. Wolfgang Hoschek

**0207378 Altinger Doris  
0207381 Bauer Astrid  
0209751 Memic Alisa**

**Wien, 5. Mai 2003**

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung .....</b>	<b>4</b>
<b>2</b>	<b>Einführung: Software .....</b>	<b>5</b>
2.1	<b>Software .....</b>	<b>5</b>
2.1.1	Eigenschaften von Software.....	5
2.1.2	Die Rolle der Software .....	6
2.1.3	Wozu dient Software? .....	6
2.1.4	Software-Entwicklung .....	7
2.2	<b>Software Engineering .....</b>	<b>8</b>
2.3	<b>Ziele und Mittel des Software Engineerings.....</b>	<b>8</b>
<b>3</b>	<b>Programmierung.....</b>	<b>9</b>
3.1	<b>Überblick .....</b>	<b>10</b>
3.1.1	Strukturierte Programmierung.....	10
3.1.2	Datenabstraktion.....	11
3.1.3	Programmierung im Großen .....	11
3.1.4	Formale Spezifikation und Verifikation von Programmen .....	12
3.2	<b>Modellierung.....</b>	<b>13</b>
3.2.1	Modellierungsmittel.....	13
3.2.2	Methoden .....	14
3.3	<b>Architektur von Softwaresystemen.....</b>	<b>15</b>
3.3.1	Architekturprinzipien .....	15
3.3.2	Organisationsformen.....	17
<b>4</b>	<b>Softwareentwicklungsprojekte .....</b>	<b>19</b>
4.1	<b>Überblick .....</b>	<b>19</b>
4.1.1	Tätigkeiten bei der Produktentwicklung .....	19
4.1.2	Vorgehensmodelle (life cycle models) .....	21
4.1.3	Evolutionäre Systementwicklung und Prototyping .....	24
4.2	<b>Softwareentwicklungsprozess .....</b>	<b>25</b>
4.2.1	Beteiligte bei der Softwareentwicklung .....	25
4.2.2	Leitbilder der Softwareentwicklung .....	26

4.2.3	Kooperation und Koordination .....	26
4.2.4	Produkt- und Konfigurationsverwaltung.....	27
4.2.5	Qualitätssicherung.....	27
4.2.6	Sicherung der Prozessqualität .....	29
<b>4.3</b>	<b>Softwareentwicklungswerkzeuge .....</b>	<b>30</b>

# 1 Einleitung

Die Softwaretechnik (software engineering) ist dasjenige Teilgebiet der Informatik, das sich mit der Erstellung, Nutzung und Weiterentwicklung (insbesondere großer) Softwaresysteme beschäftigt.

**Software** sind Programme, Verfahren, zugehörige Dokumentationen und Daten, die mit dem Betrieb eines Computersystems zu tun haben.

**Technik** bedeutete ursprünglich im Sinne des Aristotelischen Begriffs ‚Techne‘ (=Kunstfertigkeit). Heute versteht man unter Technik die Gesamtheit aller Objekte (Werkzeuge, Geräte, Maschinen u.a.), Maßnahmen und Verfahren, die vom Menschen durch Ausnutzung der Naturgesetze und –Prozesse sowie geeigneter Stoffe hergestellt bzw. entwickelt werden und sich bei der Arbeit und in der Produktion anwenden lassen. Darüber hinaus bezeichnet Technik die Wissenschaft von der Anwendung naturwissenschaftlicher Erkenntnisse.

**Engineering (=Ingenieurwesen)** ist die zusammenfassende Bezeichnung aller technischen Fachrichtungen, die sich heute zu eigenen Wissenschaftsbereichen entwickelt haben.

Ein **Softwaresystem** ist ein Ausschnitt aus der realen oder gedanklichen Welt, bestehen aus Softwareelementen und darin vorhandene Strukturen (z.B. deren Aufbau aus Teileinheiten oder Beziehungen untereinander).

Softwaretechnik ist heute ein umfangreiches und ausdifferenziertes Fachgebiet der Informatik mit vielfältigen Bezügen zu anderen Fachgebieten, wobei die Grenzen unscharf sind. Ferner steht sie im Austausch mit anderen Disziplinen wie Organisationstheorie, Arbeitspsychologie, Betriebswirtschaft und den Ingenieurwissenschaften. Seit der Entstehung der Softwaretechnik gibt es eine Vielzahl nebeneinander existierender Schulen, die jeweils ihre Sicht vertreten, eigene Methoden lehren, weiterentwickeln und sie veränderten Anforderungen anpassen. Dazu stellen sie passende

Werkzeuge zur Unterstützung dieser Methoden bereit. Daher existiert nicht die Softwaretechnik, sondern es gibt verschiedene sinnvolle Ausprägungen für unterschiedliche anwendungs- und technologiebezogene Schwerpunktsetzungen.

## **2 Einführung: Software**

Software ist in den vergangenen Jahrzehnten zu einem derjenigen Faktoren geworden, ohne die in den hochentwickelten Ländern nichts mehr geht. Die Entwicklung von Software wird jedoch bis heute nur ungenügend beherrscht. Da die softwarekosten den Hardwarekosten längst den Rang abgelaufen haben und weltweit horrenden Summen für Software ausgegeben werden, ist dies ein sehr unbefriedigender Zustand.

### **2.1 Software**

#### **2.1.1 Eigenschaften von Software**

Für ein adäquates Verständnis der Softwareentwicklung sind die spezifischen Eigenschaften von software zu berücksichtigen, deren Zusammenspiel sie von anderen technischen Produkten unterscheidet:

- Software ist nicht sinnlich wahrnehmbar.
- Software besteht aus Sprache. Sie kann fast beliebig strukturiert und verformt werden.
- Software ist digital.
- Software ist fehlerhaft. Fehler treten im Einsatz in unvorhersehbaren Situationen auf.

**Große Softwaresysteme weisen weitere Eigenschaften auf:**

- Sie sind komplex. Sie verknüpfen mehrere Realitätsbereiche mit ihrem Fachwissen und ihrer Fachsprache.
- Sie bestehen aus umfangreichen Texten: Spezifikationen, Programmen und unterschiedlichen Formen von Dokumentation.
- Sie verfestigen Sichtweisen. Verständnis und Interesse der Beteiligten schlagen sich im computerverarbeitbaren Modell des Anwendungsbereichs nieder. Umgekehrt prägt der Einsatz von Software Rahmenbedingungen für Arbeitsmilieus.

**2.1.2 Die Rolle der Software**

Rechner durchdringen alle Lebensbereiche. Wirtschaft und Gesellschaft sind abhängig geworden von der Software und die Abhängigkeit nimmt zu. Viele Selbstverständlichkeiten des Alltagslebens sind ohne Rechner und deren Software nicht mehr möglich.

In krassem Gegensatz zu dieser Abhängigkeit steht die Tatsache, dass weltweit die Erstellung von Software nur ungenügend beherrscht wird. Termin- und Kostenüberschreitungen bei Software-Projekten sind die Regel. Software, die Fehler enthält oder sich nicht entsprechend den Vorstellungen und Bedürfnissen der Benutzenden verhält, gehört zum Alltag.

**2.1.3 Wozu dient Software?**

Software dient dazu, ein Problem zu lösen oder zu dessen Lösung beizutragen, indem menschliche oder technische Arbeitsvorgänge automatisiert oder unterstützt werden. Software steht daher in einer ständigen Wechselwirkung mit Arbeits- und Produktionsprozessen und mit den daran beteiligten Menschen. Daraus folgen drei weitere wesentliche Eigenschaften von Software.

Wenn ein Problem von seiner Natur her komplex und schwierig zu lösen ist, so ist die Software zur Lösung dieses Problems in der Regel nicht weniger komplex und schwierig.

Bei der Lösung von Problemen mit Software sind immer zwei Schwierigkeiten gleichzeitig zu bewältigen:

- Das Problem ist im Kontext seines Sachgebiets zu verstehen und befriedigend zu lösen.
- Die Problemlösung muss auf adäquate Software-Strukturen abgebildet werden.
- Problemlösungen schaffen neue Realitäten und wecken neue Bedürfnisse.

Software ist daher nicht einfach ein Abbild der Realität und bisheriger manueller Problemlösungen. Sie konstruiert und verändert die Realität. Beispielsweise ermöglicht Logistik-Software die Fertigung von Gütern nach dem „Just in time“-Prinzip (Zulieferteile werden genau dann angeliefert, wenn sie in der Produktion benötigt werden). Dies schafft streng termingebundenen Lastwagenverkehr als neue Realität und softwaregestützte Optimierung solchen Verkehrs als neues Bedürfnis.

#### **2.1.4 Software-Entwicklung**

Software-Entwicklung umfasst alle Tätigkeiten und Ressourcen, die zur Herstellung von Software notwendig sind.

Sie ist die Umsetzung der Bedürfnisse von Benutzern in Software. Software-Entwicklung umfasst Spezifikation der Anforderungen, Konzept der Lösung, Entwurf und Programmierung der Komponenten (Dabei wird unter <<Programmierung>> sowohl das Schreiben neuer Programme wie auch das Erweitern oder Modifizieren vorhandener Programme verstanden.), Zusammensetzung der Komponenten und ihre Einbindung in vorhandene Software, Inbetriebnahme der Software sowie Überprüfung des Entwickelten nach jedem Schritt.

## 2.2 Software Engineering

In der Anfangszeit der Informatik gab es mit der Entwicklung von Software kaum Probleme. Dies ist nicht weiter verwunderlich, denn die Software bestand aus einzelnen, weitgehend voneinander unabhängigen Programmen, die jedes für sich eine beherrschbare Größe hatten. Die Schwierigkeiten der Software-Entwicklung manifestierten sich erst in den 60er Jahren, als immer umfangreichere Software entwickelt wurde und die bis dahin verwendeten Vorgehensweisen zunehmend versagten. Zur Überwindung dieser Software-Krise erhob Friedrich Ludwig Bauer 1968 die Forderung nach Software Engineering, d.h. einem systematischen Vorgehen bei der Entwicklung, wie es in klassischen Ingenieurdisziplinen seit langem üblich ist.

Dabei muss man wissen, dass zur damaligen Zeit allgemein die Auffassung herrschte, das Schreiben von Software sei eine Kunst und erfordere daher individuelle künstlerische Freiheit für die Programmierer. Auf diesem Hintergrund gesehen war Bauers Forderung revolutionär.

In der Zwischenzeit hat sich die Erkenntnis weitestgehend durchgesetzt, dass bei nichttrivialen Aufgaben eine wirtschaftliche und termintreue Entwicklung qualitativ guter Software ohne Software Engineering nicht möglich ist.

## 2.3 Ziele und Mittel des Software Engineerings

Mit Software Engineering werden drei grundsätzliche Ziele verfolgt.

Die Produktivität bei der Herstellung von Software soll gesteigert werden. Angesichts der horrenden Summen, die jährlich weltweit für Software ausgegeben werden, sind auch kleine Produktivitätssteigerungen von großem wirtschaftlichem Interesse.

Die Qualität der erstellten Software soll verbessert werden. Dies bringt neben Wettbewerbsvorteilen durch zufriedene Kunden auch Kostensenkungen bei der Pflege und Weiterentwicklung von in Betrieb befindlicher Software.

Die Führbarkeit von Software-Entwicklungsprojekten soll erleichtert werden.

### 3 Programmierung

In der Softwaretechnik ist vor allem die imperative Programmierung als Implementierungstechnik von besonderer Bedeutung. Unter Implementierungstechnik versteht man die Überführung der Entwurfsergebnisse in Programme. Imperative Programmierung bedeutet, dass die Zustandsänderungen eines Algorithmus durch Anweisungen erfolgen.

Ziel der imperativen Programmierung sind Konzepte, die die Herstellung von Programmen mit hoher Produktqualität ermöglichen. Weiters muss darauf geachtet werden, dass die Softwaresysteme änderbar sind und ihre Komponenten in anderen Kontexten wiederverwendet werden können.

Die Umsetzung erfolgt durch eine sprachunabhängige Programmiermethodik. Wichtiges Schlagwort ist dabei Abstraktion. Darunter versteht man eine Methode, ein Problem umzusetzen. Abstraktion bedeutet (unwichtige) Details wegzulassen und sich auf das Wesentliche zu konzentrieren. Dadurch kann man ein Problem von einer höheren Ebene aus betrachten und findet schneller eine gute Lösung, die leichter verständlich ist.

*Beispiel für eine Abstraktion:*

Ein Computer besteht aus vielen Komponenten, wie z.B. dem Mainboard, der Grafikkarte, der Festplatte usw. Wenn man nun noch weiter in die Tiefe hinabsteigt, so stellt man fest, dass, um beim Mainboard zu bleiben, dieses wiederum aus vielen verschiedenen Bauteilen und Leitungen besteht. Die Chips bestehen wiederum aus Siliziumatomen.

Man hat nun durch die Abstraktion die Möglichkeit, den Computer als eine Einheit (ein Objekt) zu betrachten, ohne etwas über die genauen Bestandteile, oder gar über die Struktur der Siliziumatome wissen zu müssen. Wenn man auf dem PC einen Brief schreiben möchte, ist es dank der Abstraktion nicht notwendig, zu wissen, wie die einzelnen Chips zusammenarbeiten.

## 3.1 Überblick

### 3.1.1 Strukturierte Programmierung

Die strukturierte Programmierung betrifft die Ausgestaltung von kleinen Programmen, d.h. einzelne Programmaufgaben werden von unabhängigen Codeabschnitten (Funktionen) ausgeführt.

Diese Programmiermethodik umfasst:

- Abstrakte Anweisungen
- Datentypen und
- Prozessabstraktion

Vorteile:

- Es ist einfacher, ein strukturiertes Programm zu schreiben, da komplexe Programmierprobleme in kleinere und leichtere Aufgaben zerlegt werden können.
- Es ist einfacher, ein strukturiertes Programm zu debuggen. Wenn das Programm einen Fehler (englisch „bug“) aufweist, der die ordnungsgemäße Ausführung behindert, kann ein strukturiertes Design die Eingrenzung des Problems auf einen bestimmten Codeabschnitt (z.B. eine bestimmte Funktion) erleichtern.
- Zeitersparnis, denn wenn man eine Funktion schreibt, die eine bestimmte Aufgabe in einem Programm lösen soll, kann diese Funktion problemlos in einem anderen Programm verwendet werden, in der die gleiche Aufgabe gelöst werden muss. Auch wenn das Problem im neuen Programm etwas anders ist, ist es einfacher, eine bereits bestehende Funktion zu ändern, als sie neu zu schreiben.

Probleme:

Zunächst ist es äußerst schwierig, die Robustheit des Codes selbst zu gewährleisten (damit meint man Aspekte wie Sicherheit vor Abstürzen, Ausfallsicherheit, Fähigkeit mit unvorhergesehenen Situationen umgehen zu können). Außerdem ist die Wartung

häufig sehr aufwendig und kostenintensiv und die Wiederverwendbarkeit stark eingeschränkt. Damit sind neue Projekte mit größerer Komplexität kaum möglich.

### **3.1.2 Datenabstraktion**

Der Begriff Datenabstraktion definiert abstrakte Datentypen durch Angabe von Funktionen. In der Programmierung ist die Realisierung in Typmodulen oder Klassen (bei objektorientierter Programmierung) von Bedeutung.

Datenabstraktion dient außerdem dazu, die Programmiersprache besser auszudrücken, sowie die Modularität der Programme zu gewährleisten. Unter dem Begriff Modularität versteht man das Anlegen von Interfaces (Schnittstellen) in einem Programm.

### **3.1.3 Programmierung im Großen**

Für große Software stellt sich die Frage nach getrennt entwickelbaren Komponenten in einem Softwaresystem. Um den Zugriff auf das Innere einer Komponente zu verhindern, werden die Deklarationen von Objekten (Konstanten und Variablen), Typen und Operationen gekapselt. Namen, die nach außen hin sichtbar sein sollen, werden an der Schnittstelle bekannt gemacht, die innere Struktur des Datentyps und die Details der Implementierung bleiben allerdings verborgen. Eine Kapselung ermöglicht Datenabstraktion (siehe Kapitel 3.2) und bildet die Grundlage jeder Softwarearchitektur.

Grundlage der objektorientierten Programmierung stellen Klassen und Vererbung dar. Klassen dienen zur Definition gemeinsamer Attribute von Objekten und ihrer Operationen.

### 3.1.4 Formale Spezifikation und Verifikation von Programmen

Formale Ansätze legen die Bedeutung von Programmen (Semantik) fest. Ausgangspunkt können dabei auch fertige Programme sein. Wichtiger jedoch ist der Einsatz formaler Ansätze zur Spezifikation.

#### *Formale Spezifikation:*

Auf Spezifikationsebene wird festgelegt, was das Programm tut, während die Implementierung vorschreibt, wie dies technisch umgesetzt wird. Der Übergang zur Implementierung kann manuell oder durch teilautomatisierte Transformationen erfolgen.

#### *Formale Verifikation:*

Die formale Verifikation weist die Korrektheit von Programmen zu ihrer Spezifikation nach.

#### *Axiomatische Semantik:*

Die axiomatische Semantik nimmt Bezug auf die allgemeinen Eigenschaften von Zuständen. Praktische Anwendung findet die axiomatische Semantik bei den wissensbasierten Systemen in der künstlichen Intelligenz (KI)

#### *Algebraische Spezifikation:*

Die Definition abstrakter Datentypen erfolgt durch algebraische Spezifikation mittels Sorten und Operationen.

#### *Teilformale Ansätze:*

Da formale Ansätze zur Spezifikation und Verifikation sehr aufwendig, und allenfalls auf der Ebene kleiner Programmkomponenten anwendbar sind, sucht man nach anderen Möglichkeiten. Eine Möglichkeit sind teilformale Ansätze. Sie machen die Prinzipien formaler Verfahren in vereinfachter Weise zugänglich.

## 3.2 Modellierung

### 3.2.1 Modellierungsmittel

Modellierungsmittel unterstützen die Entwicklung eines abstrakten Modells bei Analyse und Entwurf als Grundlage für Spezifikation und Implementierung.

Wichtige Modellierungsmittel sind:

- Kommandosprachen: Sie basieren auf der Eingabe von Befehlen in einer vorgegebenen Syntax. Daraus ergibt sich jedoch ein hoher Lernaufwand, denn ohne Kenntnis der Syntax ist eine Nutzung nicht möglich. Weiters ist die Eingabe fehleranfällig, da Befehle und Parameter erinnert werden müssen.
- Entscheidungsstrukturen: Die einfachste Form, die Reaktion von Programmen in Abhängigkeit von Ereignissen darzustellen, sind Tabellen.
- Zustandsübergänge: Hier werden endliche Automaten zur Realisierung herangezogen.
- Prozessmodellierung: Die Prozessmodellierung beschreibt den organisatorischen Rahmen für die Softwareentwicklung (zB. Reihenfolge und Phasen des Arbeitsablaufs, durchzuführende Aktivitäten, Definition der Teilprodukte, erforderliche Inputs, Mitarbeiterqualifikationen, Verantwortlichkeiten und Kompetenzen, Werkzeuge, usw.).
- Ablaufmodellierung: Sequentielle Prozesse (Abläufe) können mittels Flussdiagrammen oder mit Struktogrammen dargestellt werden.
- Datenmodellierung: Für besonders langlebige (persistente) Daten sind unter anderem das relationale Datenmodell und das Entity-Relationship-Modell von Bedeutung.

### 3.2.2 Methoden

Der Begriff Methode umfasst einerseits einzelne Aufgaben wie Analyse, Entwurf oder Programmierung, andererseits aber auch die Softwareentwicklung insgesamt. Methoden grenzen die Softwareentwicklung von anderen Tätigkeiten ab und heben spezielle Teilaufgaben (z.B. Spezifikation) und Vorgehensweisen hervor. Sie beruhen auf Zielen und Wertvorstellungen.

Der Einsatzbereich einer Methode ist die Klasse von Softwareprojekten, für die die Methode geeignet ist. Die meisten Methoden streben eine universelle Einsetzbarkeit an, daher ist die Auswahl der Methoden bereits bei der Gestaltung von Softwareprojekten eine wichtige Aufgabe.

Es ist sehr schwer, angesichts der Fülle an Methoden, diese zu ordnen. In der Praxis wird vor allem zwischen strukturierten und objektorientierten Methoden unterschieden.

#### ***Strukturierte Methoden:***

Strukturierte Methoden sind Anfang der Siebziger Jahre entstanden. Ziel ist ein standardisiertes Vorgehen nach dem Top-Down-Prinzip, d.h. der Übergang eines abstrakten Modells hin zu seiner Detaillierung und programmsprachlichen Realisierung. In der Praxis wird ein globales Datenmodell erstellt und letztendlich in einem eigenen Arbeitsschritt mit dem Funktionsmodell abgestimmt. Ein Problem stellt der Übergang zwischen den Modellebenen dar → Folge sind Modellbrüche.

#### ***Objektorientierte Methoden:***

1967 kam die erste objektorientierte Programmiersprache namens Simula auf den Markt. Mit der Verbreitung von Smalltalk nach 1980 begann dann die Entwicklung objektorientierter Methoden. Heute sind sie in ihrer Fülle kaum mehr überblickbar.

Die Objektorientierung bietet die Möglichkeit, Daten zusammen mit Operationen zu modellieren, bereits bestehende Systeme flexibel zu erweitern und zu integrieren, Produkte zu strukturieren und in Produktlandschaften zu integrieren und vieles mehr.

### 3.3 Architektur von Softwaresystemen

Definition:

*„The software architecture of a program or computing system is the structure or the structures of the system, which comprise software components, the externally visible properties of those components, and then relationships among them.“ (Len Bass)*

Elemente der Softwarearchitektur:

- Komponenten
- Beziehungen (Konnektoren)
- Eigenschaften

Fazit:

*„To be architectural is to be the most abstract depiction of the system that enables reasoning about critical requirements and constrains all subsequent refinements.“ (Mark Klein)*

Allgemein kann man sagen, dass Softwarearchitektur erreichen will, dass das Anwendungssystem die Anforderungen erfüllt, robust ist gegenüber Änderungen und eine gewisse Schönheit hat.

Wesentliche Eigenschaften einer Softwarearchitektur sind unter anderem:

- die Umsetzung des Prinzips der Datenkapselung,
- die Modul- und Klassenbildung, sowie
- die Erweiterbarkeit von Strukturen.

#### 3.3.1 Architekturprinzipien

***Geheimnisprinzip (information hiding):***

Eine der wichtigsten Grundlagen der Softwareentwicklung ist das Prinzip, dass für den Benutzer eines Softwareproduktes die interne Verarbeitungsform irrelevant sein

muss, sogar vor ihm geheim gehalten werden sollte, um einen möglichen Missbrauch der Software zu vermeiden.

***Modulkopplung und –kohäsion (coupling and cohesion):***

Modulkopplung bezeichnet die Abhängigkeit zwischen den Modulen, z.B. ausgetauschte Daten oder gemeinsam genutzte Daten. Es ist auf eine möglichst geringe Modulkopplung zu achten.

Modulkohäsion beschreibt den inneren Zusammenhalt der Module. Diese werden nach dem Eingabe-Verarbeitung-Ausgabe-Prinzip (EVA) erstellt. Ein Modul soll eine möglichst hohe fachliche und logische Kohäsion aufweisen.

***Offen-Geschlossen-Prinzip (open-close principe):***

Ein Modul ist offen, wenn es erweitert oder geändert werden kann. Ein Modul ist geschlossen, wenn es für die Benutzung mit einer festgelegten Schnittstelle und Beschreibung bereitsteht.

In der prozedurorientierten Programmierung wird die Kontrolle über den jeweiligen Zustand eines Moduls (welche Module für die Benutzung offen und welche geschlossen sind) durch Konfigurationsmanagement bestimmt. In der objektorientierten Programmierung wird dies durch den Vererbungsmechanismus gewährleistet.

***Entwurfsmuster (design patterns):***

Definition:

*„Jedes Muster beschreibt in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, sodass sie diese Lösung beliebig oft anwenden können, ohne sie jemals ein zweites Mal gleich auszuführen.“*

*(Christopher Alexander)*

### 3.3.2 Organisationsformen

#### ***Programmbibliothek:***

Unter einer Programmbibliothek versteht man eine Sammlung von selbstständigen Programmkomponenten, wie z.B. Unterprogrammen, Modulen oder Klassen, die für die Wiederverwendung gedacht sind. Bei der Verwendung von Bibliothekskomponenten bestimmt der Entwickler des Anwendungsprogramms, wie die Struktur der Komponenten aussieht (entweder neu entwickelte oder eingebundene Komponenten) und wie der Steuerfluss durch das System insgesamt ist.

#### ***Rahmen, Anwendungsrahmen:***

Ein Rahmenwerk (framework) ist ... *eine Konfiguration von Klassen, die ein Lösungsschema für eine Problemstellung vergegenständlichen.* (Johnson)

Ein Anwendungsrahmen (application framework) ist eine spezielle Form eines Rahmens, der die Grundstruktur und den Steuerfluss einer vollständigen Anwendung enthält. Er liefert eine generische Lösung für eine Klasse von Anwendungsprogrammen. Diese Lösung muss allerdings meist noch spezialisiert bzw. ergänzt werden. Zudem bieten Anwendungsrahmenwerke oft eine einheitliche Benutzungsschnittstelle, sowohl in Form der Präsentation als auch der Handhabung (look and feel). Die Komplexität großer Anwendungsrahmenwerke erfordert, dass bei ihrer Entwicklung und Verwendung softwaretechnische Architekturprinzipien eingehalten werden.

#### ***Komponenten:***

Ein aktueller Trend ist die Zusammensetzung von Anwendungen aus Komponenten.

Die Grundidee geht auf das Client-Server Prinzip zurück:

- beliebige Kunden fordern eine Dienstleistung (Client)
- der Anbieter stellt diese Dienstleistung zur Verfügung (Server)

Handelt es sich bei dem Anbieter nicht nur um einen Datenlieferanten (z.B. File Server), sondern um eine eigenständige Anwendung, spricht man von einer Komponente. Um diese Komponenten als eigene Prozesse in einem Netz nutzen zu können,

benötigt man eine vermittelnde Software (sog. Middleware), die das Auffinden, die Bindung der Prozesse und die Dienstnutzung ermöglicht.

Um Entwicklungsaufwand zu sparen, werden die Anwendungen aus dem Markt erhältlicher Komponenten zusammengesetzt. Das Ziel ist die Zusammenstellung einer angemessenen und wartbaren Anwendung aus individuellen Komponenten und kostengünstigen Standardprodukten in einem ausgewogenen Preis-Leistungsverhältnis.

Mit dem Internet eröffnet sich dieser Komponentenidee eine weitere Entwicklungsmöglichkeit. Die Software-Bausteine können über das Netz als Bestandteil von HTML-Seiten des World Wide Web geladen werden.

Inzwischen hat sich Client-Server-Computing als das vorherrschende Modell fast aller neuen Anwendungen durchgesetzt.

Vorteile:

- mehr Eigenverantwortung für die Geschäftsbereiche,
- schnellere Realisierung neuer Anforderungen
- mehr Flexibilität für den Endbenutzer, usw.

Allerdings zeichnet sich bei den bisherigen Client-Server-Modellen auch ein Nachteil ab: Die Administration und Wartung der Arbeitsplatzrechner verursacht hohe Kosten. Daher ist es anzuraten, vor dem Umstieg auf eine Client-Server-Architektur zu überlegen, wie der Betreuungsaufwand im Griff behalten werden kann.

## 4 Softwareentwicklungsprojekte

### 4.1 Überblick

Softwareentwicklung umfasst alle Tätigkeiten und Ressourcen, die zur Herstellung von Software notwendig sind. Softwareprojekte lassen sich nach verschiedenen Sichtweisen klassifizieren:

- Projektziel
- Projektgegenstand
- Anwendungsorientierung
- Organisatorischer Rahmen
- Technischer Kontext

Die jeweilige Projektsituation wird aus dem Zusammenspiel dieser Merkmale bestimmt.

#### 4.1.1 Tätigkeiten bei der Produktentwicklung

##### *Anforderungsermittlung*

Am Anfang jeder Softwareentwicklung steht die Systemanalyse. Wenn der Projektauftraggeber an den Entwickler herantritt ist es daher von besonderer Bedeutung, dass in der Projekteingangsphase der momentane Zustand genau analysiert wird um später die konkreten Maßnahmen zur Lösung des anstehenden Problems definieren zu können. Diese Analyse muss alle involvierten Prozesse, Daten und Kommunikations- sowie Organisationsstrukturen mit einbeziehen.

##### *Systemspezifikation*

Darunter versteht man die detaillierte Beschreibung der Teile eines Systems bezüglich ihrer Eigenschaften und Beziehungen untereinander. Es ist festzulegen worauf das System wirken soll, was es tun soll, unter welchen Bedingungen das System arbeiten soll, in welche Kommunikations- und in welche Organisationsstruktur das System eingebettet ist und welche Einschränkungen einzuhalten sind.

### ***Entwurf***

Bei der Systemspezifikation wird festgelegt, WAS das System tun soll und darauf aufbauend wird in dieser Phase formuliert wie dies getan werden soll (Algorithmen-Entwurf). Es erfolgt die Zerlegung in Komponenten und die Festlegung ihrer Verknüpfung.

Dadurch wird eine Verringerung der Komplexität, die Gewährleistung von Verständlichkeit, Änderbarkeit und Wiederverwendbarkeit der Komponenten und eine Vorbereitung für die Einteilung in Arbeitspakete bei der Programmierung erreicht.

### ***Implementierung***

Es besteht keine scharfe Trennung zwischen der vorhergehenden Phase und der Implementierung, da bei der Programmierung der Entwurf verfeinert wird. Der Algorithmus und die zugehörigen Datenstrukturen werden in eine geeignete Programmiersprache übertragen, wobei der Auswahl der Sprache eine große Bedeutung zukommt.

Wichtig bei der Einteilung in einzelne Arbeitspakete ist die Einhaltung von Konventionen und Standards und die Koordination der Fertigstellung der Komponenten.

Qualitätskriterien sind Effektivität, Effizienz, Zuverlässigkeit, Wartungsfreundlichkeit, Anpassungsfähigkeit und Benutzerfreundlichkeit. Diese Kriterien sollen Benutzerakzeptanz und Ausbaufähigkeit gewährleisten.

### ***Validation***

Hierzu gehören sämtliche Maßnahmen der konstruktiven Qualitätssicherung, insbesondere das Testen, das sich in folgende Stufen unterteilen lässt: Komponententest, Integrationstest, Systemtest (an Hand der Spezifikation) und Annahmetest (mit echten Anwendungsdaten).

Das Thema Testen wird ausführlich im Kapitel Softwareentwicklungsprozess – Qualitätssicherung behandelt.

### ***Systemeinführung***

Der vorläufige Abschluss der Entwicklung ist die Installation, d.h. die technische Übertragung aus der Entwicklungs- in die Einsatzumgebung. Meist wird das Programm in ein bestehendes System integriert.

Von Bedeutung sind selbstverständlich auch organisatorische Umstellungen die berücksichtigt werden müssen und die Durchführung von Benutzerschulungen.

### ***Wartung***

Diese sich an die Entwicklung anschließende Phase ist nicht nur auf Anpassung und Fehlerbereinigung ausgerichtet, sondern im wesentlichen eine Weiterentwicklungsphase, die für die Erhaltung der Qualität der Software unerlässlich ist.

## **4.1.2 Vorgehensmodelle (life cycle models)**

Software-Entwicklung wird heute als ein komplexer, iterativer Prozess verstanden. Prozessmodelle sind daher unumgänglich bei der Entwicklung moderner Software. Dies bringt mehr Aufwand für Planung und Verwaltung mit sich, allerdings kann man dadurch die Dauer der Entwicklung besser abschätzen und hat eine detaillierte Übersicht.

### ***Lineares Phasenmodell***

Die einzelnen Entwicklungsschritte werden in Phasen gegliedert, die zeitlich sequentiell durchlaufen werden und es werden Zwischenergebnisse (Meilensteine) definiert. Am Ende jeder Phase muss ein Review erfolgen und es wird darüber entschieden ob mit dem nächsten Schritte begonnen werden kann. Es handelt sich um ein ereignisorientiertes Modell.

Das **Wasserfall-Modell** war das 1. veröffentlichte Phasenmodell zur Softwareentwicklung (Royce 1970) und wurde abgeleitet vom Phasenmodell anderer Ingenieursdisziplinen.

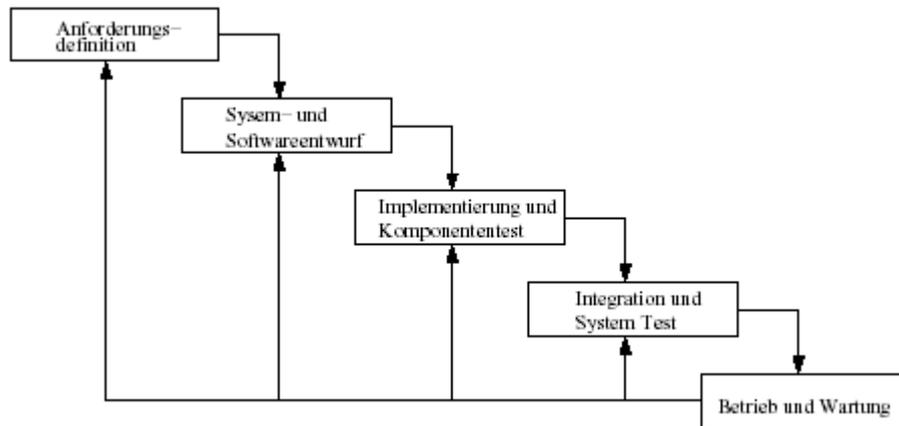


Abb.3: Wasserfall-Modell

1. Schritt: Analyse und Definition der Anforderungen  
Ergebnis: Systemspezifikation
2. Schritt: System- und Softwareentwurf  
Ergebnis: AD, DD und vernünftigerweise Testfälle
3. Schritt: Implementierung und Komponententest
4. Schritt: Integration und Systemtest  
Ergebnis: nach erfolgreichem Test Übergabe an den Kunden
5. Schritt: Betrieb und Wartung

Das Wasserfall-Modell bei dem man von einer streng getrennten und rückkopplungsfreien Abfolge der einzelnen Phasen ausging, wird dem Prozess der Software-Entwicklung nicht mehr gerecht, da sequentielle Modelle für den Auftraggeber ein relativ hohes Risiko bedeuten. In der Praxis wird daher auf häufig vorangehende Phasen zurückgegriffen.

### ***Spiralmodell***

Das Spiralmodell ist eine Weiterentwicklung des Wasserfall-Modells und wesentlich flexibler und risikoorientiert. Die Entscheidung über das weitere Vorgehen wird nach jedem Zyklus gefällt.

Die zu durchlaufenden Phasen in den einzelnen Durchläufen hängen von der Risikoanalyse ab:

- Geringes Risiko: ähnlich dem Wasserfall-Modell
- Hohes Risiko: nähert sich dem evolutionären Modell an

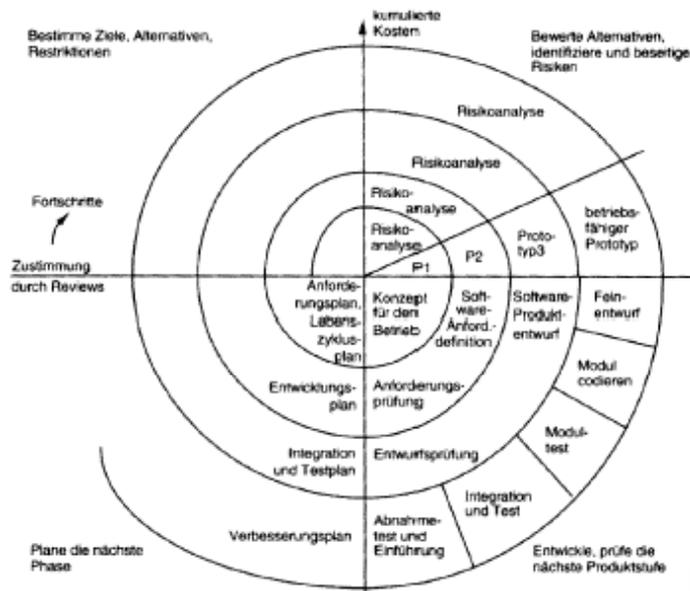
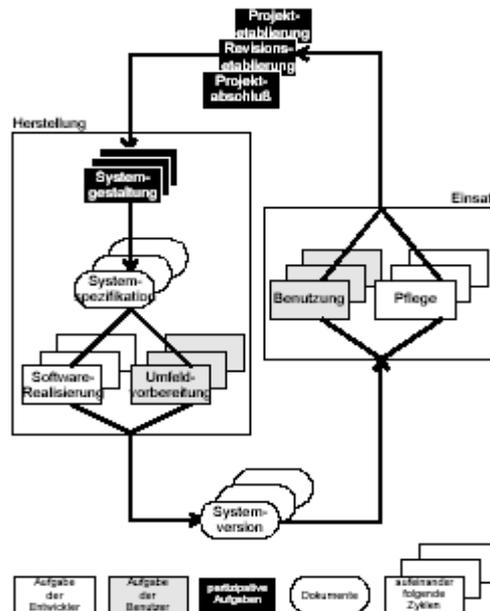


Abb.4: Spiralmodell

**Zyklisches Modell**

Die Software wird nicht mehr als ein Produkt, sondern als Folge von Entwicklungszyklen oder Versionen verstanden. In diesem Modell wird die Wartung durch Pflege



der aktuellen und Übergang zur nächsten Version ersetzt.

### 4.1.3 Evolutionäre Systementwicklung und Prototyping

Die evolutionäre Systementwicklung strebt die Überwindung der starren Beschränkungen des Phasenmodells, eine verbesserte Zusammenarbeit mit den Benutzern und eine Erhöhung der Gebrauchsgüte der Software an.

Der Entwicklungsprozess wird häufig nach den Konzepten des Prototyping durchgeführt.

#### *Prototyping*

ist ein iteratives Verfahren der Software-Entwicklung, bei der die wesentlichen Schritte mehrfach durchlaufen werden, bis die erforderliche Produktqualität erreicht ist. Ein wesentlicher Grundsatz ist die Einbeziehung der zukünftigen Nutzer in die Entwicklung eines Programms und das damit verbundene wechselseitige Lernen zwischen Entwicklern und Anwendern.

Man unterscheidet 3 Arten von Prototyping (exploratives, experimentelles und evolutionäres) anhand der Ziele.

#### *Prototyp*

Der Prototyp ist eine Vorabversion eines Softwaresystems mit eingeschränkter Funktionalität. Im Sinne von [Kieback 92] gibt es folgende Arten von Prototypen:

- Demonstrationsprototyp
- Prototyp im engeren Sinne (funktionaler Prototyp)
- Labormuster
- Pilotsystem

## 4.2 Softwareentwicklungsprozess

Organisation und Management von Projekten sind ein zentrales Anliegen der Softwaretechnik. Ein gut organisierter Softwareentwicklungsprozess leistet einen Beitrag zur Steigerung der Wettbewerbsfähigkeit durch eine Verbesserung der Produktqualität, eine Steigerung der Produktivität und eine Verkürzung der Entwicklungszeiten.

### 4.2.1 Beteiligte bei der Softwareentwicklung

Akteure im Softwareentwicklungsprozess sind der Auftraggeber und der Softwareproduzent. Hier treffen anwendungsfachliches und softwaretechnisches Wissen aufeinander und man muss sich darüber im Klaren sein, dass damit tiefgreifende betriebliche Veränderungen einher gehen können. Die unterschiedlichen Interessen und Ziele der Personengruppen die an diesem Prozess beteiligt sind sowie ökonomische Erfordernisse müssen berücksichtigt werden.

Rollenhierarchie:

- Auftraggeber und Softwarehersteller:  
Sie sind rechtlich und auch organisatorisch gesehen die Vertragspartner eines Softwareprojektes.
- Entwickler:  
umfasst alle Mitglieder des Entwicklungsteams, wie Analytiker, Designer, Programmierer. Sie besitzen das softwaretechnische Know-How. Die Verantwortung hingegen trägt das Entwicklungsmanagement und nicht die Teammitglieder selbst (siehe Kapitel 4.2.3)
- Anwender:  
Sie setzen das Softwaresystem (direkt oder indirekt) ein und repräsentieren das Anwendungswissen. Zu betonen ist, dass die Entscheidung über die Verwendung des Systems im Einsatzkontext, beim Anwendungsmanagement liegt.

### 4.2.2 Leitbilder der Softwareentwicklung

Die Leitbilder spielen bei der Softwareentwicklung eine wichtige Rolle. Ein Leitbild beschreibt die jeweilige Sichtweise für die Gestaltung von Software und bestimmt somit, wie die Realität wahrgenommen, verstanden und gestaltet wird. Auch für den Entwicklungsprozess selbst haben Leitbilder eine Bedeutung. Ein alternatives Leitbild stellt Design dar. Hier zählt vor allem die Kommunikation zwischen allen Beteiligten und die Einbettung der Software in einen veränderlichen Einsatzkontext.

### 4.2.3 Kooperation und Koordination

#### *Teammodelle*

Teammodelle beschreiben Formen der Zusammenarbeit bei der Entwicklung von Software. Sie erläutern die Aufgabenbereiche eines Projektleiters, Entwicklers, Testers, Projektverwalters, usw. Man muss jedoch unterscheiden zwischen einem hierarchisch gegliederten Chefprogrammiererteam, wo dem Projektleiter die Gesamtverantwortung übergeben wird (→ Top-Down-Vorgehensweise) und einem demokratischem Team. Hier wird die Verantwortung unter den gleichberechtigten Teammitgliedern aufgeteilt, was soviel heißt wie, jedes einzelne Teammitglied besitzt einen klar definierten Arbeitsbereich und trägt die Verantwortung dafür.

#### *Koordination des Entwicklungsprozesses*

Unter dem Begriff Projektmanagement versteht man unter anderem die Kommunikationssicherung im Team, die Fertigstellung und laufende Überprüfung von Zwischenergebnissen, Gewährleistung der Termineinhaltung, sowie Qualitätssicherung. Im ersten Schritt werden Ziele vereinbart und die Grundlagen für eine gute Zusammenarbeit geschaffen (Projektetablierung). Im Laufe des Projekts werden anhand von sogenannten Referenzlinien und/oder strukturierten Reviews die Zwischenergebnisse überprüft. Projektstadien oder Meilensteine dienen der Überprüfung von Projektzielen und Projektfortschritt, sowie der Planung der weiteren Vorgehensweise.

### ***Kooperation mit Anwendern***

Von Bedeutung ist hier vor allem die Klärung des Mitspracherechts aller Beteiligten und zu überlegen, wie sich deren Qualifikationen verbessern lassen. Dazu stehen verschieden Organisationsmodelle zur Verfügung, die von der jeweiligen Projektsituation abhängen, z.B. partizipative Projekte mit direkter Entscheidungskompetenz von Benutzern, die Einrichtung eines Anwenderbetreuers für Benutzerbelange oder aber auch die Bildung von User-Groups, die die Interessen der Anwender vertreten. Ziel ist es daher, die Gebrauchsqualität von Software zu gewährleisten.

#### **4.2.4 Produkt- und Konfigurationsverwaltung**

Ein Softwareprodukt setzt sich im allgemeinen aus Programmen und dem zu definierenden Texten (Dokumenten) zusammen.

Die Verwaltungsaufgabe eines entstehenden Produkts beginnt bereits beim Entwurf. Hierzu führt der Projektverwalter eine sogenannte Projektbibliothek ein. Während des Entwicklungsprozesses wird z.B. zwischen der Arbeitsversion, der Testversion und der freigegebenen Version unterschieden, die einer geordneten Zustandsüberführung unterworfen werden. Nach der Auslieferung muss die Bildung von lauffähigen Konfigurationen nach verschiedenen Aspekten gewährleistet sein.

#### **4.2.5 Qualitätssicherung**

Unter Qualitätssicherung versteht man alle Tätigkeiten und Maßnahmen um Gebrauchsqualität, die Qualität von Prozessen und Produkten zu gewährleisten. Einen besonderen Stellenwert in der Softwareentwicklung hat die konstruktive Qualitätssicherung, die auf dem Einsatz von Methoden, Werkzeugen und Maßnahmen zur Wiederverwendung qualitativ hochwertiger Komponenten basiert.

Bei der analytischen Qualitätssicherung wird das Qualitätsniveau der entwickelten Softwarekomponente geprüft und gemessen, dazu gehören die statische Analyse (Überprüfung mittels Einsatz von Metriken, Inspektionen, Reviews) und die dynamische Analyse (Testen).

Beim Testen wird ein Programm auf systematische Art ausgeführt um Fehler zu finden. Man unterscheidet:

- Black-Box-Test (funktionsbezogenes Testen)
- White-Box-Test (Codebezogenes oder strukturelles Testen)
- Grey-Box-Test (schnittstellenbezogenes Testen)

Der White-Box-Test ist für den Entwickler von Bedeutung, der den Komponententest durchführt. Weitere Teststufen sind der Integrationstest, der Systemtest (der die Spezifikation sicherstellen) und schließlich der Abnahmetest (in der technischen Umgebung mit echten Daten des Einsatzkontextes).

*„Testen ist destruktiv. Es zeigt nicht die Korrektheit eines Programms, sondern seine Inkorrektheit.“ (Dr. Friederike Nikl)*

*„Testen kann nur die Anwesenheit von Fehlern zeigen, niemals jedoch deren Abwesenheit.“ (Dijkstra, 1975)*

*„(Der relative Korrektheitsbeweis) kann und soll lediglich demonstrieren, dass der Testling bestimmten, in Form von Testfällen festgelegten Anforderungen genügt. Die Güte des Tests hängt somit ganz und gar von den Testfällen ab.“ (Denert, 1991)*

### **Konstruktive Kritik**

Informelle Analyseverfahren (z.B. Reviews, Audits, Walk-Throughs) bei denen spezifizierende und dokumentierende Texte im Entwicklerteam oder von Gutachtergremien bewertet werden spielen in der Praxis eine große Rolle.

### ***Evaluation***

Die Evaluation der Qualität ist eine zentrale Aktivität bei der Entwicklung jedes Systems. Ohne Evaluation ist es nicht möglich zu wissen, ob ein System die gestellten Anforderungen erfüllt. Evaluation der Gebrauchstauglichkeit bedeutet das Sammeln von spezifischen Kennzahlen eines Produktes bei einer bestimmten Gruppe von Benutzern für eine bestimmte Aktivität innerhalb eines bestimmten Arbeitskontextes.

#### **4.2.6 Sicherung der Prozessqualität**

In den letzten Jahren sind Ansätze vorgeschlagen worden, die sich auf die Qualität des Prozesses selbst beziehen. Für die Praxis sind das Reifemodell des Softwareprozesses und vor allem die Bestimmung der Normen unter der allgemeinen Bezeichnung ISO 9000.

#### ***Reifemodell des Softwareprozesses (capability maturity model):***

Mit Hilfe dieses Modells soll eine Verbesserung des Softwareprozesses – die gesamten Tätigkeiten bei der Softwareentwicklung – systematisch erreicht werden. Das wesentliche Merkmal des capability maturity model (CMM) sind die sogenannten Reifegrade (maturity levels). Sie definieren je eine Stufe des Softwareprozesses – „anfänglich“, „wiederholbar“, „definierbar“, „verwaltbar“ und „optimierbar“ und legen fest in welchem Maß die Softwareentwicklung zu definieren, zu planen, zu regulieren und zu kontrollieren ist. Die Bedeutung des Reifemodells für die europäische Softwareentwicklung ist derzeit eher gering und auch für die USA noch schlecht einzuschätzen.

#### ***ISO 9000 (International Organization of Standardization):***

ISO 9000 stellt eine Menge von Anforderungen an das Qualitätsmanagementsystem die sicherstellen sollen, dass der beschrittene Prozess im Stande ist, konsistent Produkte hervorzubringen, die die Erwartungen der Kunden erfüllen.

Für die Softwareentwicklung sind vor allem die Normen 9000 (Begriffssystem und Anwendungshinweise) und 9004 („Leitfaden für Dienstleistungen“) wichtig. Der Softwarehersteller kann sich bei einer Institution, die dafür autorisiert ist, nach ISO 9000 zertifizieren lassen.

### 4.3 Softwareentwicklungswerkzeuge

Softwareentwicklungswerkzeuge (software tools) dienen zur Unterstützung und Teilautomatisierung der Softwareentwicklung.

#### *Einzelwerkzeuge*

Ursprünglich wurden nur einzelne grundlegende Komponenten wie Compiler, Editoren und Testhilfen als Werkzeuge bezeichnet. Heute zählen viele spezialisierte Entwicklungs- und Administrationswerkzeuge die zur Unterstützung in den Bereichen der Herstellung und Verarbeitung von Dokumenten, Konsistenzprüfung von einzelnen Dokumenten und des gesamten Druckbestandes, Unterstützung einzelner Entwicklungsschritte, Umsetzung einer Methode, Produktverwaltung während der Herstellung und bei der Weiterentwicklung, dazu. Beispiele hierfür sind Benutzeroberflächengenerator, Code-Generator, Very High Level Languages, Programmanalysewerkzeuge, Debugger, Dokumentationswerkzeuge und Konfigurationsverwaltungswerkzeuge.

#### *Entwicklungsumgebung*

Aufgrund der hohen Anzahl und der Inkompatibilität der software tools wurden ab den siebziger Jahren Softwareentwicklungsumgebungen konzipiert. In der kommerziellen Datenverarbeitung wurde um das Datenlexikon als eine Entwicklungsdatenbank ein Satz von Werkzeugen zur Analyse, Konstruktion und Verwaltung von Datenbank Anwendungen entwickelt. Zur Entwicklung großer technischer Softwaresysteme wurden integrierte Werkzeugumgebungen konstruiert. In der Praxis werden Entwicklungsumgebungen eingesetzt, die auf eine Programmiersprache oder ein Datenbanksystem zugeschnitten sind.

#### *CASE-Tools (Computer Aided Software Engineering)*

Das Entwicklungswerkzeug kann auf verschiedene Methoden und die individuellen Anforderungen des Anwendungsbereiches und der Entwickler angepasst werden. Auf dem Markt finden sich zahlreiche CASE-Werkzeuge. Diese sind zumeist auf einzelne Softwareentwurfsmethoden beschränkt und berücksichtigen nur einige Aspekte im Softwareentwurfsprozess. Als Problem bei CASE-Tools zeigt sich die mangelnde Eignung für die konstante Weiterentwicklung großer Softwaresysteme.

## Zusammenfassung

Es gibt zwei Arten der *Programmierung*:

- strukturierte Programmierung und
- objektorientierte Programmierung.

In der Praxis ist vor allem die objektorientierte Programmierung von Bedeutung, da es bei der strukturierten Programmierung äußerst schwierig ist, die Robustheit des Codes zu gewährleisten. Außerdem ist die Wartung häufig sehr aufwendig und kostenintensiv und die Wiederverwendbarkeit stark eingeschränkt.

Für große Softwaresysteme stellt sich die Frage nach getrennt entwickelbaren Komponenten. Um den Zugriff auf das Innere einer Komponente zu verhindern, werden die Deklarationen von Objekten, Typen und Operationen gekapselt.

Im Bereich der *Modellierung* wird mit Hilfe sogenannter Modellierungsmittel die Entwicklung eines abstrakten Modells unterstützt.

Die *Softwarearchitektur* soll erreichen, dass das Anwendungssystem die Anforderungen erfüllt, robust ist gegenüber Änderungen und eine gewisse Ästhetik aufweist.

*Softwareentwicklung* vereint in sich alle Tätigkeiten, die zu einem funktionierenden Softwaresystem führen und wird in Form von Projekten durchgeführt. Mit Hilfe von Vorgehensmodellen wie das Phasenmodell, das Spiralmodell und das Zyklische Modell werden produktbezogenen Tätigkeiten benannt und geordnet.

Beim *Softwareentwicklungsprozess* werden die Rollen der Beteiligten, die Form und die Gestaltung des Produktes, Verantwortungsbereiche, Organisationsform, Produkt- und Dokumentenverwaltung genau definiert und auf eine Sicherung der Qualität geachtet.

*Software tools* dienen zur Unterstützung und Teilautomatisierung der Softwareentwicklung.

## Literaturverzeichnis

### Bücher

- Andersen, N.E.; Kensing, F.; Lundin, J.; Mathiassen, L.; Munk-Madsen, A.; Rasbech, M.; Sorgaard, P.: Professional systems development (1990)
- Dahl, O.-J.; Dijkstra, E.W.; Hoare, C.A.R.: Structured programming (1972)
- Johnson, R.; Foote, B.: Designing reusable classes (1988)
- Wegner, P.: Concepts and paradigms of object-oriented programming (1990)

### Internetadressen

<http://homepages.fh-giessen.de>

<http://home.nwn.de>

<http://siskin.pst.informatik.uni-muenchen.de>

<http://webdoc.gwdg.de>

<http://www.bruegge.in.tum.de>

<http://www.cpp-entwicklung.de>

<http://www.gwdg.de>

<http://www.gm.fh-koeln.de>

<http://www.htl-tex.ac.at>

<http://www.informatik-uni-erlangen.de>