Softwaretechnik

Defizite und Chancen aus Benutzersicht.

von Reinhard Keil-Slawik

Definition

Softwaretechnik (engl. Software Engineering) bezeichnet die Anwendung von Prinzipien. Methoden und Techniken auf den Entwurf und die Erstellung von Programmen und Programmsystemen. Gegenüber der Programmierung spricht man dann von Software-Entwicklung. wenn mehrere Personen an der Entwicklung beteiligt sind und wenn mehr als eine Version des Programms produziert wird.

Inhalt

1	**Warum gibt es ein Fachgebiet Softwaretechnik?**	**92**
2	**Wozu dient Softwaretechnik?**	**93**
2.1	Worum geht es bei der Projektorganisation?	94
2.1.1	Das Phasenmodell	94
2.1.2	Die Probleme des Phasenmodells	97
2.2	Was leisten Methoden?	99
3	**Was bringen neuere Ansätze?**	**102**
4	**Wichtig für Arbeitnehmer**	**105**
4.1	Worauf sollte man bei der Software-Entwicklung achten?	105
4.2	Was sollte man von Technikern erwarten?	106
4.3	Zusammenfassung	107

1 Warum gibt es ein Fachgebiet Softwaretechnik?

Die Softwarekrise

Die grundlegenden Probleme der Software-Erstellung wurden zuerst im militärischen Bereich deutlich, da er enormen Anforderungen an die Leistungsfähigkeit von Software stellt: Software wurde zu spät ausgeliefert, enthielt (zu) viele Fehler und war vom Funktionsumfang her unvollständig. Sie entsprach nicht den Erfordernissen der Anwender und mußte deshalb nachträglich verbessert und erweitert werden. Die endgültigen Kosten lagen dementsprechend weit über dem ursprünglich veranschlagten Betrag. Man sprach von der Softwarekrise.

Ein neues Fachgebiet entsteht

Im Wissenschaftsausschuß der NATO kam man daher zu dem Schluß, daß diese Probleme nur im Rahmen einer entsprechenden Grundlagenforschung gelöst werden könnten. Nach mehreren vom Militär vorbereiteten Konferenzen wurde diese Initiative von Informatikern aufgegriffen; auch im zivilen Bereich häuften sich die Softwarekatastrophen.

Stand der Kunst

In den zwanzig Jahren, seit auf der NATO-Konferenz in Garmisch-Partenkirchen 1968 der Begriff ***Software Engineering*** (Softwaretechnik) als Ausdruck der Hoffnung geprägt wurde, es möge doch bitteschön eine ingenieurmäßige Herangehensweise zur Softwareproduktion geben, hat sich an diesen Problemen wenig geändert. Auch heute noch gilt der Satz: Die Qualität von Software erweist sich erst im Einsatz. Eine bestürzende Erkenntnis, denn dann ist es meist zu spät, noch etwas zu verändern.

Zwar sind seitdem viele Unzulänglichkeiten im Bereich der systematischen Programmierung erkannt und beseitigt worden, doch als eine wissenschaftlich begründete Ingenieurdisziplin kann die Softwaretechnik auch heute noch nicht charakterisiert werden.

2 Wozu dient Softwaretechnik?

Das wesentliche Ziel besteht darin, sowohl die Projektabwicklung als auch die Programme selbst zuverlässiger zu gestalten. Eine klare Strukturierung der Programme und die Gliederung des Projektablaufs sollen ein systematisches Vorgehen ermöglichen, das durch die Verwendung geeigneter Hilfsmittel und Werkzeuge effektiv unterstützt wird.

Bausteine in der Softwaretechnik

Ein erstes Ergebnis dieser Bemühungen war die ***strukturierte Programmierung***. Mit ihrer Hilfe soll ein Programm so entwickelt werden, daß es aus einer Folge von einzelnen, in sich abgeschlossenen «Bausteinen» besteht, die auf eine gut definierte und überschaubare Art und Weise zusammenwirken. Gefordert wurde zusätzlich, daß die Programmstruktur die Aufteilung des Problems in Teilprobleme widerspiegeln sollte, um die Durchschaubarkeit zu erhöhen.

Dimensionen der Softwaretechnik

Doch nicht nur bei der Programmierung, sondern auch auf der Ebene der Projektorganisation war es notwendig, einzelne Arbeitsschritte und Aufgaben voneinander abzugrenzen, um die Arbeit der beteiligten Programmierer **koordinieren** und **kontrollieren** zu können. Im Zusammenhang mit der strukturierten Programmierung wurden deshalb in der Fachliteratur zum ersten Mal auch die drei Dimensionen der Softwaretechnik benannt: **Werkzeuge**, **Projektorganisation** und **Methoden**.

Werkzeuge sollen den Arbeitsaufwand bei der Programmerstellung und der Dokumentation reduzieren. Da zum Verständnis ihrer Arbeitsweise detaillierte Kenntnisse der Softwaretechnik erforderlich sind, werden sie hier nicht weiter behandelt.

2.1 Worum geht es bei der Projektorganisation?

An der Erstellung von Software sind mehrere Menschen über einen längeren Zeitraum beteiligt. Um die Arbeit vernünftig aufteilen zu können, müssen **Aufgaben**, **Rollen** und **Entscheidungsbefugnisse** definiert werden, die festlegen, wer wann für was zuständig ist.

Zur Kontrolle des Projektfortschritts ist darüber hinaus eine zeitliche Gliederung der Arbeitsschritte erforderlich. Zu diesem Zweck wurde Anfang der siebziger Jahre das Phasenmodell (auch ***Phasenschema***) eingeführt, das bis heute allgemein zur Strukturierung von Softwareprojekten angewandt wird.

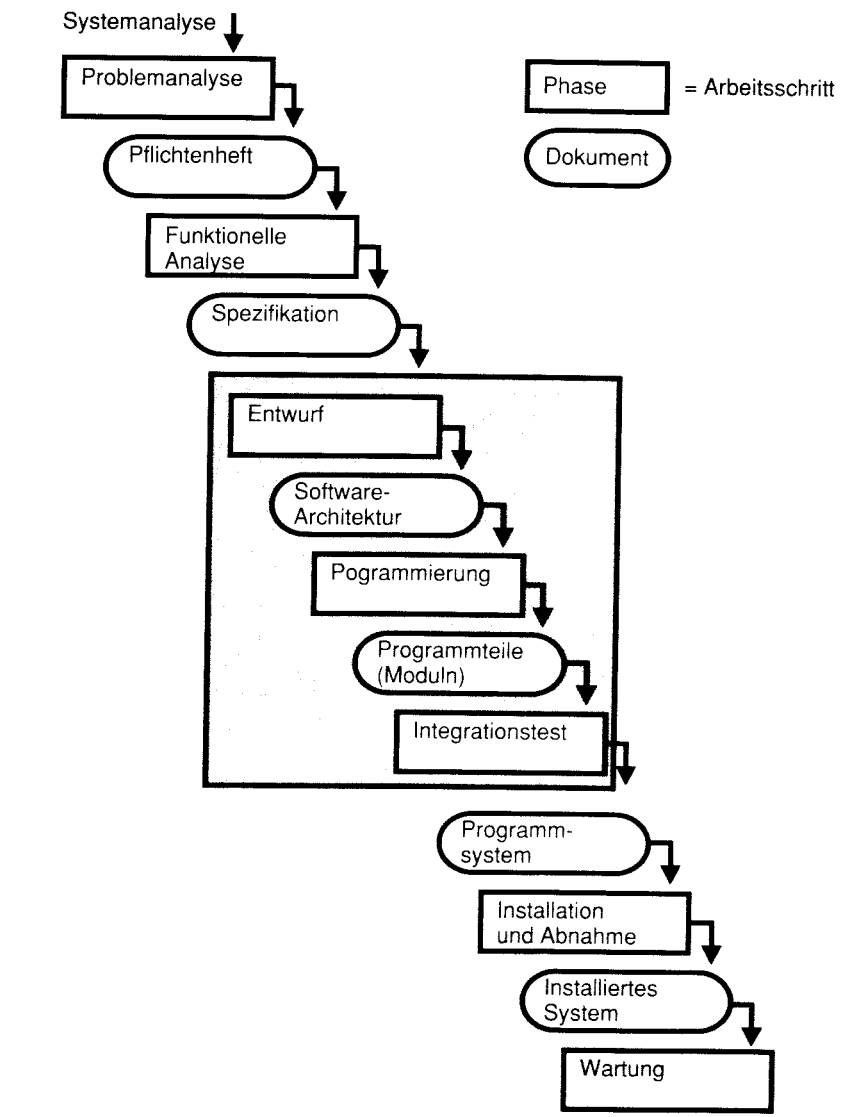
2.1.1 Das Phasenmodell

Mit Hilfe des Phasenmodells wird die Entwicklung in eine Reihe von jeweils aufeinanderfolgenden und in sich abgeschlossenen Einzelschritten (Phasen) unterteilt. In jeder Phase wird das gesamte zu erstellende System unter einem bestimmten Blickwinkel bearbeitet. Das Ergebnis wird in einem Dokument niedergelegt, das als Ausgangspunkt für den nächsten Arbeitsschritt dient.

Phasenmodelle sind eingeführt worden, weil sie

- eine vertragliche Grundlage für die Software-Entwicklung bieten,
- bei einem sorgfältigen Entwurf die Qualität der Software verbessern helfen und Fehler z.T. früher erkannt werden können,
- Zwischenergebnisse im Herstellungsprozeß definieren und damit
- eine Grobabschätzung für die Terminplanung und den Aufwand geben.

Bild 7:
Das traditionelle Phasenmodell der Software-Entwicklung.



Es gibt zwar eine Vielzahl von im Detail unterschiedlichen Phasenmodellen, doch stimmen sie in ihrer Grundstruktur mit dem in Bild 7 angegeben Schema überein.

Problemanalyse

Ausgangspunkt für die Software-Entwicklung ist die Problemanalyse, in der die Anforderungen an die zu erstellende Software ermittelt werden soll. Das Ergebnis dieser Phase ist die Anforderungsdefinition, die in der Praxis auch als ***Pflichtenheft*** bezeichnet wird. Dieses Dokument beschreibt, was das DV-System leisten soll und ist die vertragliche Grundlage zwischen Hersteller und Anwender.

Was in der Problemanalyse im einzelnen passiert, hängt von der jeweiligen Situation ab. So können die zu lösenden Probleme bereits in einer vorangegangenen Systemanalyse definiert worden sein oder aber Problemdefinition und Anforderungsermittlung werden in einem durchgeführt (vgl. 1 Software. ***Systemanalyse***).

Trennung von WAS und WIE

Die im Pflichtenheft aufgeführten Anforderungen sollten weitestgehend unabhängig von der technischen Realisierung beschrieben sein; d.h. sie sollten angeben, WAS das DV-System leisten soll (Problem), ohne festzulegen, WIE dies technisch umgesetzt werden soll (Lösung). Die Konzeption der technischen Lösung in Form einer Spezifikation wird in der nachfolgenden Phase erarbeitet. Allerdings wird diese Unterscheidung häufig nicht berücksichtigt. Probleme und Lösungsvorschläge werden im Pflichtenheft gemischt, so daß es schwer fällt festzustellen, ob eine angemessene Lösung für das richtige Problem beschrieben wird.

Funktionelle Analyse

Nach der Anforderungsermittlung muß das zu entwickelnde Produkt spezifiziert werden. Generell beschreibt eine Spezifikation die wesentlichen Eigenschaften eines Produktes und sagt aus, unter welchen Bedingungen es ordnungsgemäß eingesetzt werden kann. Bei der Software-Entwicklung handelt es sich dabei im wesentlichen um eine Beschreibung der ***Systemfunktionen*** und Teilfunktionen, die Definition der ***Benutzerschnittstelle*** sowie die Aufstellung eines ***Datenmodells***, das angibt, wie die zu verarbeitenden Daten gespeichert und verwaltet werden sollen. Bereits zu diesem Zeitpunkt muß auch festgelegt werden, auf welcher Hardware die Programme später laufen sollen, da diese Wahl wesentliche Rahmenbedingungen und Einschränkungen für die Software-Entwicklung setzt.

Realisierung

In den nachfolgenden Schritten der Realisierung geht es darum, das Gesamtsystem zu entwerfen, d.h. in unabhängig voneinander zu bearbeitende Teile zu zerlegen (Software-Architektur), diese Teile zu programmieren und, soweit das möglich ist, für sich zu testen. Nachdem alle Programmteile fertiggestellt sind, werden sie zu einem Gesamtsystem zusammengefaßt. Im Rahmen des Integrationstests wird jetzt festgestellt, ob alle Bausteine ihre Funktion korrekt erfüllen und ordnungsgemäß zusammenwirken.

Installation

Die eigentliche Produktion wird mit der Installation des Systems beim Anwender und der Übergabe der technischen Dokumentation und der Handbücher abgeschlossen.

Wartung

In der Regel schließt der Anwender mit dem Hersteller einen **Wartungsvertrag** ab. Der Begriff **Wartung** wird hier aber irreführend gebraucht, denn Software weist im technischen Sinne keinen Verschleiß auf. Statt dessen fällt unter Wartung die Beseitigung von Fehlern, die erst beim Einsatz des Systems erkannt werden, die Modifikation von Programmteilen beispielsweise zur Optimierung der Verarbeitungsgeschwindigkeit sowie Modifikationen und kleinere Ergänzungen, die sich aus veränderten Einsatzbedingungen ergeben.

Obwohl das Phasenmodell den definierten Zielen teilweise durchaus gerecht wird, reicht es als grundlegendes Projektmodell nicht aus.

2.1.2 Die Probleme des Phasenmodells

In der durch das Phasenmodell vorgegebenen linearen Vorgehensweise sind einige Annahmen enthalten, die sich in der Praxis als problematisch erweisen.

Softwaregenerationen

Die Untersuchung von Softwaresystemen, die über viele Jahre eingesetzt werden, hat ergeben, daß die ***Lebensdauer*** von Software (engl. ***Life Cycle***) nicht dem linearen Phasenschema entspricht. Tatsächlich handelt es sich um eine Folge von ausgelieferten Systemversionen, die auch als **Generationen** bezeichnet werden.

Selbst bei Standardaufgaben wie, z.B. der Textverarbeitung, ist meist erst die zweite oder dritte Generation akzeptabel, weil nur durch den praktischen Einsatz eines DV-Systems die vielen Unzulänglichkeiten und Fehler entdeckt werden.

Unzureichende Kommunikation

Dieses Phänomen wird durch ein weiteres Problem verschärft: Hersteller und Benutzer kommunizieren - soweit überhaupt - nur zu Beginn der Entwicklung miteinander. Zu diesem Zeitpunkt sind jedoch viele der späteren Wirkungen noch nicht bekannt. Da während der Realisierung keine Verständigung mit den Benutzern oder eine praktische Erprobung vorgesehen ist, haben Mißverständnisse und eigenständige Entscheidungen des Projektteams oftmals verheerende Auswirkungen.

Dokumente als Ersatz für Erfahrung

Die Dokumente, die als Zwischenergebnisse im Phasenmodell produziert werden, sind als Grundlage für eine Bewertung des Systems nicht ausreichend. Zum einen beschreiben sie nicht, was ist, sondern was sein soll. Zum anderen können wichtige Aspekte wie das Anwortzeitverhalten oder die Systemreaktion im Fehlerfall nicht oder nur unzureichend dargestellt werden.

Eine Beschreibung ist immer statisch und unanschaulich; sie kann die Erfahrung am System nicht ersetzen. Deshalb können die Benutzer erst nach der Installation und Abnahme eines lauffähigen Systems überprüfen, ob die Software ihren Vorstellungen und Anforderungen entspricht.

Bewertungen kommen zu spät

Wichtig ist aber nicht nur, wann und wie oft eine Kommunikation stattfindet, sondern auch, worüber dabei geredet wird. In der Problemanalyse müßte eigentlich ermittelt werden, welche Arbeitsaufgaben in welchem Umfang durch Software unterstützt werden sollen (vgl. 1 Software.***Arbeitsgestaltung***). Das Pflichtenheft als Ergebnis dieser Phase beschreibt aber bereits eine technische Lösung in Form von Anforderungen an das DV-System. Eine Gegenüberstellung von Problem und technischer Lösung wird nicht vorgenommen. Die Frage nach der angemessenen Unterstützung der Arbeitsaufgaben durch die Software kann man folglich erst behandeln, wenn das Entwicklungsteam mit seiner Arbeit fertig ist.

Diese Probleme werden noch dadurch verstärkt, daß die praktisch eingesetzten Methoden zur Softare-Entwicklung fast ausschließlich die durch das Phasenmodell gemachten Annahmen zugrundelegen.

2.2 Was leisten Methoden?

Eine Methode ist von der Wortbedeutung her der Weg zu etwas hin. Eine Methode soll also helfen, die Gedanken zu ordnen, indem sie den Blick auf das jeweils Wesentliche richtet und eine mehr oder minder deutlich ausgeprägte Anleitung gibt, in welcher Reihenfolge einzelne Arbeitsschritte zu erledigen sind.

Graphische Darstellung von Sachverhalten

Da Programme wie auch die beschreibenden Dokumente aus Text bestehen, liegt ein Schwergewicht bei fast allen in der Praxis eingesetzten Methoden auf der graphischen Darstellung bestimmter Sachverhalte. Je nach dem, was jeweils beschrieben oder modelliert werden soll, haben sie Namen wie (Daten-)Flußdiagramme, Datagramme, Infogramme, Objektstrukturdiagramme, Ablaufpläne, Interaktionsnetze usw.

Zur Veranschaulichung solcher Diagramm- bzw. Netzsprachen werden die wesentlichen der in der Methode SADT (Strukturierte Analyse und Entwurfs-Technik) verwendeten Elemente in Bild 8 erläutert. Entscheidend ist aber weniger, wie die graphische Aufbereitung im einzelnen funktioniert, sondern vielmehr, wie die zugrundeliegenden Konzepte aussehen und welche Sichtweise die Entwickler damit verknüpfen.

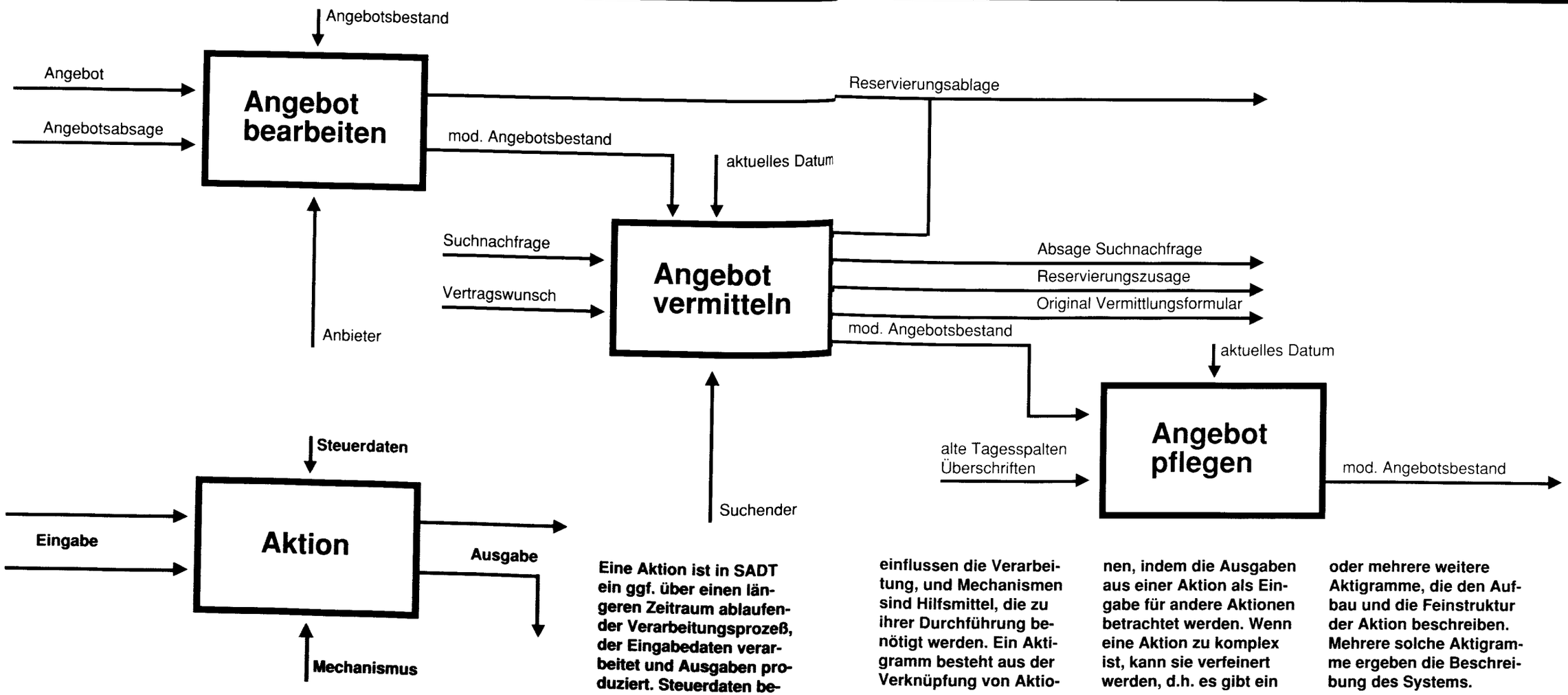
Funktionen und Algorithmen statt Arbeitsaufgaben

Generell kann man feststellen, daß die in der Praxis eingesetzten Software-Entwicklungsmethoden darauf angelegt sind, Eigenschaften und Charakteristika von Maschinen, d.h. ***Funktionen***, ***Algorithmen*** und ***Datenmodelle***, zu beschreiben. Die Analyse, Beschreibung und Bewertung von Arbeitsaufgaben und die Einbettung der Systemfunktionen in die Arbeitsabläufe wird ebensowenig unterstützt wie die Entwicklung der Benutzerschnittstelle.

Das traditionelle Methodenrepertoire der Softwaretechnik erweist sich folglich als unzureichend, wenn man eine an den Interessen und den Möglichkeiten der Benutzer orientierte Systementwicklungsstrategie umsetzen will.

1 Software . **Softwaretechnik**

Bild 8: SADT-Diagramme (Beispiel: Aktigramm zur Vermittlung von Mitfahrgelegenheiten)



Eine Aktion ist in SADT ein ggf. über einen längeren Zeitraum ablaufender Verarbeitungsprozeß, der Eingabedaten verarbeitet und Ausgaben produziert. Steuerdaten beeinflussen die Verarbeitung, und Mechanismen sind Hilfsmittel, die zu ihrer Durchführung benötigt werden. Ein Aktigramm besteht aus der Verknüpfung von Aktionen, indem die Ausgaben aus einer Aktion als Eingabe für andere Aktionen betrachtet werden. Wenn eine Aktion zu komplex ist, kann sie verfeinert werden, d.h. es gibt ein oder mehrere weitere Aktigramme, die den Aufbau und die Feinstruktur der Aktion beschreiben. Mehrere solche Aktigramme ergeben die Beschreibung des Systems.

3 Was bringen neuere Ansätze?

Software-Entwicklung erfolgt letztlich immer in Zyklen von Analyse, Entwurf, Einsatz und Auswertung. Deshalb bemüht man sich in der Softwaretechnik seit einigen Jahren verstärkt darum, in Abkehr vom linearen Phasenmodell eine prozeßbezogene Entwicklungsstrategie auszuarbeiten, die von vornherein Entwicklungszyklen und Überarbeitungen (Revisionen) einplant. Eine solche Vorgehensweise hat zwei wesentliche Vorteile. Sie erkennt an, daß

- es nicht möglich ist, alle Anforderungen zu Beginn einer Entwicklung zu erheben, sondern nur auf der Grundlage auswertbarer Zwischenergebnisse qualitativ hochwertige Software entwickelt werden kann, und
- sowohl für die Entwicklung als auch für die Benutzung das Wissen in den Köpfen der Menschen entscheidend ist, und daher Kommunikations- und Lernprozesse geeignet unterstützt werden müssen.

Dies kann durch mehrere, sich teilweise ergänzende oder unterstützende Ansätze verwirklicht werden: den Bau von Prototypen (Prototyping), eine Entwicklung in Ausbaustufen (inkrementelle Systementwicklung), die Verwendung von ***Sprachen der 4. Generation*** (4GL) und das Entwickeln von Teillösungen durch die Benutzer (Benutzer-/Anwenderprogrammierung).

Prototyping

Die Entwicklung von Prototypen zielt darauf ab, so früh wie möglich eine (erste) auswertbare Systemversion (***Prototyp***) zu haben, die unterschiedlich ausgebaut sein kann: Es kann z.B. ein voll lauffähiges System, eine Vorversion oder auch nur eine vorführbare Benutzerschnittstelle sein.

Das Entscheidende bei der Entwicklung von Prototypen ist nicht der Prototyp selbst, sondern die **sorgfältige Auswertung**. Dies ermöglicht es sowohl den Entwicklern als auch den Benutzern, sich ein Verständnis des zu entwickelnden Systems und seiner Einbettung in die Arbeitsaufgaben zu erarbeiten. Verfehlte Erwartungen und unzureichende Lösungsvorschläge können so frühzeitig erkannt und korrigiert werden.

Inkrementelle Systementwicklung

Eine Vorgehensweise in Ausbaustufen versucht, die Vorteile der Prototypenentwicklung auf den gesamten Entwicklungszeitraum auszudehnen, indem von vornherein mehrere Ausbaustufen (***Versionen***) eingeplant

werden. Jede Version wird wiederum sorgfältig ausgewertet und Änderungswünsche sowie Verbesserungsvorschläge werden eingearbeitet. Für die nächste Ausbaustufe sind dann noch die geplanten Erweiterungen (Inkremente) einzubeziehen.

Diese Entwicklungsstrategie setzt voraus, daß man die **Ziele** vor Projektbeginn festgelegt hat, d.h. welche Aufgaben insgesamt unterstützt werden sollen und welche Probleme zu lösen sind. Außerdem müssen alle Projektbeteiligten zu einer kooperativen Systementwicklung bereit sein und sich über die Vorgehensweise und die sich daraus ergebenden Verpflichtungen verständigen.

Prototyping und inkrementelle Systementwicklung sind sehr änderungsaufwendig. Damit die Kosten hierfür möglichst gering bleiben und Lösungsvorschläge mit nur geringem Aufwand entwickelt werden können, sind neuartige Programmiersprachen entwickelt worden.

Sprachen der 4. Generation

Sie werden als «Sprachen der 4. Generation» bezeichnet und kommen bei der Verwendung von ***Datenbanken*** zum Einsatz. Sie sind stark problembezogen und daher verhältnismäßig einfach zu handhaben. Es brauchen nur die Daten und ihre Beziehungen untereinander beschrieben zu werden, nicht aber wie und in welcher Reihenfolge sie vom Rechner verarbeitet werden sollen, und ***Bildschirmmasken*** können quasi per Knopfdruck erzeugt werden.

Benutzerprogrammierung

Damit eigenen sich diese Sprachen nicht nur zur schnellen Erstellung und einfachen Änderung von Systemversionen, sondern auch zur Programmierung durch die Benutzer. Allerdings sind damit zwei wesentliche Einschränkungen verbunden:

- Problembezogene Programmiersprachen gehen immer auf Kosten der Flexibilität, denn nicht alle Änderungen sind leicht durchzuführen. Außerdem lassen sich bestimmte Probleme nur mit einer herkömmlichen Programmiersprache lösen.

- Auch wenn mit einfachen Mitteln wirkungsvolle Effekte erzielt werden können, ist trotzdem eine sorgfältige Ausbildung erforderlich (vgl. 1 Software.***Qualifizierung***), weil bei vielen Problemen grundlegende Kenntnisse über die interne Organisation der Daten erforderlich sind.

Die Entwicklung größerer Anwendungen sollte daher gut ausgebildeten Fachleuten übertragen werden. Der große Vorteil besteht jedoch darin, daß die Entwickler sich auf die wesentlichen Punkte konzentrieren können und nicht alle Details selbst erfassen und bearbeiten müssen, weil die Benutzer die Möglichkeit haben, sich die Arbeitsumgebung ihren individuellen Anforderungen gemäß anzupassen.