# Konzepte der Programmierung

## Das Ziel der Lehrveranstaltung

Die Vorlesung heißt „Konzepte der Programmierung“. In der heutigen Stunde möchte ich Sie einstimmen in das Thema. Gleich zu Beginn muss betont werden, dass es sich hier nicht um einen Programmierkurs handelt. Das Kennzeichen eines Programmierkurses besteht darin, dass eine ganz bestimmte Programmiersprache im Zentrum des Kurses steht und dass gezeigt wird, wie man mit den einzelnen Sprachelementen umgeht, d.h. was man mit den einzelnen Sprachelementen ausdrücken kann. Bei uns dagegen steht keine bestimmte Programmiersprache im Zentrum der Betrachtung, sondern wir werden im Laufe der Vorlesung unterschiedliche Programmiersprachen verwenden für unsere Übungen.

Die ersten Programmiersprachen wurden in den 50-iger Jahren entwickelt, und bis heute sind immer wieder neue Programmiersprachen hinzugekommen. Deshalb werden Sie möglicherweise vermuten, dass ich in meinem verhältnismäßig fortgeschrittenem Alter gar nicht mehr der geeignete Dozent für eine solche Vorlesung sein könne, da ich doch vermutlich nicht mit den modernsten Programmiersprachen vertraut sei. Was die fehlende Programmierroutine angeht, über die ich sicher schon seit längerem nicht mehr verfüge, haben Sie natürlich Recht. Aber genau das ist für diese Vorlesung völlig unerheblich, denn auch Sie werden immer wieder vor neue Programmiersprachen gestellt werden, die Sie dann noch nicht routiniert verwenden können und als Softwareingenieure möglicherweise auch gar nie routiniert benutzen müssen. Das Lernen einer Programmiersprache kann weitgehend autodidaktisch, d.h. ohne die Hilfe eines Lehrers, allein unter Verwendung geeigneter Lehrmittel erlernt werden, falls man zuvor die grundsätzlichen Konzepte beigebracht bekam, die in den einzelnen Programmiersprachen ihre Ausprägung finden. Wenn Sie in ihrem späteren Berufsleben, was sicher noch öfter passieren wird, vor eine neue Programmiersprache gestellt werden, brauchen Sie im wesentlichen nur zwei Dinge zu fragen:

1. Welche neuen Konzepte sind in dieser Programmiersprache realisiert worden, die es in den bisherigen Programmiersprachen noch gar nicht gab. Das werden gar nicht viele neue Konzepte sein, d.h. im allgemeinen kommen meist nur ein oder zwei neue Konzepte hinzu, alle anderen Konzepte waren in den früheren Programmiersprachen auch schon da.

2. Man muss fragen, in welcher sprachlichen Form die alten und die neuen Konzepte in der neuen Programmiersprache ausgedrückt werden.

Im Grunde könnte man, wenn die Konzepte gegeben sind, festlegen, dass diese Konzepte grundsätzlich in Zukunft immer in der gleichen sprachlichen Form ausgedrückt werden sollen. Aber in gleicher Weise, wie sich die einzelnen Sprachvölker nicht darauf einigen, in Zukunft nur noch eine einzige Sprache, beispielsweise Englisch, zu sprechen, so werden auch in Zukunft sehr viele unterschiedliche Programmiersprachen nebeneinander stehen, obwohl sie doch nur unterschiedlichen Formen zum Ausdrücken der gleichen Inhalte sind.

Wenn man die Konzepte in den Programmiersprachen schon kennt, wird man beim Lernen einer neuen Programmiersprache das Sprachbuch nicht als Lehrbuch benutzen und von vorne bis hinten durchstudieren. Man wird es vielmehr als Nachschlagewerk benutzen, worin man jeweils nach ganz bestimmten Dingen sucht. Das wonach man sucht sind zum einen die Art

und Weise, wie bestimmte Konzepte, die man schon kennt, auszudrücken sind, und das andere wonach man sucht werden Konzepte sein, die man in den frühren Programmiersprachen noch nicht vorgefunden hat.

An dieser Stelle möchte ich auf den Unterschied zwischen Erkenntniswissen und Bedienwissen hinweisen. Erkenntniswissen bleibt für uns von Wert bis an unser Lebensende, wogegen Bedienwissen nur so lange von Wert ist, wie wir uns in einem bestimmten Kontext befinden. Die Konzepte der Programmierung gehören zum Erkenntniswissen, während die sprachliche Ausprägung der Konzepte in den unterschiedlichen Programmiersprachen zum Bedienwissen gehören. Der Begriff Bedienwissen verweist auf Geräte, die man bedienen muss, bei denen man bestimmte Knöpfe drücken muss oder bestimmte Hebel bewegt. Und da man diese Geräte normalerweise nicht sein ganzes Leben lang behält, muss man immer wieder neu lernen, wo die Hebel und Knöpfe beim neuen Gerät sind. Dies ist der jeweilige Kontext des Bedienwissens. Es gehört auch zum Bedienwissen, dass man weiß, wo in der aktuellen Arbeitsumgebung die Toiletten sind.

Ich habe durchaus einmal alle sprachlichen Details unterschiedlichster Programmiersprachen routinemäßig beherrscht, angefangen von FORTRAN über ALGOL, IBM-Assembler, ER56-Assembler, LISP, SAP-ABAP, C++ und andere. Das meiste davon konnte ich in der Zwischenzeit wieder vergessen, weil ich in meinem aktuellen Kontext hierfür keinen Bedarf mehr habe.

Es ist mir auch ein Anliegen in diesem Kurs, dass Sie in den Übungsbeispielen erkennen, dass ein- und dasselbe Konzept in unterschiedlichen Programmiersprachen unterschiedlich ausgedrückt wird, obwohl es hierfür eigentlich keinen zwingenden Grund gibt.

Durch die Art und Weise, wie ich Sie an die Software heranführen will, erleben Sie einen ganz anderen Zugang, als er zur Zeit in der Informatik weltweit üblich ist. Ich kann hierzu eine Analogie darstellen. Denken Sie sich eine Gemäldeausstellung, die Sie besuchen wollen. Nun wird zu dieser Gemäldeausstellung eine Vorbesprechung angeboten, an der Sie teilnehmen, bevor Sie in die Ausstellung gehen. In dieser Vorbesprechung weist man Sie auf besondere Dinge hin, auf die Sie achten sollen. Sie sollen beispielsweise darauf schauen, in welcher Weise ein bestimmter Maler mit der Farbe Gelb umgeht. Sie werden darauf hingewiesen, dass bei einem anderen Maler auf jedem Bild in ganz besonderer Weise eine Diagonalstruktur realisiert ist. Sie werden darauf hingewiesen, wie sich über die Zeit hinweg die Art zu malen bei dem einen Künstler in einer ganz bestimmten Richtung verändert hat usw.

Wenn Sie nun in diese Ausstellung gehen, werden Sie neben Leuten stehen, die an der Vorbesprechung nicht teilgenommen haben. Ein Betrachter, der an der Vorbesprechung nicht teilgenommen hat, wird natürlich das gleiche Bild sehen wie Sie, aber er wird nicht das gleiche darüber denken. Vor seinem geistigen Auge sieht er nicht das gleiche wie Sie. Und wenn Sie in angemessener Weise vorbereitet wurden, sehen Sie vor Ihrem geistigen Auge viel klarere Strukturen als der andere, der durch die Oberfläche nicht hindurchsehen kann.

**Die zwei Seiten des Softwarebegriffs**

Nun also müssen wir uns mit dem Begriff Software befassen. Es geht um das, was wir denken sollten, wenn wir das Wort Software benutzen. Ich verhalte mich hier wie ein Philosoph, denn es ist das Wesen des Philosophen zu fragen, was man mit der Verwendung bestimmter Wörter

meint. Es ist gar nicht selbstverständlich, dass sich jeder mit den von ihm benutzten Wörtern auf solche philosophische Weise befasst. Deswegen kommen die meisten auch in Schwierigkeiten, wenn man sie bittet, mit klaren Worten darzustellen, was sie sich zu den von ihnen verwendeten Wörtern denken. Die meisten Softwarefachleute verbinden mit dem Wort Software die Vorstellung von Programmtext. Wenn sie über Software kommunizieren, reden sie über Eigenschaften von Programmtexten. Ich möchte aber erreichen, dass Sie das Wort Software nicht so eingeschränkt benutzen. Ich möchte, dass Sie den Programmtext nur als einen Aspekt der Software sehen und den Programmtext nicht mit der Software gleichsetzen. Wenn der Programmtext nur ein Aspekt der Software ist, dann muss Software noch andere Aspekte haben.

Um es plausibel zu machen, dass Software eine zweiseitige Medaille ist, von der der Programmtext nur die eine Seite ist, betrachten wir eine Analogie aus dem Bereich der Elektronik. Auf der linken Seite des Bildes 1 ist das Schaltbild eines Transistorverstärkers dargestellt, wie es jeder Elektroniker auf der ganzen Welt versteht. Rechts neben diesem Schaltbild ist eine gedruckte Platine gezeigt, auf der die Bauelemente des Verstärkers sitzen und mit Kupferbahnen verbunden sind. Der eigentliche Verstärker, wie ihn der Kunde erhält und bezahlen muss, umfasst nur die gedruckte Platine. Trotzdem versteht der Elektronikingenieur unter dem Begriff Verstärker nicht nur die gedruckte Platine, sondern auch das Schaltbild auf der linken Seite. Wenn ein Verstärker entworfen wird, entsteht das Schaltbild zuerst, und das Schaltbild ist Gegenstand der Entwurfsoptimierung.

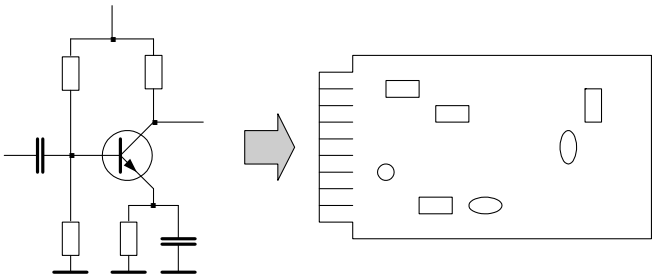

**Bild 1** Schaltbild und gedruckte Platine

Häufig ist der erste Entwurf eines Schaltbildes nicht bereits so ausgereift, dass man seine Realisierung in Form einer gedruckten Platine rechtfertigen könnte. Das Schaltbild wird zum Gegenstand der Optimierung. Häufig werden hierzu Kollegen herangezogen, mit denen man die Frage diskutiert, an welchen Stellen man den Entwurf noch verbessern könnte. Erst danach bildet man das Schaltbild auf die gedruckte Platine ab. Die Diskussionen über das Schaltbild finden unter Ingenieuren statt. Die Abbildung vom Schaltbild auf die gedruckte Platine ist dagegen im allgemeinen keine Aufgabe für Ingenieure, sondern für Techniker, die eine andere Ausbildung haben.

Nun behaupte ich, dass es sich beim Entwerfen und Realisieren von Software um einen analogen Vorgang handelt, wie ich ihn im Bereich der Elektronik soeben beschrieben habe. Das heißt wir müssen zwei Seiten der Medaille unterscheiden: Zum einen die „Schaltpläne der Software“ und zum anderen den Programmtext. Der Programmtext entspricht der gedruckten Platine, d.h. er entsteht als Ergebnis einer Abbildung, die einen Entwurf in Form technischer Pläne voraussetzt.

Das Bemühen um ein Verständnis des Verstärkers, eine Diskussion über den geschickten oder ungeschickten Entwurf des Verstärkers erfolgt nie unter Bezug auf die gedruckte Platine,

sondern immer unter Heranziehung des Schaltbildes. Es wäre unsinnig, jemandem eine gedruckte Schaltung in die Hand zu drücken und ihn aufzufordern zu erklären, um was für ein System es sich hierbei handele. In gleicher Weise sollte man im Bereich der Software verfahren. Man sollte also niemandem den Programmtext vorlegen und von ihm verlangen, er solle erklären, um was für ein System es sich hierbei handele. Wenn die Entwurfszeichnungen nicht vorgelegt werden, ist eine angemessene Diskussion über die Software nicht möglich. Ich betone also ganz besonders, dass es nicht die programmiersprachlichen Konstrukte und nicht die im Programm lesbaren Zeilen sind, über die der Softwareingenieur nachdenken und kommunizieren soll. Es muss etwas anderes sein. Es stellt sich nun zwangsläufig die Frage, was dieses andere denn sein könnte. Und damit sind wir bei der besonderen Schwierigkeit, durch welche sich die Softwaresystemtechnik sich von den traditionellen Ingenieurdisziplinen Bauwesen, Maschinenbau und Elektrotechnik grundlegend unterscheidet.

---

**Visualisierung abstrakter Strukturen**

Wenn man sagt, man wolle etwas darstellen, dem die natürliche Sichtbarkeit fehlt, dann sollte man zwei Fälle auseinander halten: Zum einen gibt es konkrete physikalische Individuen, denen die natürliche Sichtbarkeit fehlt, und zum anderen sind es die Abstrakta, zu deren Wesen es gehört, dass sie nicht sichtbar sind. Als Beispiel für ein konkretes physikalisches Individuum, welches keine natürliche Sichtbarkeit hat, sei das Atom genannt. Atome sind zweifellos physikalisch konkret, aber sie haben kein Aussehen. Wenn man sagt, man habe ein Atom sichtbar gemacht, dann heißt das, dass man ein Gerät benutzt hat, welches ein Bild erzeugt, das man eindeutig dem Atom zuordnen kann. Wenn man das Atom wegnimmt, verschwindet auch das Bild. Wenn man solche Bilder von Atomen betrachtet, sollte man nicht sagen, nun wisse man, wie ein Atom aussieht. Dies wäre ähnlich unsinnig, wie wenn man beim Betrachten der Anzeige eines Voltmeters sagen würde, nun wisse man, wie elektrische Spannung aussieht.

Im Gegensatz zum Atom, welchem man immerhin mittels eines Gerätes ein Bild zuordnen kann, ist es nicht möglich, einem abstrakten Individuum mit technischen Mitteln ein Bild zuzuordnen. Das dem Atom zugeordnete Bild lässt sich über eindeutige physikalische Kausalität dem Atom zuordnen, wogegen es keine physikalische Kausalität gibt, die von einem abstrakten Individuum ausgehen könnte. Man denke beispielsweise an den Begriff Primzahl. Dieser Begriff kann nicht mit physikalischen Mitteln in ein Bild überführt werden, sondern er kann nur symbolisiert werden. Symbolisierung bedeutet, dass einem Individuum willkürlich ein wahrnehmbares Muster zugeordnet wird. Dabei ist es zulässig, einem gedachten Individuum mehrere alternative Muster zuzuordnen. So ist beispielsweise die Buchstabenfolge *Primzahl* eine mögliche Symbolisierung für den Begriff Primzahl.

Der Gegenstandsbereich der Softwaresystemtechnik ist vorwiegend abstrakt, wogegen die Gegenstandsbereiche der traditionellen Ingenieurdisziplinen vorwiegend konkret sind. Im Studium des Maschinenbaus, des Bauwesens oder der Elektrotechnik ist es nicht erforderlich, dass ein Hochschullehrer die Studenten im ersten Semester in den Gegenstandsbereich ihres Faches einführt, denn diesen Gegenstandsbereich kennen sie ja schon seit ihrer Kinderschulzeit. Jeder erlebt sehr früh, was ein Auto ist, was eine Lokomotive ist, was ein Fahrstuhl ist, was eine Kaffeemaschine ist, was ein Fernsehgerät oder ein Telefon ist.

Diese technischen Produkte haben eine natürliche Sichtbarkeit und werden deshalb sehr früh erlebt. Die Softwarewelt ist weitgehend unsichtbar und nur im ganz Kleinen sichtbar, nämlich

in Form der Programmtextstücke. Wenn wir die Verhältnisse aus der Softwarewelt in die konkrete Alltagswelt übertragen, müssen wir uns vorstellen, unsere Welt sei so eingenebelt, dass man nur wenige Zentimeter weit sehen kann. Dann sieht man lauter kleine Gegenstandsbereiche: Man sieht einen Stein; man sieht einen Kugelschreiber, eine Brille, eine Nase, einen Schuh, aber man sieht nie etwas größeres Zusammenhängendes. Auf diese Weise kann vor dem menschlichen Auge kein merkbares Gesamtbild entstehen. Das ist das zentrale Problem der Softwaresystemtechnik, dass abstrakte Gegenstände vermittelt werden müssen, die nicht durch Zeigen und Erleben vermittelt werden können. Der Hochschullehrer ist also in der Rolle eines Philosophen, denn auch Philosophen müssen immer abstrakte Gegenstände vermitteln, d.h. sie müssen ihren Studenten irgendwie klar machen, was sie meinen, wenn sie bestimmte Wörter benutzen, die nicht für etwas Sicht- oder Zeigbares stehen.

Stellen Sie sich vor, ich würde jetzt von einem abstrakten Ding reden wollen und sage Ihnen, dass dieses Ding im Deutschen „Pasamuff“ und im Englischen „Baxtreme“ genannt wird. Wenn es mir nicht gelingt, Sie dazu zu bringen, das mit diesen Wörtern Gemeinte zu denken, können Sie unter Verwendung dieser Wörter nicht angemessen kommunizieren, d.h. diese Wörter bleiben für Sie bedeutungslos. Es ist immer Voraussetzung, dass es gelingt, bei denen, die die Bedeutung der Wörter lernen sollen, zuerst einmal zu erreichen, dass sie das damit Bezeichnete denken. In der konkreten Welt ist es einfach, denn dort wird das zu Bezeichnende auf natürliche Weise erlebt. Man denke an ein kleines Kind, welches sich die Finger an der heißen Herdplatte verbrennt. Dieses Kind erlebt den Schmerz, und es kann ein solches Erleben haben, lange bevor es sprechen kann. Das Kind kann die Eigenschaft einer Herdplatte, die ihm den Schmerz verursacht hat, denken, ohne zu wissen, wie man diese Eigenschaft bezeichnet. Später erlebt es sogar, dass es unterschiedliche Bezeichnungen für diese Eigenschaft gibt. Die Deutschen sagen „heiß“, die Franzosen „chaud“ und die Engländer „hot“. Im Bild 2 wird symbolisiert, dass es zwei Bereiche gibt, die man streng auseinander halten muss: das zu Bezeichnende und die Bezeichnungen oder Notationen.

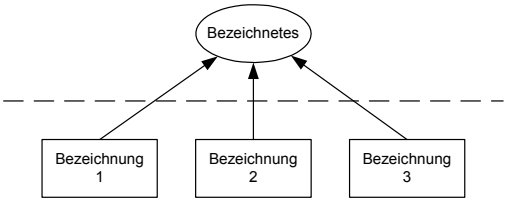


**Bild 2** Ein Bezeichnetes und mehrere alternative Bezeichnungen

Wenn man eine Fremdsprache lernt, muss man meistens das zu Bezeichnende nicht mehr lernen, weil man das schon erlebt hat, bevor man die Muttersprache als erste Notation gelernt hat. Anschließend muss man nur noch neue Notationen lernen. Ebenso verhält es sich beim Lernen einer neuen Programmiersprache, wenn man vorher schon viele andere Programmiersprachen gelernt hat. Nur diejenigen Konzepte, die es tatsächlich erst mit der neuen Programmiersprache gibt, stellen den Lernenden vor die Aufgabe, nicht nur Notationen sondern auch neue Elemente der abstrakten Welt zu lernen.

Nun ist die ganze Zeit immer betont worden, dass über diese abstrakte Welt kommuniziert werden müsse. Man könnte ja auch die Vorstellung haben, dass es im technischen Bereich genügt, wenn diese abstrakte Welt gedacht werden kann. Derjenige, der sie denkt, ist dann auch derjenige, der durch sein Denken zu dem Programmtext kommt. Dies würde aber voraussetzen, dass die durch Programmierung zu realisierenden Probleme so klein sind, dass man dabei keine Arbeitsteilung braucht. Das gilt aber immer nur für die allerersten Anfänge

der Entwicklung technischer Disziplinen. Später werden die in diesen Disziplinen geschaffenen Systeme immer größer, und der Grad der erforderlichen Arbeitsteilung wird immer höher.
Ein Großteil seiner Zeit ist der Ingenieur mit Informationsbeschaffung und mit Wissensablieferung befasst. Er muss sich Informationen über Systemkomponenten beschaffen, er muss sich Informationen über die Entscheidungen des Planers beschaffen, der ihm seine Aufgaben zuweist, und er muss seine eigenen Entwurfsentscheidungen für andere verständlich weiter geben. Man wird ein besonders guter Ingenieur, wenn man besondere Freude daraus zieht, anderen technisches Wissen weiter zu geben.

**********

Es ist selbstverständlich, dass kein Mensch über alle Details eines großen Systems Bescheid wissen kann. Er sollte aber immer so viel über das System wissen, dass für ihn die Funktionalität des Systems plausibel ist. Er sollte nie das System als geheimnisvolles Gebilde ansehen müssen, wo er sich gar nicht vorstellen kann, dass normale Menschen in der Lage sein können, so ein System zu bauen und zu verstehen.

Wir müssen also immer so viel über die Systeme wissen, dass sie für uns entzaubert sind. Denn dann kann man einigermaßen abschätzen, ob Aussagen, die über das System gemacht werden, überhaupt zutreffen können oder völlig Unmögliches behaupten.

Die Physikkenntnisse des allgemein gebildeten Menschen reichen aus, dass für ihn ein Flugzeug oder ein Fernsehgerät plausibel sind. Während die Ingenieure des Maschinenbaus und der Elektrotechnik im allgemeinen bemüht sind, dem allgemein gebildeten Nichtingenieur die Ingenieursprodukte plausibel zu machen, herrscht unter Softwareingenieuren eher die Einstellung vor, dass die von ihnen geschaffenen Systeme so komplex seien, dass man sie den Nichtfachleuten nicht plausibel machen könne. Als zukünftige Ingenieure der Softwaresystemtechnik sollen Sie es nicht nur gelernt haben, sondern Sie sollen auch immer den dringenden Wunsch haben, die komplexen Softwaresysteme Ihren Mitmenschen, auch wenn sie keine Fachleute sind, plausibel zu machen. Sie sollen nicht stolz darauf sein, dass Ihre Mitmenschen Sie als Zauberer bewundern, sondern Sie sollen Ihren Stolz darauf gründen, dass es Ihnen gelungen ist, Ihren Mitmenschen Ihre Ingenieursprodukte als menschliche Konstruktionen verstehbar zu machen.

Es wird in letzter Zeit immer sehr viel von der Notwendigkeit gesprochen, bei der Ingenieurausbildung auch darauf zu achten, dass die Absolventen teamfähig sind. Ein Team ist eine Gruppe von Menschen mit einer gemeinsamen Aufgabe, wobei aber dieses Team in einem engen Kommunikationsverbund lebt. Im Allgemeinen handelt es sich um 5 bis maximal 10 Leute, die gemeinsam arbeiten, den ganzen Tag miteinander reden können und von denen jeder genau die Fähigkeiten des anderen einschätzen kann. Wenn ich aber von Arbeitsteilung rede, dann habe ich nicht die Vorstellung von 5 bis 10 Leuten sondern durchaus auch die Vorstellung von Tausend oder noch mehr Leuten. SAP hat etliche Tausend Entwickler, die alle in irgendeiner Weise einen Beitrag zu dem großen SAP-System liefern. Die in solch großen Unternehmen arbeitsteilig zusammenwirkenden Fachleute können nur durch ein gemeinsames Wissen zusammengehalten werden. Deswegen müssen sie gelernt haben, ihr Wissen abgeben zu können, und dazu müssen sie es so formulieren, dass diese Formulierungen verstanden werden können, ohne dass der Autor daneben steht und die Fragen des Lesenden beantwortet. Zur Zeit ist im Softwarebereich dieser Zustand längst noch nicht erreicht, d.h. die meisten Zeichnungen, die man von Softwareingenieuren gezeigt bekommt, kann man nur verstehen, wenn der Autor daneben steht und erklärt, was er mit

jedem einzelnen Strich gemeint hat. Wenn ich Kommunikation sage, meine ich also meistens monologische Kommunikation, d.h. Darstellung für andere - so wie ein Romanautor nicht im Dialog mit seinen Lesern steht, sondern in einer Monologbeziehung.

**Trennung der Themenbereiche für die Kommunikation**

Wenn es um das Problem der Kommunikation und der Darstellung von Softwaresystemen geht, muss man geeignete Abstraktionen schaffen, die man dann kommunikationsfreundlich symbolisieren muss. Wenn man dem Wesen der Software gerecht werden will, muss man zuerst einmal fragen, welchem Zweck Software im Allgemeinen dient. Und da kann man generell feststellen:

Software dient in jedem Falle der Simulation eines Dienstleistungsunternehmens für Informationsverarbeitung.

Das kleinste denkbare Dienstleistungsunternehmen besteht aus einem einzigen Mitarbeiter, der nur eine einzige Funktion f(x), beispielsweise die Sinusfunktion sin(x), kennt, die er auf angelieferte Argumente anwenden kann. Er erbringt also die Dienstleistung, zu einer angelieferten Zahl x das Wert f(x) zu berechnen und mitzuteilen.

Bei der Kommunikation über Softwaresysteme müssen wir also zwei Themenbereiche streng auseinander halten. Zum einen kommunizieren wir über das Dienstleistungsunternehmen, und zum anderen kommunizieren wir über den Simulator.

Wenn wir über ein Dienstleistungsunternehmen für Informationsverarbeitung kommunizieren, brauchen wir überhaupt keinen Bezug zu nehmen auf Computer oder Software. Denn Dienstleistungsunternehmen für Informationsverarbeitung gab es schon über tausend Jahre vor der Erfindung des Computers. Ein solches Dienstleistungsunternehmen besteht nämlich immer aus Unternehmensmitarbeitern, die nichts anderes tun, als mit Kunden zu kommunizieren, untereinander zu kommunizieren, Eingabeinformationen in Ausgabeinformationen zu transformieren und Informationen auf Speichern festzuhalten. So wie es in Unternehmen mit menschlichen Mitarbeitern die Vorgänge der Einstellung und der Kündigung gibt, wodurch Mitarbeiter entstehen beziehungsweise verschwinden, muss es dies auch in den simulierten Unternehmen geben.

Wenn wir über den Simulator für ein solches Dienstleitungsunternehmen kommunizieren wollen, müssen wir fragen, wie das Verhalten der einzelnen Mitarbeiter im Unternehmen simuliert wird, wie das Eintreten und das Ausscheiden von Mitarbeitern simuliert wird und wie die Speicher realisiert werden, auf denen die Speichervorgänge und Lesevorgänge stattfinden.

Der Simulator kann selbst wieder als Dienstleistungsunternehmen für Informations-verarbeitung betrachtet werden und ist deshalb selbst wieder simulierbar. Man kann sagen, dass die Computer-Hardware ein nichtsimulierter Simulator ist, der die Simulation unter Nutzung physikalischer Effekte durchführt.

Ein Programmtext kann stets als Beschreibung eines Dienstleistungsunternehmens für Informationsverarbeitung betrachtet werden, die für einen bestimmten Simulator verständlich ist. Wenn dieser Simulator selbst wieder simuliert wird, muss es einen weiteren Programmtext geben, der den Simulator als Dienstleistungsunternehmen beschreibt und den ein tiefer

liegender Simulator versteht. Auf diese Weise gibt es in großen Softwaresystemen eine Schichtung von Programmtexten, so dass man das System nur verstehen kann, wenn man die verschiedenen Programmtextabschnitte den richtigen Ebenen zuordnet.

Jedermann wird sofort einsehen, dass es sinnvoll ist, die gewünschten Dienstleistungsunternehmen für Informationsverarbeitung zu simulieren und nicht als Unternehmen mit menschlichen Mitarbeitern zu realisieren. Denn durch die Simulation können die gewünschten Leistungen viel schneller, mit größerer Zuverlässigkeit und in viel größerem Umfang erbracht werden als durch eine menschliche Gruppe. Dennoch ist der Unterschied zur menschlichen Gruppe für das Verständnis gering, es geht nur um „schneller, zuverlässiger und umfangreicher“, aber nicht um grundsätzlich etwas anderes.

Während wir in der Vorlesung „Konzepte der Programmierung“ die Computer-Hardware als Simulatorrealisierung hinnehmen, ohne nach den physikalischen Prinzipien und den technischen Konzepten zu fragen, die der Konstruktion zugrunde liegen, wird Ihnen die Computer-Hardware in der Vorlesung „Einführung in die Digitaltechnik“ plausibel gemacht, so dass auch das untere Ende der Systembetrachtung für Sie keine Zauberei mehr darstellt.

Programmierung ist computerverständliche Formulierung des Gewollten. Damit der Computer das gewollte System simulieren kann, muss man eine bestimmte Beschreibung dieses Systems als Mitteilung an den Computer geben. Bevor man aber den Computer zur Simulation veranlasst, wird man den Entwurf des Gewollten mit anderen Menschen validieren wollen. Validierung bedeutet, dass man überprüft, ob es wirklich so ist, wie man es dargestellt hat. Im gegebenen Falle heißt das, dass man überprüft, ob es wirklich zweckmäßig ist, das zu wollen, was man formuliert hat. Man denke an das Gespräch zwischen einem Bauherrn und seinem Architekten. Der Architekt formuliert das, was er als den Willen des Bauherrn verstanden hat. Dann zeigt er dem Bauherrn seine Formulierung, d.h. seine Gebäudezeichnungen, und es findet eine Validierung statt, indem der Bauherr entweder bestätigt, dass sein Wille genau getroffen wurde, oder aber indem der Bauherr sagt: Nein, so habe ich das nicht gewollt. Neben der Validierung gibt es noch die Verifikation. Dieser Begriff bezeichnet die Überprüfung, ob das Realisierte mit dem formulierten Willen verträglich ist. Am Beispiel des Architekten bedeutet dies die Überprüfung, ob die Handwerker das Haus so gebaut haben, wie es der Architekt durch seine Zeichnungen verlangt hat.