

Mittel zur effizienten Kommunikation über große Softwaresysteme

Dr.-Ing. Peter Tabeling

HASSO-PLATTNER-INSTITUT
für Softwaresystemtechnik GmbH
Am Luftschiffhafen 1, 14471 Potsdam
Tel.: 0331 90972-0
Fax: 0331 90972-10
e-Mail: tabeling@hpi.uni-potsdam.de

Einleitung

Zweck dieses Beitrages ist es, die Notwendigkeit ingenieurmäßiger Darstellungen im Bereich großer Softwaresysteme herauszustellen. Diese Notwendigkeit läßt sich aufzeigen, indem man erstens der Frage nachgeht, welche Probleme bei der Kommunikation über *große* Softwaresysteme auftreten. Desweiteren ist die Frage zu klären, was unter einer *ingenieurmäßigen* Darstellung zu verstehen ist. Dies betrifft letztlich die Frage nach Zweck, Inhalt und Form von Beschreibungsmitteln, die dem Informationsaustausch zwischen den an der Systemerstellung beteiligten Personen dienen.

Große Softwaresysteme

Die Unterscheidung von kleinen und großen Softwaresystemen wirft die Frage auf, nach welchem Kriterium ein System als groß einzustufen ist. Als wesentliches Merkmal wird hier der Aufwand bei der Systementwicklung betrachtet. Große Softwaresysteme können nicht von einer oder wenigen Personen entwickelt werden, sondern erfordern den arbeitsteiligen Einsatz hunderter oder gar tausender Personen. Daraus ergibt sich ein entsprechender Umfang der entstehenden Software. Ein kleines System umfaßt typischerweise Tausende von Programmzeilen, während ein großes System durchaus 20 Millionen Zeilen Quelltext umfassen kann.

Verschiedene Arten von Kommunikation

Im Zusammenhang mit der Softwareentwicklung sind verschiedene Arten von Kommunikation gegeben, die sich anhand folgender Fragen unterscheiden lassen:

- Was ist der Gegenstand der Kommunikation?
- Wer sind die Kommunikationspartner?

Gegenstand der Kommunikation wird meistens das zu entwickelnde System sein - daneben kommen Sachverhalte in Frage, die zwar mit der Systementwicklung in Zusammenhang stehen, aber nicht das System selbst betreffen. Dies sind z.B. Informationen, die zur Organisation und Steuerung der Softwareentwicklung ausgetauscht werden. Die folgenden Betrachtungen beschränken sich auf Kommunikation, die das zu entwickelnde System zum Gegenstand hat.

Bezüglich der Frage nach den Kommunikationspartnern ist es besonders wichtig, zwischen der Kommunikation unter Menschen und der Mensch-Maschine-Kommunikation zu unterscheiden.

Mensch-Maschine-Kommunikation

Das naheliegendste Beispiel für eine Mensch-Maschine-Kommunikation ist das Programmieren, bei dem der Entwickler dem Entwicklungssystem das Programm als Beschreibung des gewünschten Systems eingibt. In die gleiche Kategorie fällt auch die Erstellung von Grafiken, die als Ausgangspunkt für die Generierung von Programmcode verwendet werden (können) sowie die Eingabe formaler Spezifikationen, die eine automatische Verifikation bestimmter Programmeigenschaften erlauben.

Desweiteren können moderne Entwicklungsumgebungen Beschreibungen generieren, welche zwar auf Basis des Programmcodes erzeugt werden, aber dem Menschen einen leichteren Zugang zu bestimmten Programmstrukturen ermöglichen, wie z.B. eine automatisch erstellte grafische Darstellung des Klassenbaumes zu einem objektorientiert programmierten System.

Diese Beispiele zeigen, daß bei der Mensch-Maschine-Kommunikation Beschreibungsmittel verwendet werden, die zwar zusätzlich zum Programmcode gegeben sind, aber mit diesem in einer *formalen, vorab definierten Beziehung* stehen. Sie ist formal, weil sonst eine automatische Verifikation bzw. Generierung unmöglich wäre, und

vorab definiert, da sie bereits bei der Erstellung des Entwicklungssystems (welches die Verifikation bzw. Generierung nach bestimmten Algorithmen durchführt) festgelegt werden mußte. Derartige Beschreibungsmittel werden im folgenden als „codenah“ bezeichnet.

Mensch-Mensch-Kommunikation

Neben der Mensch-Maschine-Kommunikation erfordert die Softwareentwicklung eine Kommunikation unter Menschen. Bei kleinen Systemen kann diese Kommunikation noch ad hoc und in wenig organisierter Weise erfolgen, da die Entwicklung in enger Zusammenarbeit im Team stattfindet. Entwickler können sich z.B. bei Bedarf treffen und im direkten Gespräch alternative Entwürfe diskutieren. Derartige Diskussionen können gut auf Basis codenaher Beschreibungen - oder gar des Programmcodes selbst - erfolgen. Die Beschreibungsmittel der Mensch-Maschine-Kommunikation sind also bei kleinen Systemen auch für die Mensch-Mensch-Kommunikation geeignet.

Dagegen stellt die Mensch-Mensch-Kommunikation bei großen Systemen ein Problem dar, welches nur durch geeignete Organisation und mittels spezieller Beschreibungsmittel in befriedigender Weise gelöst werden kann. Arbeiten mehrere tausend Entwickler an einem System, so erfordert dies weitestgehende Arbeitsteilung. Enge Zusammenarbeit und ad-hoc-Kommunikation kann dann nur noch innerhalb einzelner Arbeitsgruppen stattfinden.

Von entscheidender Bedeutung für eine funktionierende Arbeitsteilung ist der Überblick, den der Einzelne über das Gesamtsystem hat - auch wenn die eigene Arbeit nur bestimmte Systembereiche betrifft. Dieser Überblick kann jedoch nicht mehr auf Basis codenaher Beschreibungen (geschweige denn des Programmcodes) entstehen. Die Komplexität und der Umfang dieser Beschreibungsmittel wächst mit der Komplexität und dem Umfang des Programmcodes. So mögen z.B. generierte Grafiken kompakter als der entsprechende Quelltext sein. Ab einer bestimmten Systemgröße genügt diese Kompaktheit jedoch nicht mehr zur Erstellung einer handhabbaren Systembeschreibung. Eine Unmenge generierter Grafiken ist für den Menschen ebenso nutzlos wie eine Flut von Programmzeilen.

Eine effiziente Mensch-Mensch-Kommunikation erfordert Systembeschreibungen, deren Umfang, Form und Inhalt speziell auf die Bedürfnisse des Menschen abgestimmt sind. Immer umfangreichere Softwaresysteme erfordern immer höhere Abstraktionen, ohne die eine kompakte Darstellung der primär interessierenden Sachverhalte nicht möglich ist.

Die dazu erforderliche Abstraktionsleistung kann nur von Menschen erbracht werden. Die Vorstellung, daß übergeordnete Modelle durch ein Werkzeug generierbar wären, kommt der Behauptung gleich, man habe einen „universellen Algorithmus des Abstrahierens“ gefunden - niemand wird dies ernsthaft behaupten wollen. Welche die wichtigen Strukturen und die wesentlichen Entwurfsmerkmale eines Systems sind, kann nur von einem Menschen entschieden werden, da „wichtig“ und „wesentlich“ Eigenschaften sind, die sich einer formalen Definition entziehen.

Praktische Erfahrungen bei der Dokumentation großer Softwaresysteme haben gezeigt, daß bedeutende Entwurfsschritte deutlich über einfache Implementierungsschritte wie Dekomposition von Komponenten oder prozedurale Umschreibung von Operationstypen hinausgehen. Die entsprechenden Beziehungen zwischen übergeordneten Modellen und den letztlich verfassten Quelltexten sind i.d.R. gar nicht den programmiersprachlichen Konstrukten zu entnehmen, sondern nur noch den Köpfen der Entwickler. Dies bedeutet, das insbesondere diejenigen Modelle, die für das Verständnis grundlegender Systemstrukturen unverzichtbar sind, nicht auf Basis von Programmcode oder codenahen Beschreibungsmitteln erstellt werden können. Schon deshalb ist einer werkzeuggestützten „Modellierung“ von vorneherein der Boden entzogen.

Beschreibungsmittel für eine effiziente Mensch-Mensch-Kommunikation sind nach den Bedürfnissen des Menschen zu gestalten. In praktischen Projekten haben sich vor allem halbformale grafische Darstellungen als sehr nützlich erwiesen.

Die Mensch-Mensch-Kommunikation über große Softwaresysteme erfordert spezielle Beschreibungsmittel, die zusätzlich zu den Beschreibungsmitteln der Mensch-Maschine-Kommunikation zu erstellen sind.

Von „klassischen“ Ingenieuren lernen

Nachdem der Bedarf nach speziellen Beschreibungsmitteln für die Mensch-Mensch-Kommunikation verdeutlicht wurde, stellt sich natürlich die Frage nach der zweckmäßigen äußeren und inhaltlichen Gestaltung. Hier lassen sich möglicherweise Ansätze aus den Systembeschreibungen der klassischen Ingenieursdisziplinen wie z.B. Maschinenbau oder Elektrotechnik ableiten.

Die konstruktive Tätigkeit eines Ingenieurs besteht primär darin, Modelle zu entwickeln. Dabei ist unter einem Modell ein zunächst nur gedanklich vorhandenes, abstraktes System zu verstehen, welches erst später in Form des tatsächlich gefertigten Produktes realisiert wird. Der Ingenieur denkt sich solche Modelle jedoch nicht nur aus. Er wird sie auch in einer angemessenen Form beschreiben, um das Ergebnis seiner Denkarbeit anderen zugänglich zu machen - er erstellt also Systembeschreibungen für die Mensch-Mensch-Kommunikation. Beispiele für solche Beschreibungen sind Schaltpläne oder Baupläne für Getriebe.

Was sind nun typische Merkmale dieser ingenieurmäßigen Beschreibungen?

Zunächst weisen diese Pläne eine hohe *darstellerische Qualität* auf. Diese Qualität beruht möglicherweise darauf, daß man bei der Entwicklung der Plantypen lange Zeit nur den Menschen als Betrachter vor Augen hatte, da eine Eignung zur maschinellen Verarbeitung erst nach der Entstehung des Computers in Frage kam. Dies erklärt auch, warum *halbformale, grafische* Darstellungen im klassischen Ingenieurbereich selbstverständlich eingesetzt werden. Derartige Darstellungen setzen sich im Softwarebereich erst langsam durch, da dort allzu oft der Wunsch nach maschineller Verarbeitbarkeit im Vordergrund steht.

Die darstellerische Qualität der Ingenieurspläne äußert sich u.a. in einer *ausgereiften Symbolik*. Beispielsweise verfügen Maschinenbauer über ein bewährtes und teilweise sogar genormtes Repertoire von Zeichnungselementen. Dieses Repertoire ist so *universell*, daß sich unterschiedlichste Systemtypen wie Turbinen, Getriebe oder Nähmaschinen damit beschreiben lassen. Eine ähnlich ausgereifte Symbolik wird für den Softwarebereich dringend benötigt.

Daß ein Ingenieur die klare Darstellung seiner Entwürfe als wichtigen und selbstverständlichen Teil seiner Arbeit ansieht, liegt nicht zuletzt daran, daß er *entsprechend ausgebildet* wird. Fächer wie „Technisches Zeichnen“ sind typische Bestandteile eines Maschinenbau-Grundstudiums. Ingenieure für Softwaresysteme sollten eine vergleichbare Ausbildung erhalten.

Die Qualität ingenieurmäßiger Darstellungen beschränkt sich keineswegs auf deren Form. Sie ist auch in der *ausgereiften Terminologie* begründet, die hinter diesen Beschreibungen steht. Die Systembeschreibungen von Ingenieuren basieren weltweit auf einem festen begrifflichen Fundament - der (klassischen) Physik. Eine gleichermaßen anerkannte begriffliche Basis läßt sich im Bereich der Softwaresysteme derzeit nicht abgrenzen.

Vielversprechende Ansätze

Darstellungen aus den „klassischen“ Ingenieursdisziplinen beschreiben i.d.R. einen mehr oder weniger abstrakten Aufbau eines Systems aus einfacheren Komponenten. Man denke z.B. an die Register-Transfer-Netze des Digitaltechnikers oder die Konstruktionszeichnungen eines Maschinenbauers. Das Verhalten des Systems wird zweckmäßigerweise erst auf Basis eines solchen Aufbaus beschrieben, da erst im Systemaufbau die Orte identifiziert werden können, an denen eben dieses Verhalten beobachtbar ist. In diesem Sinne ist die Modellierung durch den Ingenieur als *aufbauorientiert* zu bezeichnen.

Daß diese aufbauorientierte Sicht auch auf Softwaresysteme übertragbar ist, kann durch eine Analogie begründet werden. Informationsverarbeitende Systeme übernehmen Tätigkeiten, die ansonsten von Menschen zu erledigen wären. Daher ist es stets möglich, sich diese Systeme durch kooperierende Menschen zu veranschaulichen [1] (wenn man einmal von Schnelligkeit, Komplexität und Zuverlässigkeit der Verarbeitung absieht). Dabei kann man sich zu jeder im System vorliegenden Information einen physikalischen Träger (z.B. einen beschrifteten Zettel) denken, der auf einem Ort abgelegt ist und zu dem bestimmte Personen Zugang haben. Aus dieser Anschauung läßt sich ein abstrakter Komponenten- und Ortsbegriff ableiten. Dies erlaubt die Beschreibung eines abstrakten Systemaufbaus, welcher losgelöst von der physikalisch-räumlichen Systemstruktur zu betrachten ist und die Grundlage für die Verhaltensbeschreibung des Systems bildet [2].

Eine derartige, aufbauorientierte Beschreibung von Softwaresystemen hat verschiedene Vorteile. Zum einen weist sie - aufgrund der dahinterstehenden Analogie - eine gewisse Anschaulichkeit auf. Dies stellt ein nicht zu unterschätzendes Qualitätsmerkmal im Hinblick auf die Mensch-Mensch-Kommunikation dar und hat sich bereits bei der Dokumentation großer industrieller Systeme wie z.B. dem System R/3 von SAP bewährt. Darüber hinaus hat sich gezeigt, daß erst die ganzheitliche Betrachtung von Systemaufbau und -verhalten zu einer in sich schlüssigen Terminologie führt.

Die aufbauorientierte Sichtweise alleine genügt jedoch nicht. Ein wesentliches Merkmal speziell großer Softwaresysteme besteht darin, daß eine Betrachtung des Systems auf einer einzigen, bestimmten Abstraktionsebene unzweckmäßig ist. Vielmehr ist eine Halbordnung aus mehr oder weniger realisierungsnahen Modellen zu betrachten [3]. Bei einem großen System ist daher stets eine *Modellhierarchie* zu beschreiben, d.h. es sind eine Vielzahl von Modellen sowie deren Beziehungen untereinander zu betrachten.

Besonders große Systeme sind gut zu modellieren, wenn man diese beiden Prinzipien - Aufbauorientierung und Unterscheidung von Modellebenen - befolgt. Typische Merkmale derartiger Systeme wie z.B. Nebenläufigkeit, Verteilung und Transaktionen lassen sich auf diese Weise optimal erfassen [2].

Pläne „klassischer“ Ingenieure sind reine *Systembeschreibungen*. Übertragen auf Softwaresysteme bedeutet dies, dass ingenieurmäßige Beschreibungen dieser Systeme *nicht* die Software zum Gegenstand haben sollten, denn Software ist eine für die Mensch-Maschine-Kommunikation erforderliche Systembeschreibung und somit klar vom eigentlichen System zu unterscheiden. Es gilt Strukturen des Beschriebenen (=System) und Strukturen der Beschreibung (=Software) zu trennen. Zur Bewältigung der Komplexität großer Softwaresysteme ist es von besonderer Bedeutung zuerst ein Verständnis des eigentlich gewollten Systems zu erzeugen. Erst danach sollte dargestellt werden, wie der zugehörige Programmcode organisiert ist. Programmverständnis setzt Systemverständnis voraus - aber nicht umgekehrt.

Ein weiteres Mittel der Komplexitätsreduktion sind Beschreibungen von *Aspektmodellen*. Zweckmäßigerweise wird man nicht nur Aufbau und Verhalten in getrennten Plänen darstellen, sondern auch bestimmte, gerade interessierende Merkmale eines Systemmodells isoliert betrachten. Beispiele hierfür sind Beschreibungen bestimmter Teilsysteme oder die Darstellung von Kommunikationsprotokollen. Anstelle eines komplexen Dokumentes treten dann mehrere mit jeweils verringerter Komplexität.

Literatur

- [1] Siegfried Wendt, *Nichtphysikalische Grundlagen der Informationstechnik*, 1. Auflage; Springer Verlag, Heidelberg 1989; S. 246
- [2] Peter Tabeling, *Der Modellhierarchieansatz zur Beschreibung nebenläufiger, verteilter und transaktionsverarbeitender Systeme*, Dissertation, Universität Kaiserslautern 2000
- [3] Andreas Bungert, *Beschreibung programmierter Systeme mittels Hierarchien intuitiv verständlicher Modelle*, Shaker Verlag, Aachen 1998 (zugl. Dissertation, Universität Kaiserslautern)