DER 'ARBEITSKONTEXT' ALS KOMPONENTE DER BENUTZERSCHNITTSTELLE

W.Dzida; C.Hoffmann; W.Valder

Zusammenfassung: Für Benutzerschnittstellen an komplexen Anwendungssystemen wird eine Komponente entwickelt, die einen ausreichenden Transfer von Wissen über die Anwendung des komplexen Systems sicherstellt. Es gilt, das Wissen von erfahrenen Benutzern so aufzubereiten and anzubieten, daß weniger erfahrene Benutzer das reiche Funktionsangebot ebenfalls nutzen können. Andernfalls bleibt die Gefahr bestehen, daß das Funktionsangebot eines Systems nicht genutzt wird und das System nicht wirtschaftlich eingesetzt werden kann. Das hier vorgestellte Konzept kann auf komplexe Konstruktionsarbeitsplätze beim CAD und in der Software-Entwicklung angewandt werden, soweit diese Arbeitsplätze auf der Basis von Unix entwickelt worden sind.

Entwurfskonzepte und ihre Auswirkungen

Abstraktion und Konkretion von Datentypen haben auf die Benutzung eines komplexen Anwendungssystems einen großen Einfluß. Dies kann man sich an einer relativ einfachen Handlung klar machen, der "Briefwahl" (vgl.Abb.1).

Stimmzettel markieren	Wahlschein signieren	Stimmzettel kuvertieren	Umschlag und Wahlschein kuvertieren
Schritt 1	Schritt 2	Schritt 3	Schritt 4

Abb.1: Drei Datentypen bei der Darstellung der Handlung "Briefwahl"

Es ist ohne weiteres möglich, die in Abb.1 dargestellten drei Datentypen (nämlich "Stimmzettel", "Wahlschein" und "Umschlag") auf zwei zu reduzieren, und zwar auf "Papier" und "Umschlag" (vgl.Abb.2).

Papier beschriften	Papier beschriften	Umschlag schließen	Umschlag schließen, beschriften
Schritt 1	Schritt 2	Schritt 3	Schritt 4

Abb.2: Die Handlung "Briefwahl" auf zwei Datentypen reduziert

Die in Abb.2 dargestellte Reduktion von 3 auf 2 Datentypen entspricht einer Abstraktion. Hierdurch bleibt der Vorgang der "Briefwahl" jedoch nicht mehr klar verständlich. Diesem Nachteil steht jedoch ein Vorteil gegenüber. Die Datentypen

sind nicht mehr spezialisiert und somit auch in anderen Anwendungssituationen verwendbar. Man verliert durch Abstraktion zwar die Nähe zum ursprünglichen spezifischen Problem, aber man gewinnt eine gewisse Anwendungsunabhängigkeit. Diese Erkenntnis wird bei den meisten software-technischen Implementierungen ausgenutzt. So sind z.B. beim UNIX-System sämtliche konkreten Anwendungen eines Benutzers auf einen Datentyp reduziert, nämlich auf "Zeichenfolgen".

Die am Beispiel "Briefwahl" demonstrierten Vor- und Nachteile von Abstraktion und Konkretion der Datentypen haben auf die Benutzung eines Software-Systems Einfluß. In einem Anwendungsfall, in dem sowohl Ausgangs- und Zielsituation als auch die Zwischenschritte dem Benutzer bekannt sind, braucht er nicht von den konkreten Datentypen zu abstrahieren, das heißt er kann so planen und ausführen wie z.B. in dem in Abb.1 vorgezeichneten Verfahren. Ein solcher Anwendungsfall wird in der Psychologie des problemlösenden Denkens als "Interpolationsproblem" (Dörner, 1978) bezeichnet. Der Benutzer eines Software-Systems kann in dieser Problemsituation am besten unterstützt werden, wenn ihm Daten und Werkzeuge konkret und anwendungsspezifisch angeboten werden. Systeme, die nach dem Entwurfsprinzip des "objektorientierten" Programmierens entwickelt werden, bieten den Benutzern diese Vorteile (siehe z.B. XEROX-Star, Apple-Macintosh).

Nicht in allen Arbeitsbereichen kann man zwischen Ausgangs- und Zielsituation in der Weise interpolieren, daß die Zwischenschritte nur noch in die richtige Ordnung gebracht zu werden brauchen. Die meisten Konstruktionsprobleme, z.B. beim CAD oder in der Software-Entwicklung, bergen das Risiko in sich, daß kein allgemein bekanntes Lösungsverfahren existiert. Das Inventar der Werkzeuge, das zur Lösung des Problems benötigt wird, wird also in der Regel unvollständig sein.

In solchen Situationen muß ein benötigtes Werkzeug erst noch entwickelt oder ein vorhandenes angepaßt werden. Dörner (1978) spricht in solchen Fällen von Problemsituationen mit "Synthese-Barriere". Der Benutzer eines Software-Systems kann in dieser Problemsituation am besten unterstützt werden, wenn ihm Daten und Werkzeuge auf einem Abstraktionsniveau zur Verfügung stehen, das eine gewisse Anwendungs-Neutralität mit sich

bringt. Somit wird er in die Lage versetzt, eingetretene Lösungspfade leichter verlassen zu können, weil durch die relativ unspezifischen Datentypen die Werkzeuge auf die jeweils spezifische Problemsituation leichter angepaßt werden können. Die Lösung besteht dann in der Entwicklung von etwas Neuem, das als Synthese aus dem vorgefundenen, unspezifischen Werkzeug und der eigentlich erforderlichen, spezifischen Funktionalität entsteht. Wir vermuten, daß die weltweite Verbreitung des UNIX-Systems unter anderem auf diesen Umständen beruht.

Stellt man die Entwicklungskonzepte des XEROX-Star (mehrere Datentypen) und des UNIX-Systems (ein Datentyp) gegenüber, so kann man im Hinblick auf die System-Benutzung folgende verallgemeinernde Aussagen treffen (vgl.Abb.3):

Entwurfskonzept	Vorteile für die Benutzung	Nachteile für die Benutzung
System-Entwurf auf der Basis mehrerer Datentypen	klar strukturierte Aufgaben; selbsterklärende Anwendungssituationen	beschränkt auf vorweg definierte Anwendungen; Integration v.Anwendungen schwierig
System-Entwurf auf der Basis eines Datentyps	relative neutral gegenüber spezifischen Anwendungsverfahren; gute Kobinierbarkeit der Werkzeuge	schlechte Übersicht über die Anwendungsmöglichkeiten; zunehmende Komplexität

Abb.3: Vor- und Nachteile für die Benutzung eines Systems je nach Entwurfs-Konzept

Wir haben versucht, die Vorteile beider Entwurfskonzepte an einer Programmier-Umgebung des UNIX-Systems zu kombinieren, indem wir die Benutzerschnittstelle nach dem Entwurfskonzept des "objektorientierten" Programmierens (mehrere Datentypen) gestaltet haben. Auf diese Weise ist eine Planungs- und Erklärungskomponente des UNIX-Systems entstanden, die die Nachteile des UNIX-Systems ausräumen hilft, nämlich schlechte Übersicht über die Anwendungsmöglichkeiten und mangelnde Beherrschung der Komplexität. Die sonst bestehenden Benutzungs-Vorteile des UNIX-Systems bleiben von dieser Benutzerschnittstelle unberührt.

Das Konzept "Arbeitskontext"

In der arbeitspsychologischen Literatur werden u.a. zwei Arbeitstile unterschieden: (1) der Arbeitende arbeitet "drauf

los", plant meist nur den nächsten Schritt, bedenkt die Spätfolgen nicht, holt nicht genug Information ein; (2) der Arbeitende trifft sorgfältige Vorbereitungen; er legt sich die Werkzeuge zurecht, überlegt in welcher Reihenfolge vorzugehen ist, deckt Schwachstellen auf, holt Information ein bevor er mit der Ausführung beginnt (vgl. Hacker, 1978).

Wer ein komplexes Funktionsangebot einsetzen will, wird vermutlich keine Routineaufgaben erledigen, sondern Problemlösungen anstreben. Für solche Arbeitssituationen ist der zuerst geschilderte Arbeitsstil sicher ungeeignet. Eine planerische Vorgehensweise wird zweckmäßig sein. Dazu gehört z.B. auch das umsichtige Einholen der Erklärungen anderer Kollegen.

Diese Form der Arbeitsvorbereitung ist meist mühsam, weil z.B. kein kooperatives Arbeitsklima herrscht oder weil die Erfahrungen anderer gar nicht in brauchbarer Form aufbereitet sind. Folglich dauert der Einarbeitungsprozeß an einem System mit komplexem Funktionsangebot viel zu lange. Meist liegt das Angebot ungenutzt brach.

Mit dem Konzept "Arbeitskontext" sollen diese Schwierigkeiten überwunden werden. "Arbeitskontext" repäsentiert eine bereits von anderen getroffene Arbeitsvorbereitung für ein bestimmtes Problem. Man muß von vornherein betonen, daß nicht die Absicht besteht, mit dem Angebot eines "Arbeitskontext" zu einer schematischen, voreingestellten Problemlösung zu verleiten. Vielmehr soll das mittels "Arbeitskontext" aufbereitete Wissen als Anregung dienen, das Funktionsangebot des komplexen Systems in dieser oder ähnlicher Weise "auszubeuten". Ein an der Benutzerschnittstelle angebotener "Arbeitskontext" ist somit der Vorschlag für einen Plan, den man ganz oder teilweise übernehmen kann.

Der Dialog mit dem "Arbeitskontext"

Eine Benutzerschnittstelle kann unter mindestens zwei Aspekten gestaltet werden:

a) "Werkzeugschnittstelle": Welche Werkzeuge eines Anwendungssystems und welche Werkzeuginformationen stehen dem Benutzer zur Verfügung?
b) "Dialogschnittstelle": Wie kann der Benutzer im Dialog auf Teile des Anwendungssystems zugreifen?

Der "Arbeitskontext" ist eine Komponente der Werkzeugschnittstelle. Er repräsentiert die Menge aller im System vorhandenen Werkzeuge, die zur Bearbeitung einer Aufgabe notwendig und relevant sind. Wählt der Benutzer den Zugang zum Anwendungssystem über den "Arbeitskontext", so greift er nicht unmittelbar auf die Anwendungsprogramme zu, sondern mittels der im "Arbeitskontext" vorausgewählten Programme. Diese Form des vermittelten Zugriffs bedeutet keine Einschränkung, sondern dient lediglich der besseren Orientierung bei der Auswahl von Anwendungsprogrammen. Diese Form des Zugriffs erfordert jedoch eine besondere Dialogtechnik (vgl.Abb.4). In Abb.4 bedeutet "Arbeitskontext" = 'context' = Werkzeugschnittstelle.

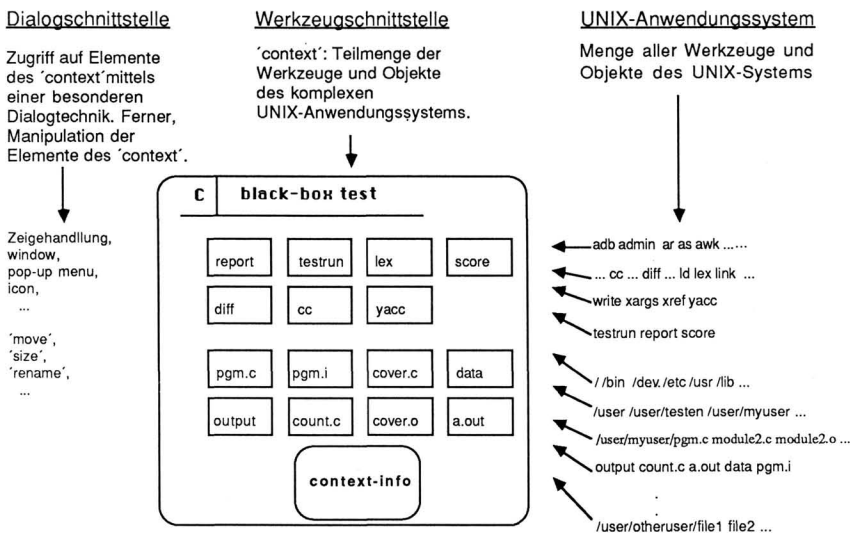

Abb.4: Zugriff auf den 'context' und Vorauswahl von Werkzeugen.

In Abb.4 wird ein Arbeitskontext zur Lösung eines Teilproblems im Rahmen des Testens von Software beschrieben:der "black-box"-Test als eine Form der "dynamischen Analyse". Bei der dynamischen Analyse wird ein Programm mit Testdaten ausgeführt; dabei werden die tatsächlichen Ausgaben mit den erwarteten Ausgaben verglichen.

Um Aussagen über die Qualität der Testdaten machen zu können, werden Abdeckungsmaße definiert, die beschreiben, bis zu welchem Grade bestimmte syntaktische Elemente eines Programms beim Testlauf ausgeführt wurden. Ein Beispiel für ein Abdeckungsmaß ist die Entscheidungsabdeckung, die beschreibt, welcher Prozentsatz aller Entscheidungen eines Programms durchlaufen wurde. Ein Test gilt dann als erfolgreich, wenn ein bestimmter, vorher festgesetzter Abdeckungsgrad erreicht ist.

Im Kontext "black-box test" sind Werkzeuge zusammengestellt, die helfen sollen, eine Entscheidungsabdeckung für ein gegebenes Programm zu finden. Dazu wird das Programm geeignet instrumentiert: nach jeder Entscheidung wird im source-Text ein Zählstatement eingefügt, das zählt, wie oft die entsprechende Entscheidung ausgeführt wurde. Die Instrumentierung erfolgt durch das Werkzeug "instrument". Damit die Instrumentierung erfolgreich durchgeführt werden kann, muß ein Programm vorliegen, das nur reinen c-source Text enthält. Dies erreicht man, indem man den c-Compiler mit einem geeigneteten Parameter aufruft ("cc -p"). Der Compiler erzeugt aus dem Source-Programm in der Datei "pgm.c" eine Version des Programms in der Datei "pgm.i", die mit Hilfe des Werkzeugs "instrument" instrumentiert werden kann. Die instrumentierte Fassung des Programms steht in der Datei "cover.c". Diese wird übersetzt zu einem lauffähigen Programm "pgm". Ein Testlauf wird gestartet mit dem Werkzeug "testrun", das das Programm "pgm" mit den in der Datei "testdata" enthaltenenen Testdaten aufruft. Die Ausgaben des Programms werden in die Datei "output" geschrieben und können mit Hilfe des UNIX-Werkzeugs "diff" mit den erwarteten Ausgaben in der Datei "expected-output" verglichen werden. Die Information über die Entscheidungsabdeckung wird mit dem Werkzeug "testrun" durch den Parameter "-e" erzeugt und die Datei "auswertung" geschrieben. Diese Information kann mit dem Werkzeug "report" aufbereitet dem Benutzer zur Verfügung gestellt werden.

Das Werkzeug "instrument" ist vom Entwickler des Kontext selbst entworfen worden. Es kann nur benutzt werden, um c-Programme zu instrumentieren. Die Erfahrung, die bei seiner Konstruktion gesammelt wurde, kann jedoch verwendet werden, um Werkzeuge zur Instrumentierung von Programmen in anderen

Programmiersprachen zu entwickeln. Diese Erfahrungen sind im "context-info" gesammelt und können bei Bedarf benutzt werden. Aus diesem Grunde sind auch Werkzeuge wie "yacc" und "lex", zwei UNIX-Werkzeuge zur Entwicklung von Programmen zur lexikalischen Analyse und zur Entwicklung von parsern, in den Kontext aufgenommen. Diese Werkzeuge sind verwendet worden zur Implemantation des Werkzeugs "instrument".

Für ein komplexes Anwendungssystem ist eine auf der Fenstertechnik aufbauende Dialogschnittstelle von Vorteil, da sie das Arbeiten an zusammenhängenden Teilaufgaben erleichtert. Fenster-orientierte Benutzerschnittstellen können nach den Prinzipien des objektorientierten Entwurfs realisiert werden. Das bedeutet, es existieren einige verallgemeinerte Dialogfunktionen (z.B. 'move','size','rename'), die auf alle Objekte der Benutzerschnittstelle (z.B. 'windows',icons') eine gleichartige Wirkung haben. Dies erreicht man, indem festgelegt wird, auf welchen Objekten der Schnittstelle mit welchen Funktionen gearbeitet werden darf. Die Objekte können nur mit Hilfe der vorher zugeordneten Werkzeuge manipuliert werden. Der Designer einer solchen Dialogschnittstelle ist dafür verantwortlich, daß Operationen, die einen ähnlichen oder gleichen Namen haben, ähnlich oder gleich auf die zugeordneten Objekte wirken.

Obwohl sich das Konzept des objektorientierten Entwurfs für die Gestaltung des Dialogs bewährt hat, steht es im Widerspruch zu dem Konzept, das der Benutzung der UNIX-Funktionen zugrunde liegt. Denn ein wesentlicher Grundsatz von UNIX ist, soweit wie möglich keine Annahmen darüber zu machen, auf welchen Objekten mit einem Werkzeug gearbeitet werden soll. Den Entwicklern von UNIX erschien es nicht sinnvoll, die für jeden Objekttyp gültigen Operationen zu definieren. Nimmt man dem Benutzer die Möglichkeit, Werkzeuge nach kreativem Einfall auf Objekte seiner Wahl anzuwenden, so geht eine Eigenschaft von UNIX verloren, die von erfahrenen Benutzern sehr geschätzt wird. Das Prinzip des objektorientierten Entwurfs steht also im Widerspruch zur UNIX-Philosophie.

Trotzdem soll auf die Vorteile des Dialogkonzepts, das dem Prinzip des objektorientierten Entwurfs entspricht, nicht verzichtet werden, wenn es dem Ziel dient, die Komplexität des

UNIX-Systems besser beherrschbar zu machen.

Das Dialogkonzept erlaubt eine Arbeitsvorbereitung, so daß der Zugriff auf ausgewählte Teile des UNIX-Systems voreingestellt werden kann. Diese Voreinstellung kann der Benutzer selbst vornehmen, indem er geeignete Zugriffsfunktionen benennt und in einer "box" zur Verfügung stellt; dies können aber auch andere Benutzer besorgen, um auf diese Weise Spuren ihrer Benutzungserfahrung zu hinterlassen.

Wenn der Benutzer auf die Benutzerschnittstelle schaut, so sieht er z.B. "Menus","icons","windows". Jedes "icon" repräsentiert ein Objekt, das der Benutzer "box" nennt. Der Begriff "box" ist passend, weil eine "box" zunächst nichts über den konkreten Inhalt verrät, solange sie geschlossen ist. Das wahrnehmbare "icon" ist somit lediglich eine graphish geschickt gestaltete Repräsentation einer geschlossenen "box".

Eine Benutzerschnittstelle bietet mindestens zwei Typen von "box" an: (1) eine "box" repräsentiert ein Element des UNIX-Systems (z.B. ein bestimmtes "file"); (2) eine "box" repräsentiert einen "context". Der Zugriff auf die verschiedenen Typen von "box" und das Arbeiten am Inhalt einer "box" sollen jedoch in einer einheitlichen Weise gestaltet werden. Das Gestaltungsprinzip hierfür ist die voreingestellte Auswahl von UNIX-Elementen.

Die "box" vom Typ "Element des UNIX-Systems":
Ist die "box" ein UNIX-file, so erscheint diese "box" auf dem Bildschirm als "icon", das noch durch den Namen des "file" besonders gekennzeichnet ist. Der Benutzer kann die "box" mittels Zeigehandlung (d.h. Positionierung eines "cursor" mittels Maus) öffnen. Auf dem Bildschirm erscheinen dann die für dies "file" vorausgewählten Zugriffs- und Manipulationsmöglichkeiten, etwa ein Editor oder ein Drucker oder ein Papierkorp. Die voreingestellten Zugriffsmöglichkeiten sind ebenfalls als "icon" symbolisiert, so daß der Benutzer wiederum mittels Zeigehandlung zugreifen kann. Wählt der Benutzer das "icon" Editor, so steht ihm auf den Bildschirm sofort der Inhalt des UNIX-files zur Verfügung, um darin zu editieren.

Die "box" vom Typ "context" wird vom Benutzer in gleicher Weise behandelt: Ist die "box" ein "context", so erscheint auf dem Bildschirm ebenfalls ein "icon", das durch den Namen des

"context" besonders gekennzeichnet ist. Mittels Zeigehandlung kann der Benutzer die "box" (d.h. den "context") öffnen. Auf dem Bildschirm erscheinen dann die für diesen "context" vorausgewählten UNIX-Elemente, z.B. files, Werkzeuge. Zusätzlich erscheint ein "icon", das den Zugriff auf besondere Erklärungen zu diesem "Arbeitskontext" ermöglicht. Der Zugriff auf Elemente des "context" wird wiederum mittels Zeigehandlung verwirklicht. Der Benutzer arbeitet dann im "context" an bestimmten Objekten des "context". Selbstverständlich stehen dem Benutzer Dialogfunktionen und ein Editor zur Verfügung, um den "context" seinen Bedürfnissen entsprechend anpassen zu können.

Entwicklungsprobleme

Erste Erfahrungen in der Realisierung der Benutzerschnittstelle haben gezeigt, das eine Reihe von Hindernissen zu überwinden ist.

Wir haben beobachtet, daß Software-Entwickler viel Zeit damit verbringen, sich neben den verfügbaren UNIX-Werkzeugen eigene Werkzeuge herzustellen, weil das Repertoire der Werkzeuge nicht ausreicht. Niemand hat jedoch einen Überblick darüber, welche Eigenentwicklungen entstehen und wie man vermeiden kann, daß solche Eigenentwicklungen immer wieder von Neuem gemacht werden.

Das Konzept "Arbeitskontext" soll sicherstellen, daß solche Mehrfachentwicklungen unterbleiben. Man kann jedoch das Benutzerschnittstellen-Konzept nur realisieren, wenn auch begleitende organisatorische Maßnahmen getroffen werden. Es muß in der Software-Entwicklungsabteilung einer der Ingenieure dafür gewonnen werden, sich um die Eigenentwicklungen anderer Benutzer zu bemühen. Seine Aufgabe besteht darin, die Eigenentwicklungen seiner Kollegen zu ausgereiften Werkzeugen weiterzuentwickeln. Die Kollegen selbst haben meist kein Interesse oder keine Zeit, ein selbsteintwickeltes Werkzeug professionell zu Ende zu entwickeln; denn sie begnügen sich meist mit einer pragmatischen, halbfertigen Lösung, die ihnen erst einmal weiterhilft. In solchen Entwicklungen stecken aber oft gute Ideen, die aufgearbeitet werden müßten und die auch andere zur Verwertung übernehmen könnten. Ein "Arbeitskontext" setzt sich somit in der Regel aus einer Mischung zusammen: UNIX-Werkzeuge,

Anpassungen solcher Werkzeuge und Eigenentwicklungen. Insgesamt aber entsteht ein ganzheitlicher Ansatz zur Lösungen eines komplexen Problems.

Ein weiteres Hindernis ist die Wissensakquisistion selbst, da eine mögliche Konkurrenzsituation im Betrieb daran hindert, die Erfahrungen der Experten aufzuarbeiten und anderen verfügbar zu machen. Dieses Hindernis kann nicht mit einer auch noch so gut gestalteten Benutzerschnittstelle überwunden werden. Hier helfen nur moderne Grundsätze einer kooperative Führung, die die Kollegen dazu bringt, im Team zusammenzuarbeiten, statt gegeneinander zu konkurrieren.

Außerdem hat es sich gezeigt, daß es außerordentlich mühsam ist, das für einen komplexen "Arbeitskontext" notwendige Wissen zusammenzutragen und die Information über das Arbeiten in einem solchen "context" aufzubereiten. Wenn es aber so schwierig ist, Expertenwissen über komplexe Problemlösungen aufzuarbeiten, dann ist es um so mehr wert, dies zu tun, weil das anschließend verfügbare Wissen einen multiplikativen Verbreitungseffekt hat. In der Software-Entwicklung existiert zur Zeit kein Mangel an Methoden, sondern Mangel an Wissen darüber, wie man diese Methoden mit verfügbaren oder zusätzlich entwickelten Werkzeugen anwenden kann. Wo immer dieses Anwendungswissen einmal erarbeitet worden ist, sollte man die Spuren sichern und zur Nachahmung anbieten.

Gespräche mit Entwicklern, die die Situation in der Software-Industrie kennen, haben gezeigt, daß ein Konzept, wie wir es mit "Kontext" vorschlagen, heute in der Industrie nicht eingesetzt wird. Das Management wählt eine Methode aus, mit der sich das Problem unter den gegebenen Bedingungen bearbeiten läßt. Diese Methode wird dann soweit wie möglich instrumentiert und die Mitarbeiter werden für die Arbeit mit der instrumentierten Methode geschult. Diese Art der Problemlösung wird gewählt, weil das Management leichter Kontrolle über den Stand der Arbeit bekommen will. Man kann mittels "Arbeitskontext" zusätzlich erreichen, daß die einmal ausgewählte Instrumentierung einer Methode weiterentwickelt wird; allen Mitarbeitern kann dieser Fortschritt an der Benutzerschnittstelle zugänglich gemacht werden.

Literaturverzeichnis

Dörner,D.(1979): Problemlösen als Informationsverarbeitung.
Stuttgart: Kohlhammer, 2.Auflage.

Hacker,W.(1978): Allgemeine Arbeits- und Ingenieurpsychologie.
Bern: Huber, 2.Auflage.

Wolfgang Dzida
Claus Hoffmann
Wilhelm Valder
GMD-F2.G2
Postfach 1240
D-5205 Sankt Augustin 1