Individuelle Arbeitsvorbereitungen am System UNIX

Wolfgang Dzida, Regine Freitag, Claus Hoffmann, Wilhelm Valder
St. Augustin

Zusammenfassung

Um Software-Entwicklern das Vorbereiten ihrer Arbeiten am System UNIX zu ermöglichen, werden Komponenten einer Benutzerschnittstelle (genannt ERGO-Shell) entwickelt. Ein Konzept für problem-orientierte Arbeitsvorbereitungen wird vorgestellt. Es wird untersucht, wie das Angebot der Schnittstellen-Komponenten so genutzt werden kann, daß auf der Grundlage von vorbereitenden Arbeiten Arbeitsverfahren für die Software-Produktion entstehen können. Organisatorische Randbedingungen hierfür werden thematisiert.

1. Problemsituation

Arbeitsvorbereitung ist Voraussetzung für eine methodische Vorgehensweise in der Software-Entwicklung. Wegen der Komplexität der Problemlösungen (etwa beim "Programmieren im Großen") muß gewährleistet sein, daß methodische Lösungsansätze für die technische Durchführung vorbereitet und für spätere Wiederverwendung bereitgestellt werden können. Die Wiederverwendung entlastet davon, sich in bereits durchdachte Teile der Problemlösung erneut einarbeiten zu müssen. Die Aufmerksamkeit kann sich dann auf die noch offenen Teile konzentrieren oder auf die Arbeitsausführung, etwa die Formulierung eines komplexen Kommandos. Benutzer-Zielgruppe unserer Untersuchungen sind Software-Entwickler, und zwar solche, die das System UNIX als Produktionsbasis einsetzen.

UNIX wird weltweit in der Software-Entwicklung eingesetzt. Es bietet zwar Werkzeuge an, die dem Benutzer bei der Arbeitsvorbereitung helfen können (vgl. Abschnitt 3.1); dieses Angebot wird jedoch nicht an der Benutzerschnittstelle (UNIX-Shell) unterstützt. Die Unterstützung ist auf die Möglichkeiten beschränkt, die ein Dialog mit einer Kommandozeile des Bildschirms zu bieten vermag. Somit sind die Reaktionen des Benutzers während des Dialogs stark vom momentanen Ereignisanfall in der Kommandozeile beeinflußt. Die durch UNIX gebotene Unterstützung dient eher den administrativen Teilen der Arbeitsvorbereitung; dagegen ist die Vorbereitung von Problemlösungen bisher kaum beachtet worden.

Wir verstehen unter problem-orientierter Arbeitsvorbereitung die Bereitstellung von Teilen eines technischen Lösungsansatzes (z.B. Werkzeuge und Dateien), wobei der inhaltliche Aufwand hauptsächlich auf die Instrumentierung einer Software-Entwicklungs-Methode konzentriert ist. Weil das UNIX-Repertoire unvollständig ist, sind in den Lösungsansätzen der Benutzer oft gute Ideen verborgen, die aufbewahrt, weiterentwickelt und allgemein verfügbar gemacht werden sollen. Selbst wenn der Lösungsansatz nur ein Fragment ist, soll er aufgehoben werden können, um als vorläufig entwickelter Teil einer komplexeren Lösung später bereit zu stehen.

Ferner soll der Benutzer an der Zusammenstellung von vorbereiteten Lösungsfragmenten zu komplexen "Arbeitsverfahren" beitragen können. Ein Arbeitsverfahren ist eine Ansammlung von ausgereiften Vorschlägen zu einer Problemlösung für die Software-Entwicklung, einschließlich der Instrumentierung dieser Methode. Neben mehrfach verwendbaren Werkzeugen enthält ein Arbeitsverfahren auch spezifische Werkzeuge, die auf besondere Problemfälle oder eine Abwandlung der Methode zugeschnitten sind. In der professionellen Software-Entwicklung soll ein Arbeitsverfahren für eine Klasse von Problemen allgemein akzeptiert sein.

Die Struktur solcher methodischer und technischer Lösungs-Konzepte soll an der UNIX-Benutzerschnittstelle gut kommentiert dargestellt werden (präsentiert als "Work Context"). Arbeitsverfahren sollen bei der Planung individueller Lösungswege Hilfen und Anregungen geben. Schwierigkeiten entstehen bei der allmählichen Zusammenstellung von Ungefähr-Lösungen zu Arbeitsverfahren und deren Pflege (Dzida et al., 1987).

Wegen dieser Schwierigkeiten müssen an der Benutzerschnittstelle Vorkehrungen getroffen werden, damit die Benutzer eine vorausschauende Arbeitsvorbereitung nicht als zusätzliche Last empfinden. Insofern wird Schnittstellengestaltung als Arbeitsgestaltung aufgefaßt (vgl. Hacker, 1987). Eine enge Zusammenarbeit mit den Benutzern, die hiervon profitieren sollen, ist unerläßlich. Das GMD-Projekt ERGO (Ergonomie in der Software-Technologie) entwickelt eine Benutzerschnittstelle, genannt ERGO-Shell, die zur Unterstützung von Arbeitsvorbereitungen beiträgt.

2. Ein Entwicklungs-Konzept

2.1 Arbeitsstile von Software-Entwicklern

Aus der Literatur sind zwei Arbeitsstile bekannt, die man als gegensätzliche Ausprägungen auf einer Skala für Arbeitsstile betrachten kann: der "hacker" (von Weizenbaum (1976) auch als "zwanghafter Programmierer" beschrieben) und der "kooperative Programmierer" (Yourdon, 1977).

"zwanghafter" Programmierer	"kooperativer" Programmierer
er arbeitet gern 'drauf los'	er macht ausgedehnte Vorarbeiten
er produziert kaum Dokumente, die über die Entwicklung Auskunft geben	er arbeitet umsichtig, d.h. mit Rücksicht auf die Produkte seiner Kollegen
er ist kaum an anderen Aufgaben interessiert	er erledigt nebenher administrative Arbeiten

Der "kooperative" Programmierer ist vermutlich in seiner extremen Ausprägung ebenso selten zu finden wie der "hacker"; beide Typen sind überzeichnet dargestellt.

Diese Arbeitsstile sind jedoch auch in Arbeitssituationen beobachtet worden, in denen nicht programmiert wurde (z.B. Triebe, 1977), so daß wir verallgemeinernd unterscheiden zwischen der "momentanen Strategie" und der planenden Strategie (vgl. Hacker, 1986, S. 332 ff.). "Momentan" bedeutet im Extremfall, daß Arbeitsschritte und ihre Ergebnisse kaum gedanklich vorweggenommen werden, während "planend" ein umsichtiges Vorgehen erkennen läßt, bei dem die wesentlichen Zusammenhänge des Konstruktionsvorganges gedanklich durchdrungen werden.

Mit der Entwicklung der ERGO-Shell soll versucht werden, die planende Strategie besser zu unterstützen, da empirische Befunde darauf hinweisen, daß diese Arbeitsweise höhere Effektivität bei geringerer Beanspruchung mit sich bringt (Hacker, 1986). Benutzer sollen sich vor der Arbeitsausführung eine bessere Übersicht über die inhaltlichen Voraussetzungen der Ausführung verschaffen können. Schematische Lösungswege lassen sich eher vermeiden, wenn man Hinweise auf Alternativen bekommt. Teile von Problemlösungen sollen auch später leichter zugreifbar sein. Die Machbarkeit komplexer Lösungen soll besser eingeschätzt werden. Erfahrungstransfer zwischen Kollegen soll unterstützt werden.

2.2 Kategorien der Arbeitsvorbereitung

Software-Entwickler bewältigen meist sehr komplexe Probleme; diese sind dadurch gekennzeichnet, daß es kein allgemein anerkanntes Verfahren zu ihrer Bewältigung gibt. In der Regel sind Teil-Aufgaben noch nicht hinreichend geklärt; das Repertoire der Werkzeuge ist unvollständig; die Sequenz der Arbeitsschritte liegt noch nicht fest. Probleme dieser Art werden in der Psychologie als Probleme mit Synthesebarriere bezeichnet (Dörner, 1979). Ein momentaner Arbeitsstil wäre hierfür ungeeignet.

Bei der Analyse von Arbeitsschritten, die der planenden Strategie zuzuordnen sind, werden die Kategorien "vorbereitende Kontrolle" und "vorbereitende Tätigkeit" unterschieden (Dzida, 1986). Arbeitstätigkeiten von Software-Entwicklern werden daraufhin untersucht, welcher dieser Kategorien sie angehören. Arbeitsvorbereitungen im Sinne der "vorbereitenden Kontrolle" werden dadurch unterstützt, daß am Bildschirm für eine bessere Übersicht über Arbeits- und Systemzusammenhänge gesorgt wird. Der Benutzer kann z.B. vorbeugend prüfen, ob wesentliche Teile für eine zusammengesetzte Kommando-Eingabe bereitstehen. Zur Unterstützung von vorbereitenden Tätigkeiten werden technische Komponenten entwikkelt, die eine Wiederverwendung einmal geleisteter Arbeit ermöglichen; wiederverwendbare Lösungen (auch Fragmente hiervon) sollen auch an die neuen Ausführungsbedingungen leicht angepaßt werden können.

Arbeitsvorbereitung dient somit dem besseren Erkennen oder Wiedererkennen von Zusammenhängen sowie der Wiederverwendung bereits entwickelter Lösungsansätze. Der Ausführungszeitpunkt wird weniger vom momentanen Systemzustand diktiert als vom Ausmaß der erforderlichen vorbereitenden Tätigkeiten des Benutzers. Arbeitsvorbereitung entlastet den Benutzer im Moment der Arbeitsausführung. Ein Kommando braucht nicht atomistisch zerlegt zu werden, nur um es Schritt für Schritt einzugeben; der Benutzer kann größere Dialogabschnitte planen. Eingabefehler, die auf mangelnde logische Durchdringung eines zusammengesetzten Kommandos beruhen, werden somit eher vermieden.

2.3 Integration von Vorbereitung und Ausführung

Arbeitsvorbereitungen hinterlassen "Spuren", die später bei der Arbeitsausführung wieder aufgenommen werden können. Individuelle Vorbereitung und Ausführung sollen am Bildschirmarbeitsplatz integrierbar sein. Dies kann technisch durch Fenstersysteme unterstützt werden. In verschiedenen Anzeigenbereichen des Bildschirms hat der Benutzer die Möglichkeit, Übersichten einzurichten, die das spätere Wiedererkennen schon einmal geordneter oder durchdachter Lösungsansätze erleichtern helfen (Dateien strukturieren, Dialogabschnitte aufbewahren, beispielhafte Lösungen speichern, komplizierte Kommandosequenzen kommentieren usw.). Die ERGO-Shell gewährleistet, daß bei der Arbeitsausführung (Kommandoeingabe) die vorher geordneten oder vor-ausgewählten Eingabeobjekte so berücksichtigt werden, daß überflüssiger (fehleranfälliger !) Eingabeaufwand vermieden wird.

Darüber hinaus erscheint es als zweckmäßig, Arbeitsvorbereitungen von Mitarbeitern eines Projekt-Teams so aufzubereiten, daß sie in die Entwicklungsarbeiten des Projekts integrierbar sind. Arbeitsvorbereitungen können bei der späteren Ausführung wesentlich effizienter ausgenutzt werden, wenn diese Arbeits-"Spuren" in geeigneter Form aufbewahrt werden. Sollen geordnete oder schon einmal durchdachte Lösungsansätze

auch für andere wiederverwendbar gemacht werden, so müssen solche Zwischenprodukte in besonderer Weise aufbereitet werden. Im einfachsten Fall kann es z.B. nützlich sein, eine bewährte Folge von Kommandos, die als Makro aufbewahrt werden soll, durch erläuternde Kommentare zu ergänzen. Einem Kollegen wird dadurch die Wiederverwendung oder ggf. die Weiterentwicklung des Lösungsvorschlags erleichtert.

Zweifellos können Arbeitsvorbereitungen dieser Art den Erfahrungstransfer im Projekt-Team fördern. Aus wiederverwendbaren Teil-Lösungen oder Zwischenprodukten können im Laufe der Zeit reifere Produkte entstehen, die den Stand der Erfahrung widerspiegeln, den ein Projekt-Team oder eine Entwicklungsabteilung bei der Instrumentierung von Methoden der Software-Entwicklung erworben hat. Dies kann sogar soweit führen, daß erprobte Arbeitsverfahren des Software-Engineering entstehen, die als Teile einer UNIX Entwicklungs-Umgebung allgemein akzeptiert werden (vgl. Abschn. 4).

Zweifel an der Nützlichkeit dieser Art von individueller und kollektiver Arbeitsvorbereitung lassen sich zumindest theoretisch leicht ausräumen. Muthig (1983) und Schönpflug (1986) sehen in der externen Speicherung von Wissen nicht nur eine Erweiterung individueller kognitiver Kapazitäten, sondern auch einen Fortschritt für das kollektiv verwendbare Wissen. Dennoch bleiben viele Fragen zu beantworten: "Welches Wissen ist relevant?" "Lohnt sich die Speicherung, ohne daß das Wissen zum Zwecke der Wiederverwertung besonders aufbereitet wurde?" "Wer sorgt für die Fortschreibung und Pflege von Wissen dieser Art?"

Mit der Entwicklung der ERGO-Shell wird versucht, auf diese Fragen praktische Antworten zu finden. Dies kann nur auf empirischem Wege erreicht werden; d.h. durch Implementierung von Komponenten einer Benutzerschnittstelle, die das Aufbewahren und Wiederverwenden von individuellen Arbeitsvorbereitungen unterstützen. Darüber hinaus ist geplant, mit einem Kooperationspartner der Software-Industrie die Integration von individueller Arbeitsvorbereitung in die Arbeitsausführung anderer Mitarbeiter eines Projekt-Teams zu erproben, so daß geeignete technische und organisatorische Unterstützungen gegeben werden können (vgl. Abschn. 4).

3. Technische Realisierung

3.1 Arbeitsvorbereitung an den klassischen UNIX-Shells

Auch am UNIX-System kann man gewisse Arbeitsvorbereitungen treffen. Einige Beispiele:

- Durch das Setzen von Variablen können vorsorglich Einstellungen am System vorgenommen werden. Beispielsweise kann der Benutzer definieren, welchen der vielen Editoren er als Standard-Editor wünscht.

- Das Werkzeug ´make´ kann den Benutzer beim Ändern von Programmsystemen unterstützen, indem es ihn davon entlastet, alle die von einer Änderung eines Programms mitbetroffenen Module feststellen und aktualisieren zu müssen.
- Mit Hilfe des Werkzeugs ´ar´ kann eine ´library´ eingerichtet und verwaltet werden. Sie erspart dem Benutzer, alle für das Binden benötigten Dateien einzeln angeben zu müssen.

Diese Beispiele lassen erkennen, daß Arbeitsvorbereitungen meist auf solche Teile von Aufgaben beschränkt sind, die mit dem eigentlich zu lösenden Problem des Benutzers wenig zu tun haben. Es handelt sich eher um administrative Unterstützungen, die gewährleisten sollen, daß der Dialog mit dem System von problemfernen Dialogschritten entlastet wird (sog. "aufgabenangemessener Dialog" im Sinne von DIN 66 234, Teil 8). Diese Art der Arbeitsvorbereitung ist wichtig. Mit der Entwicklung der ERGO-Shell setzt die Unterstützung der Arbeitsvorbereitung jedoch beim eigentlichen Problemlösungsprozess des Benutzers an.

3.2 Komponenten der ERGO-Shell

Die ERGO-Shell ist eine technische Ergänzung zur klassischen UNIX-Shell. Abbildung 1 zeigt eine Anordnung der Komponenten der ERGO-Shell auf dem Bildschirm. Im folgenden werden einige dieser Komponenten nur soweit vorgestellt, als sie für die Unterstützung der Arbeitsvorbereitung bedeutsam sind (vgl. Abschnitt 3.3). Die Komponenten sind in englischer Sprache benannt, da dies die übliche Sprache am System UNIX ist.

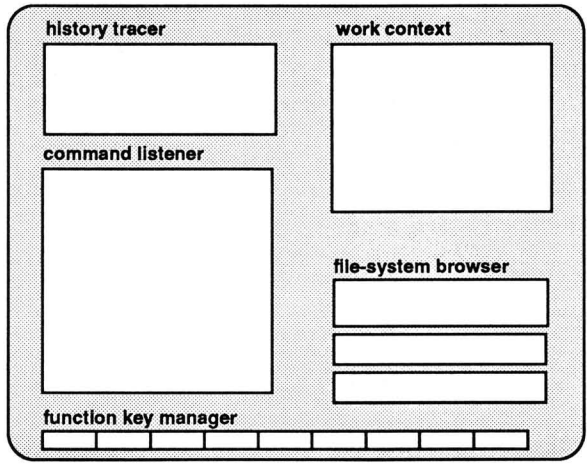

Abbildung 1: Komponenten der ERGO-Shell: history tracer, work context, command listener, file-system browser, und function-key manager

3.2.1 File-System Browser

Der File-System Browser gibt dem Benutzer ein Werkzeug in die Hand, um den hierarchischen Dateibaum von UNIX zu durchsuchen. Jedes "directory" erscheint in einem Fenster, in dem die entsprechenden Dateinamen gelistet werden. Mit Hilfe von Menüs kann der Benutzer im Dateibaum navigieren. Mit Hilfe der Maus können einzelne oder auch mehrere Dateien selektiert werden. Für selektierte Dateien sind durch Menüs vordefinierte Funktionen abrufbar (z. B. remove, copy, rename).

3.2.2 History Tracer

Der History Tracer macht die Dialoggeschichte für den Benutzer zugreifbar. Der Benutzer kann veranlassen, daß seine Eingaben aufgezeichnet werden. Er erhält dann einen dialogisch einfachen Zugriff auf bereits abgegebene Kommandos, um sie z.B. erneut ausführen zu lassen.

3.2.3 Macro-Editor

Der Macro-Editor ermöglicht die Verwendung von Makros in der Kommandosprache. Makros unterscheiden sich von Kommandoprozeduren dadurch, daß sie nicht direkt ausgeführt, sondern in expandierter Form im Kommando-Editor dem Benutzer zur Verfügung gestellt werden. Hier können sie vor der Ausführung überprüft und ediert werden.

3.2.4 Work Context

Unter einem "Work Context" verstehen wir die Präsentation eines Arbeitsverfahrens in einem Fenster des Bildschirms. Dort kann der Benutzer Anregungen und Vorschläge für die Arbeitsplanung bekommen. In diesem Fenster werden ein komplexes Funktionsangebot sowie die dazugehörigen Dateien angeboten, die sich zur Bearbeitung einer Aufgabe als nützlich erwiesen haben. Um dieses Angebot anwendbar oder wiederverwendbar zu machen, bietet der "Work Context" ein "Context Info" an. Auf diese Weise kann aufbereitetes konzeptionelles Wissen über die Anwendung eines komplexen Funktionsangebots an Benutzer weitergegeben werden, so daß sie sich in bestimmte Aufgaben besser (wieder)einarbeiten können und gesammelte Erfahrungen nutzen.

3.3 Arbeitsvorbereitungen in der ERGO-Shell

Eine sehr geschätzte Eigenschaft von UNIX ist sein File-System, das eine baumstrukturierte Ordnung der Dateien des Benutzers aufweist. Wenig geschätzt ist, daß man sich trotz der hierarchischen Ordnung kaum zurechtfindet, weil keine Übersichten angeboten werden. Arbeitet der Benutzer mit mehreren Dateien aus verschiedenen "directory", so verschlimmert sich diese Situation.

Der an der ERGO-Shell verfügbare "File-System Browser" unterstützt das Suchen in der Dateistruktur. Man kann sich Dateien aus verschiedenen Ästen anzeigen lassen und sie vorsorglich für eine spätere Verwendung besonders kennzeichnen. Diese Vorauswahl entlastet den Benutzer, so daß er sich bei der späteren Ausführung eines komplex zusammengesetzten Lösungskonzepts (repräsentiert durch die Kommando-Eingabe) auf wichtigere funktionelle Teile der Lösung konzentrieren kann.

Hat sich eine Folge von Arbeitsschritten bewährt, so kann es als lohnenswert erscheinen, diese aufzubewahren, um sie später noch einmal zu verwenden, ohne die "geistige Rüstzeit" erneut aufbringen zu müssen, die bei der ersten Ausführung erforderlich war. Diese "Spuren" der Arbeit sichert der "History Tracer", wenn der Benutzer es will. Er erhält Einblick in die Dialoggeschichte und kann eine Folge bewährter Dialogschritte für eine spätere Verwendung kommentieren und abspeichern.

In Ergänzung zum "History Tracer" stellt die ERGO-Shell einen "Makro Editor" zur Verfügung. Bewährte Kommandofolgen, die der "History Tracer" bereitstellt, können mittels "Makro-Editor" für eine Wiederverwendung besonders aufbereitet werden, indem sie mit einer Kontrollstruktur versehen werden. Im Unterschied zu Kommandoprozeduren, die anstelle von Makros oft verwendet werden, behält der Benutzer bei einem Makro eine gute Übersicht über die Ablaufstruktur des Lösungskonzepts. Einmal getroffene Arbeitsvorbereitungen lassen sich somit später an leicht veränderte Ausführungsbedingungen anpassen; Kommandoprozeduren sind zwar auch anpaßbar, aber die Erfahrung zeigt, daß dies kaum jemand tut, da sie den Anschein von endgültigen Lösungen haben.

Auch die Erfahrungen mit wiederverwendeten Makros kann der Benutzer vorausschauend nutzbar machen. Er wird darin unterstützt, diese Lösungsfragmente zu einem komplex zusammengesetzten "Work Context" auszubauen. Es wird angestrebt, einen "Work Context" gut zu kommentieren (siehe "Context Info") und die Tauglichkeit für die Wiederverwendung zu prüfen. Diese Komponente der ERGO-Shell soll dazu beitragen, den Benutzer von der verfrühten Entwicklung von Kommandoprozeduren abzuhalten. Ihr Anschein, endgültige Lösungen zu sein, verleitet leicht zur schematischen Anwendung. Statt dessen wird der Umgang mit "Ungefährlösungen" unterstützt, um Flexibilität in der Planung und Anpassungsbereitschaft zu fördern.

3.4 Integration der Schnittstellen-Komponenten

Die Arbeitsvorbereitungen des Benutzers zahlen sich aus, wenn er an die Arbeitsausführung geht. Der "Command Listener" ist die zentrale Ausführungs-Komponente der ERGO-Shell (vgl. Abb. 1). Hier kann der Benutzer sämtliche vorausgewählten Teile einer Lösung zusammenführen. Alle Kom-

ponenten der ERGO-Shell sind technisch so integriert, daß dem Benutzer unnötiger Eingabeaufwand erspart bleibt.

4. Auf dem Wege zu Arbeitsverfahren

Vorgeschriebene Arbeitsverfahren können beim Benutzer den Eindruck erwecken, daß die Software-Entwicklung durch weitere Reglementierungen diszipliniert werden soll. Dagegen wird hier unter einem Arbeitsverfahren eine Sammlung von Problemlösungs-Vorschlägen verstanden, die der Benutzer bei der individuellen Arbeitsvorbereitung berücksichtigen kann.

Jeder Benutzer organisiert seine Arbeitsweise weitgehend selbst. Der eine arbeitet vorausschauender, umsichtiger, der andere weniger. Die individuelle Arbeitsweise sollte nicht durch Vorschriften unnötig reglementiert werden. Auch die Arbeit im Projekt-Team sollte nicht von Vorschriften beherrscht werden. "Wir können kooperatives Verhalten nicht durch Vorschriften erzwingen, aber durch unser Verhalten fördern. Und wir können kreative Lösungen nicht aus Methoden ableiten oder mit Werkzeugen generieren. Was wir aber machen können ist, mit Hilfe technischer und organisatorischer Maßnahmen das Entwicklungsmilieu so zu gestalten, daß einige Ergebnisse wahrscheinlicher eintreten als andere" (Keil-Slawik, 1988).

Technische Maßnahmen, die das wünschenswerte Entwicklungsmilieu fördern, sind an der ERGO-Shell demonstrierbar. Benutzer, die unter einer klassischen UNIX-Shell gelitten haben, werden einige Vorzüge der ERGO-Shell allmählich nutzen lernen und damit von selbst zu mehr vorausschauender und umsichtiger Arbeitsvorbereitung finden als es an einer klassischen Shell möglich ist.

Eine umsichtige und vorausschauende Arbeitsvorbereitung des einzelnen Software-Entwicklers reicht jedoch nicht aus, um sicher zu stellen, daß seine Lösungsansätze aufgearbeitet und anderen zugänglich gemacht werden. Im Projekt-Team oder in der Entwicklungsabteilung muß ein Mitarbeiter gewonnen werden, der die Eigenentwicklungen der Kollegen und deren bewährte individuelle Arbeitsvorbereitungen zu kollektiv wiederverwendbaren Lösungs-Konzepten aufbereitet und pflegt. Ein einzelner Entwickler gibt zwar den Anstoß zur Entwicklung oder Fortentwicklung, braucht aber nicht allein den Aufwand auf sich zu nehmen, der mit der Aufbereitung zum Zwecke der Wiederverwendung verbunden ist.

Der Urheber eines Lösungsansatzes hat selbst meist keine Zeit oder kein Interesse, das Fragment professionell zu Ende zu entwickeln; denn er begnügt sich mit der halbfertigen, pragmatischen "ad-hoc"-Lösung, weil sie erst einmal weiterhilft. Es gilt jedoch, das in solchen Lösungen vorhandene Expertenwissen aufzuarbeiten.

5. Empirische Überprüfung

Empirische Überprüfung heißt zweierlei: 1. die ERGO-Shell implementieren, um zu zeigen, was machbar ist; 2. die Benutzer fragen, um herauszufinden, was nützlich ist.

Die ERGO-Shell wird als Prototyp für BSD-UNIX unter Verwendung des X-Window-Systems entwickelt. Die Benutzerschnittstelle wird einer Stichprobe von Benutzern zur Erprobung angeboten, um Partizipation an den vorgeschlagenen Design-Entscheidungen zu ermöglichen. Für die Beobachtung der Probenutzung wird das inzwischen implementierte wissensbasierte Interpretationssystem eingesetzt (Eisert, Olbrich, Werner, 1988). Es unterstützt modellgeleitete, empirische Untersuchungen über Mensch-Rechner-Dialoge, so daß Schwierigkeiten, die Benutzer mit der Benutzerschnittstelle und dem System haben, aufgedeckt werden.

Die Probleme, die sich mit der Entwicklung von Arbeitskontexten ergeben haben (vgl. Dzida et al., 1987, letzter Abschnitt), werden auf evolutionärem Wege zu lösen versucht. Wie in Abschnitt 3.3 ausgeführt, sollen einzelne Komponenten der ERGO-Shell dazu beitragen, daß aufeinander aufbauende Teile von Arbeitsvorbereitungen zu einem Arbeitsverfahren zusammengesetzt werden können, wenn für eine komplexe Aufgabenerledigung unter UNIX genügend Anwendungserfahrungen gesammelt worden sind. Ob Benutzer dieses Angebot der ERGO-Shell tatsächlich nutzen werden, wird sich nur im Feldversuch bei einem Software-Hersteller klären lassen. Wir wollen nicht das Problemlösen der Benutzer gestalten, aber die technischen Voraussetzungen zu schaffen versuchen.

6. Literatur

DIN 66 234, Teil 8: Bildschirmarbeitsplätze - Grundsätze ergonomischer Dialoggestaltung. Beuth, Berlin, 1988.

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

Dzida, W.: Computer assisted knowledge acquisition: Towards a laboratory for protocol analysis of user dialogues. In: F. Klix and H. Wandke (eds): Man-Computer Interaction Research: MACINTER-I. North Holland, Amsterdam, 1986, 139-150.

Dzida, W., Hoffmann, C. und Valder, W.: Der 'Arbeitskontext' als Komponente der Benutzerschnittstelle. In: W. Schönpflug und M. Wittstock (Hrsg.): Software-Ergonomie '87. Teubner, Stuttgart, 1987, 87-97.

Eisert, C., Olbrich, R. und Werner, J.: Wissensbasierte Interpretation von Mensch-Computer-Dialogen am System UNIX. Arbeitspapiere der GMD, Nr. 326, St. Augustin,1988.

Hacker, W.: Arbeitspsychologie. Huber, Bern, 1986.

Hacker, W.: Software-Ergonomie - Gestalten rechnergestützter geistiger Arbeit?! In: W. Schönpflug und M. Wittstock (Hrsg.): Software-Ergonomie'87. Teubner, Stuttgart, 1987, 31-54.

Keil-Slawik, R.: Die Gestaltung des Unsichtbaren. Computer Magazin, 7-8, 1988, 39-41.

Muthig, K.P.: Externe Speicher: Implikationen für Modellvorstellungen vom menschlichen Gedächtnis. In G. Lüer (Hrsg.): Bericht über den 33. Kongreß der Deutschen Gesellschaft für Psychologie in Mainz 1982. Hogrefe, Göttingen, 1983, 252-259.

Schönpflug, W.: The trade-off between internal and external information storage. Journal of Memory and Language, 25, 1986, 657-675.

Triebe, J.K.: Entwicklung von Handlungsstrategien in der Arbeit. Zeitschrift für Arbeitswissenschaft, 31, 1977, 221-228.

Weizenbaum, J.: Computer Power and Human Reason. From Judgement to Calculation. New York, 1976.

Wolfgang Dzida
Regine Freitag
Claus Hoffmann
Wilhelm Valder
GMD-Institut für Systemtechnik
Schloß Birlinghoven
5205 St. Augustin 1