Alle sprechen von benutzerfreundlichen Computern und Software-Ergonomie. Was damit wirklich gemeint ist und was für Voraussetzungen dafür erfüllt sein müssen, erklären zwei Spezialisten vom Institut für Informatik der Universität Stuttgart. Computersysteme der Zukunft werden einige gute Eigenschaften aufweisen, die heutigen Geräten noch weitgehend fehlen. Die Rede ist nicht von irgendwelchen neuen Hardware-Features, sondern von Benutzerschnittstellen, die das Prädikat «ergonomisch» verdienen. Die vier Haupteigenschaften sind ein konsistentes Verhalten bei gleichartigen Interaktionen, die Fähigkeit, wohldosierte Benutzerhilfe anzubieten, die Anpassungsfähigkeit an den Erfahrungs- und Wissensstand des Benutzers und eine flexible Fehlerbehandlung.

Was ist eigentlich Software-Ergonomie?

Von Joachim Bauer und Michael Herczeg

Damit die im Vorspann genannten Merkmale realisierbar werden, braucht es im System eine umfangreiche Wissensbasis, die während des Betriebs laufend aktualisiert wird.

Benutzer erwarten vom Computer konsistentes Verhalten: Gleichartige Tastenfolgen sollen zum Beispiel stets gleichartige Interaktionen bewirken. Die Verläßlichkeit des Dialogverhaltens ist eine Voraussetzung für unbelastetes Arbeiten. Diese *innere Konsistenz* eines Systems ist leider nicht immer gewährleistet, da sein Verhalten häufig an mehreren Stellen des Programms festgelegt wird, die sich nur mühevoll konsistent halten lassen.

Bei der Konzeption wissensbasierter Systeme hingegen ist das Wissen (und damit auch das Verhalten) in klar isolierbaren Teilen festgehalten, auf die von verschiedensten Stellen Bezug genommen werden kann. Dies garantiert konsistentes Verhalten.

Bei der Benutzung verschiedener Systeme muß der Anwender in der Regel jedes dieser Systeme wieder nahezu von Grund auf neu erlernen – von *Konsistenz zwischen mehreren Systemen* kann meistens keine Rede sein. Dazu benötigt man *integrierte Systeme*, die sich derselben Interaktionstechnik, unabhängig von der Anwendung, bedienen. Eine Lösung sind *anwendungsneutrale Benutzerschnittstellen*, die den Zugang zu allen benutzten Anwendungssystemen in derselben Art und Weise ermöglichen [1, 2]. Die Benutzerschnittstelle wird so zu einer weitgehend eigenständigen Systemkomponente, in der anwendungsneutrales Wissen über Methoden und Regeln der Kommunikation repräsentiert wird. Die Isolation dieses Wissens über die Kommunikation macht die Benutzerschnittstelle nicht nur für den Systemdesigner überschaubarer und leichter änderbar, sondern erlaubt auch die Modifikation der Benutzerschnittstelle durch den Benutzer selbst.

Joachim Bauer, Michael Herczeg, Forschungsgruppe INFORM, Institut für Informatik, Universität Stuttgart.

Der Schreibtisch auf dem Bildschirm

Konsistenz mit der bekannten manuellen Arbeitsumgebung bieten sogenannte *metaphorische Systeme*. Diese zeigen auf dem Bildschirm symbolisch die Arbeitsumgebung, also gewohnte Arbeitsobjekte wie Dokumente, Ordner, Schränke oder den Papierkorb als Icons (Piktogramme). Die Bildschirmfläche stellt den Schreibtisch dar. Mit Hilfe eines Zeigeinstruments können diese Objekte ausgewählt, bewegt und mit einem Editor bearbeitet werden [3]. Die Wirkungen sind sofort sichtbar und können somit auch kontrolliert werden. Man spricht deshalb auch von *direkter Manipulation* [4] oder vom *WYSIWYG-Prinzip* (What You See Is What You Get). Häufig werden für die unterschiedlichen Bildschirmdarstellungen und Arbeitskontexte verschiedene Fenster verwendet, die sich auf dem Bildschirm gegenseitig überlappen können (Bild 1).

Die Piktogramme müssen natürlich für den Benutzer verständlich sein und aktuelle Bezüge zur Anwendung herstellen. Werden sie manipuliert, müssen die durch sie repräsentierten Arbeitsobjekte entsprechend mitverändert werden.

Die Hilfeleistungen, die das System dem Benutzer anbietet, dürfen nicht dazu dienen, Mängel auszugleichen, die auf einer Fehlkonzeption des Systems beruhen. Allerdings lassen sich komplizierte Anwendungen, die sich durch hohe Funktionalität oder Funktionen mit vielen verschiedenen Parametern auszeichnen, nicht immer durch leichtverständliche Systeme realisieren. Beispielsweise können metaphorische Systeme den Dialog zwar durchsichtiger machen; da man aber aus einem Namen oder einem Piktogramm die ihm zugeordnete Aktion oft nur erahnen kann und sich nicht alle Details aus der Bezeichnung ergeben, sind *Hilfeleistungen durch das System* unumgänglich.

Wohldosierte Hilfe

Man unterscheidet einerseits zwischen aktiver und passiver Hilfe, anderseits zwischen statischer und dynamischer Hilfe.

Passive Hilfe muß der Benutzer explizit anfordern. Sie muß dem Benutzer die Formulierung von Anfragen erlauben, mit denen er schnell zu gezielter Information kommt. Das System sollte auch Fragen beantworten können, die nicht auf ein bestimmtes Kommando zurückgeführt werden können, indem es die notwendige Kommandofolge herausfindet und vorschlägt. Ein Beispiel für ein passives Hilfesystem, das Anfragen in Teilprobleme zerlegt und dem Benutzer schrittweise vorführt, ist das Hilfesystem PASSIVIST (Bild 2 [5]).

Aktive Hilfe kommt dann zum Einsatz, wenn der Benutzer einen Fehler macht beziehungsweise wenn das System bemerkt, daß der Benutzer Hilfe braucht. In sehr vielen Fällen kennt der Benutzer bestimmte Kommandos gar nicht. Ein aktives Hilfesystem kann die einem Benutzer bekannten Aktionen herauskristallisieren und Hilfe anbieten, wenn dieser bestimmte Kommandos nicht benutzt, obwohl dies sinnvoll wäre. Das Hilfesystem AKTIVIST (Bild 3 [6, 9]) ist ein Prototyp eines aktiven Hilfesystems, das dem Benutzer eines Texteditors vereinfachende Operationen erklärt, die er nie braucht. Wenn er also zum Beispiel Wörter zeichenweise löscht, sagt ihm das System, daß es auch eine Taste gibt, mit der er ein ganzes Wort auf einmal löschen kann.

Statische Hilfe gibt Auskunft über unveränderliche Systemeigenschaften. Sie beinhaltet die Informationen, die man auch im Manual findet.

Dynamische Hilfe hingegen berücksich-

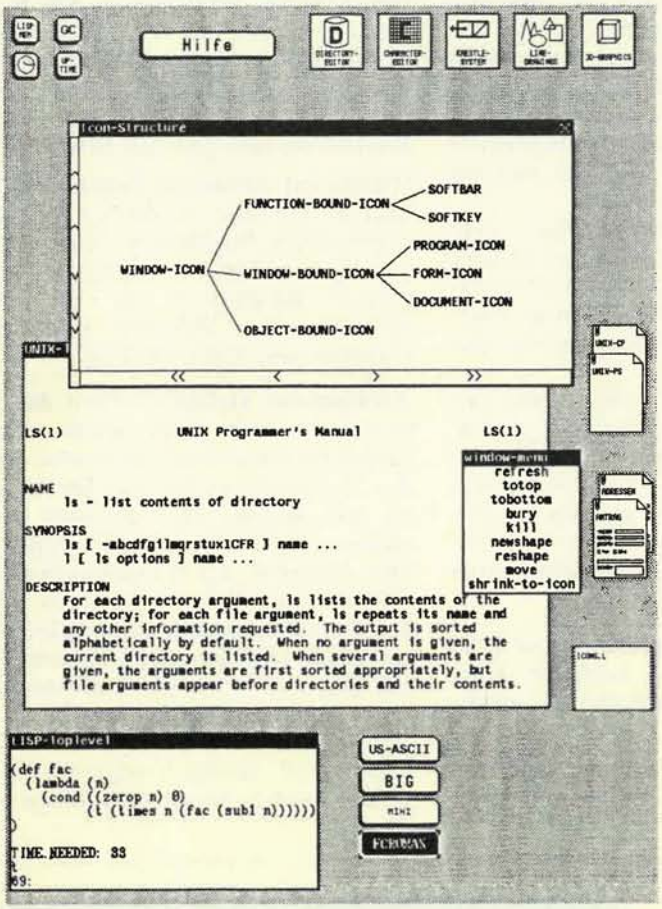


Bild 1. Verschiedene Fenster auf dem Bildschirm.

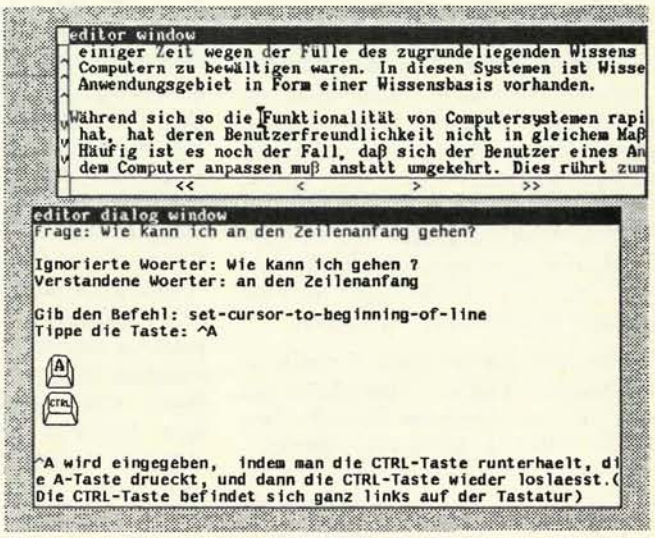


Bild 2. Das Hilfesystem PASSIVIST.

Bild 3. Das Hilfesystem AKTIVIST. ►

tigt die spezielle Umgebung zum Zeitpunkt der Hilfeanforderung, den Zustand der Interaktion [7]. Ein einfaches Beispiel für dynamische Hilfe sind Fehlermeldungen, die nicht einfach melden, daß ein Fehler auftrat (beispielsweise «falsche Eingabe!»), sondern auch, was falsch war (zum Beispiel «die Datei XY existiert nicht»).

Leider sind Fehlermeldungen meist sehr kurz und verlangen vom Benutzer bestimmte Vorkenntnisse. Sind diese nicht vorhanden, sollte der Benutzer klärende Fragen an das System stellen können. Eine gute dynamische Hilfekomponente wird bei Nachfragen des Benutzers auf die statische Hilfeinformation zugreifen. Dynamische Hilfe sollte nicht nur bei Fehlern gegeben werden, sondern auch dann, wenn vom Benutzer eine Eingabe verlangt wird und er nicht weiß, was er eingeben soll oder warum er eine bestimmte Eingabe machen soll.

Solche Hilfeleistungen erfordern vielfältiges Wissen im System. Wenn man die Erklärungen für den Benutzer aus der Wissensbasis des Systems generiert, vermeidet man Inkonsistenzen zwischen Programm und Erklärung, die sonst bei Programmänderungen gerne auftreten. Konzeptionell gegliedertes Wissen erleichtert die Suche nach einer bestimmten Information. Aus der Dialoggeschichte und dem Systemzustand läßt sich situationsspezifische Hilfe erzeugen. Ist der Ablauf von Aktionen explizit repräsentiert, lassen sich auch Erklärungen über einzelne Dialogschritte gewinnen.

Anpassungsfähige Systeme

Die wenigsten existierenden Systeme nehmen auf den unterschiedlichen Erfahrungsstand ihrer Benutzer Rücksicht. Die Folge davon ist, daß Anfänger mehr oder weniger stark überfordert werden und deshalb das System ablehnen. Anderseits stoßen Fortgeschrittene oft schon früh an die Grenzen des Systems. Die Arbeit wird dann als monoton empfunden, das System selbst als «dumm» und behindernd.

Um diese Mißstände zu mildern, benötigen wir Systeme, die an die speziellen Eigenschaften und Erfahrungen des jeweiligen Benutzers angepaßt werden können. Solche *adaptierbaren Systeme* sollen ihm insbesondere Möglichkeiten zur Gestaltung der Benutzerschnittstelle anbieten [8]. Im folgenden sind einige solcher Gestaltungsmöglichkeiten wiedergegeben:

Interaktionstechnik: Der Benutzer soll entscheiden können, wie er mit dem System kommunizieren will (zum Beispiel Menüinteraktion vs. Kommandoschnittstelle).

Informationscodierung: Es gibt sehr viele Möglichkeiten, Information auf dem Bildschirm darzustellen (beispielsweise Farbe, Zeichensätze, Größe der Darstellung). Die für einen Benutzer günstigste Art der Informationscodierung ist stark von persönlichen Präferenzen und der visuellen Aufnahmefähigkeit der betreffenden Person abhängig. Eine Einflußnahme auf die Codierungsmethode durch den Benutzer ist daher sinnvoll und wünschenswert.

Namengebung: Den Objekten und Aktionen des Systems kann nur eine mit der Anwendung vertraute Person sinnvolle Namen geben. Der Benutzer hat oft einen geeigneteren Wortschatz im Rahmen seiner Tätigkeit als der Systemdesigner, der ja die eigentliche Anwendung nicht kennt.
Systeme können so realisiert werden, daß diese Bezeichnungen frei wählbar sind. Häufig ist es zweckmäßig, für ein Objekt oder eine Aktion mehr als eine bestimmte Bezeichnung zuzulassen.

Defaults: Oft gibt es (zum Beispiel in Formularen) häufig wiederkehrende Eingabetexte, sogenannte Defaults. Der Benutzer sollte sie definieren können.
Es gibt Fälle, wo eine bestimmte Eintragung nicht gemacht werden kann, weil sie erst noch in Erfahrung gebracht werden muß. Auch in solchen Fällen ist es oft sinnvoll, einen Default einzutragen. Beispiel: Bei einer Reisebuchung nach Australien wird man als Transportmittel das Flugzeug eintragen, solange nichts anderes bekannt ist.

Definition von Aktionssequenzen: Bei der Ausführung nahezu aller Tätigkeiten kommen immer wiederkehrende Abfolgen von Aktionen vor. Diese Sequenzen sind aufgaben- und benutzerspezifisch. Sie äußern sich bei der Arbeit mit einem Computersystem meist als die ständige Wiederholung von Eingaben. Ermöglicht man dem Benutzer, solche Abfolgen selbst zu definieren, so kann ihm dadurch sehr viel stupides Eintippen erspart bleiben.
Adaptierbare Systeme vermitteln dem Benutzer ein Gefühl einer *persönlichen Arbeitsumgebung,* die er sich nach seinen Fähigkeiten und Präferenzen angepaßt hat und die er jederzeit wieder umgestalten kann.
Noch weiter gehen Systeme, die sich dem Benutzer automatisch anpassen. *Adaptive Systeme* kontrollieren die Art und Weise der Interaktion mit einem bestimmten Benutzer, um ihn bei verschiedenen Problemen besser unterstützen zu können. Einige Beispiele automatischer Anpassung:

Automatische Tippfehlerkorrektur: Das System bemerkt und korrigiert Tippfehler automatisch, sofern es eindeutig möglich ist.

Automatische Anpassung von Hilfeinformationen: Wenn ein Benutzer trotz Hilfeinformation ständig denselben Fehler bei der Verwendung eines Kommandos macht, hat er möglicherweise die Hilfeinformation nicht verstanden. Das System informiert ihn beim nächsten Mal ausführlicher über dieses Kommando und gibt ihm Beispiele.

Hinweis auf effizientere Funktionen: Sobald der Benutzer mit den für Anfänger angebotenen Funktionen sicher umgehen kann, bietet ihm das System komplexere Funktionen an, die zwar schwieriger zu bedienen sind, aber schneller zum Ziel führen [9].

Automatische Defaults: Wenn der Benutzer in einem Formularfeld ständig denselben Eingabetext verwendet, bietet das System diesen Text zur Bestätigung an und erspart dem Benutzer beim nächsten Mal das erneute Eintippen.
Das Anbieten von Gestaltungsfunktionen, die automatische Adaption sowie die Verwaltung benutzerabhängiger Unterschiede erfordern beim System Wissen über sich selbst (beispielsweise Systemzustand, Systemfunktionalität) und den Benutzer. Das Wissen über den Benutzer muß ständig korrigiert und erweitert werden. Dieses vom System ver-

waltete Wissen sollte vom Benutzer einsehbar und wenn nötig korrigierbar sein.

Tolerant bei Fehlern

Jeder Benutzer macht Fehler. Diese reichen von einfachen Tippfehlern bis zu Fehlern, die durch mangelnde Kenntnis des Anwendungsgebiets entstehen. Es gibt:

Fehler auf syntaktischer Ebene, welche typischerweise bei Benutzereingaben auftreten. Dazu gehören Tippfehler, falsche Benennung von Schlüsselwörtern und vertauschte Parameterreihenfolge usw.

Fehler durch ein falsches Modell vom System, die auftreten können, wenn der Benutzer bestimmte Konzepte des Systems nicht oder nicht richtig kennt. Ein Beispiel dafür ist die Wirkung einer Wortlöschung: wie sind dabei die Wortgrenzen definiert?

Fehler durch mangelnde Kenntnisse über die Anwendung: Während die ersten beiden Fehlerarten eine Unkenntnis von Systemeigenschaften widerspiegeln, beziehen sich diese auf den Anwendungsbereich eines Systems. Ein Beispiel dafür ist, bei einem Textformatierer einen Zeilenabstand von weniger als 1 einzustellen, obwohl die Zeilen sich dann gegenseitig überlappen.

Viele Fehler können durch einen klaren Aufbau des Systems und konsequente Verwendung gleicher Begriffe für gleiche Konzepte verringert werden. Wenn es trotzdem zu Fehlern kommt, muß der Benutzer durchschauen können, warum sie auftraten. Das System soll die Ursache von Fehlern erklären können und damit *fehlertransparent* sein.

Viele Fehler treten nur deswegen auf, weil das System starr ist und nur eine genau festgelegte Eingabeform akzeptiert [10]. Einige dieser Fehler kann das System selbst korrigieren. Es soll *fehlertolerant* sein.

Wenn Fehler nicht automatisch korrigiert werden, sollte das ganze System durch eine flexible Fehlerbehandlung dafür sorgen, daß der Benutzer gezielt nur seinen Fehler verbessern und nicht die ganze Eingabefolge wiederholen muß. Dieses gezielte Korrekturangebot macht den Fehler außerdem transparenter.

Oft ist «Fehler» nicht die richtige Bezeichnung für eine Aktion, die zu einem unerwarteten oder unerwünschten Systemzustand führt. Es gibt Situationen, in denen ein Benutzer erkunden will, was eine bestimmte Funktion in einem bestimmten Kontext bewirkt. Der Benutzer hat bei derartigen Erkundungen kein bestimmtes Arbeitsziel im Sinn, sondern will einfach die Wirkung der ausgelösten Aktion sehen.

Eine andere Situation, bei der Resultate (Systemzustände) entstehen, die nicht erwünscht sind, ist die Ausübung von Planungs- und Designprozessen. Hier ist es normal, daß immer wieder Resultate herbeigeführt werden, die nicht der Absicht des Benutzers entsprechen. Durch das Verwerfen solcher erreichter, nicht erwünschter Zustände wird der Planungs- oder Designprozeß geradezu angetrieben. Auch hier wäre es unangebracht, von Fehlern zu sprechen.

Gerade die beiden letztgenannten Fälle wecken den Wunsch des Benutzers, die Wirkung einer Aktion mit einem sogenannten *UNDO-Mechanismus* wieder beseitigen zu können.

Es gibt eine ganze Reihe von Realisierungs- und Interaktionsmodellen für UNDO, die sehr anwendungsspezifisch sind. Was höherentwickelte UNDO-Techniken jedoch gemeinsam haben, ist die Notwendigkeit, das System mit Wissen über den Dialogablauf zu versorgen, den Dialog also in irgendeiner Form zu protokollieren, sowie das System mit Methoden zur Neutralisierung von Funktionsausführungen zu versorgen. Dieses Wissen sollte dem Benutzer zugänglich sein, damit er den gewünschten Zustand spezifizieren kann, zu dem er zurückkehren will. Mit solchen UNDO-Mechanismen lassen sich auch typische Bedienungsfehler des Benutzers unwirksam machen.

Bei wissensbasierten Systemen sind Aktionen nicht in starren Schemata programmiert, sondern als Erfordernisse spezifiziert, die der Benutzer nach und nach erfüllt. Wissen über die Wirkung und den Ablauf von Aktionen verwendet man dazu, um Fehler zu erklären, zu korrigieren oder zu neutralisieren. Ungewünschte Systemzustände können durch UNDO-Mechanismen auf vorhergehende Systemzustände zurückgeführt werden. Wissensbasierte Systeme erleichtern nicht nur die Fehlerbehandlung, sondern verhindern Fehler durch konsistenten Aufbau, Anpassung an den Benutzer und Hilfeleistungen.

Aus den bisherigen Ausführungen geht hervor, daß ein System, das benutzergerecht sein soll, vielfältiges Wissen benötigt. Sein Wissen wird sowohl zur Beantwortung von Benutzerfragen als auch zur Steuerung des Systems verwendet. Im folgenden fassen wir das für die angesprochenen wünschenswerten Systemeigenschaften benötigte Wissen noch einmal zusammen:

Wissen über Interaktionsformen und Interaktionsschritte

Wissen über den Systemzustand und die Dialoggeschichte

Wissen über die Funktionalität

Wissen über den Benutzer

Wissen über den Anwendungsbereich und die Arbeitsumgebung

Bei der Realisierung von Systemen mit den beschriebenen Eigenschaften zeigt es sich immer wieder, daß herkömmliche Programmiertechniken dazu nicht besonders geeignet sind. Einen gangbaren Weg stellt die Verwendung von sogenannten Wissensrepräsentationssprachen dar. In unserer Forschungsarbeit verfolgen wir unter anderem die objektorientierte Realisierung solcher Systeme. Die dazu entwickelte und sich in Weiterentwicklung befindliche Sprache ist *ObjTalk* [11]. 13 TR

Literatur

1 Dzida W.: Das IFIP-Modell für Benutzerschnittstellen. Office Management 31 (Sonderheft), 1983.

2 Hayes P. J., Szekely P. A., Lerner R. A.: Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. In: Borman L., Curtis B. (eds.): Human Factors in Computing Systems. CHI-85 Conference Proceedings, New York, 1985.

3 Smith D. C., Irby Ch., Kimball R., Verplank B.: Designing the Star User Interface, BYTE, 1982.

4 Shneiderman B.: Direct Manipulation: A Step Beyond Programming Languages. IEEE Computer 16 (8), 1983.

5 Lemke A.: PASSIVIST: Ein passives, natürlichsprachliches Hilfesystem für den bildschirmorientierten Editor BISY. Diplomarbeit Nr. 293, Institut für Informatik, Universität Stuttgart, 1984.

6 Schwab T.: AKTIVIST: Ein aktives Hilfesystem für den bildschirmorientierten Editor BISY. Diplomarbeit Nr. 232, Institut für Informatik, Universität Stuttgart, 1984.

7 Hayes P. J.: Uniform Help Facilities for a Cooperative User Interface. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, 1982.

8 Herczeg M.: DYNAFORM – Ein interaktives Formularsytsem zum Aufbau und zur Bearbeitung von Datenbasen. In Balzert H. (ed.): Software-Ergonomie, German Chapter of the ACM, Stuttgart, 1983.

9 Bauer J., Schwab T.: Aktive Hilfesysteme. In: Notizen zu interaktiven Systemen, Fachgruppe interaktive Systeme, Darmstadt, 1985.

10 Norman D. A.: The Trouble with UNIX. Datamation, 1981.

11 Rathke C.: ObjTalk Primer. Institutsbericht, Institut für Informatik, Universität Stuttgart, 1984.

12 Herczeg M., Maier D., Rathke C., Riekert W.-F.: Vom Dialogsystem zur wissensbasierten Mensch-Computer-Kommunikation. In On-line '85, 8. Europäische Kongreßmesse für Technische Kommunikation, Düsseldorf, 1985.

Quelle: Mit freundlicher Genehmigung des Verlages B. G. Teubner, Stuttgart; übernommen aus «Berichte des German Chapter of the ACM, Bd. 24: Sofware-Ergonomie 1985: Mensch-Computer-Interaktion». Hrsg. H.-J. Bullinger, 1985.

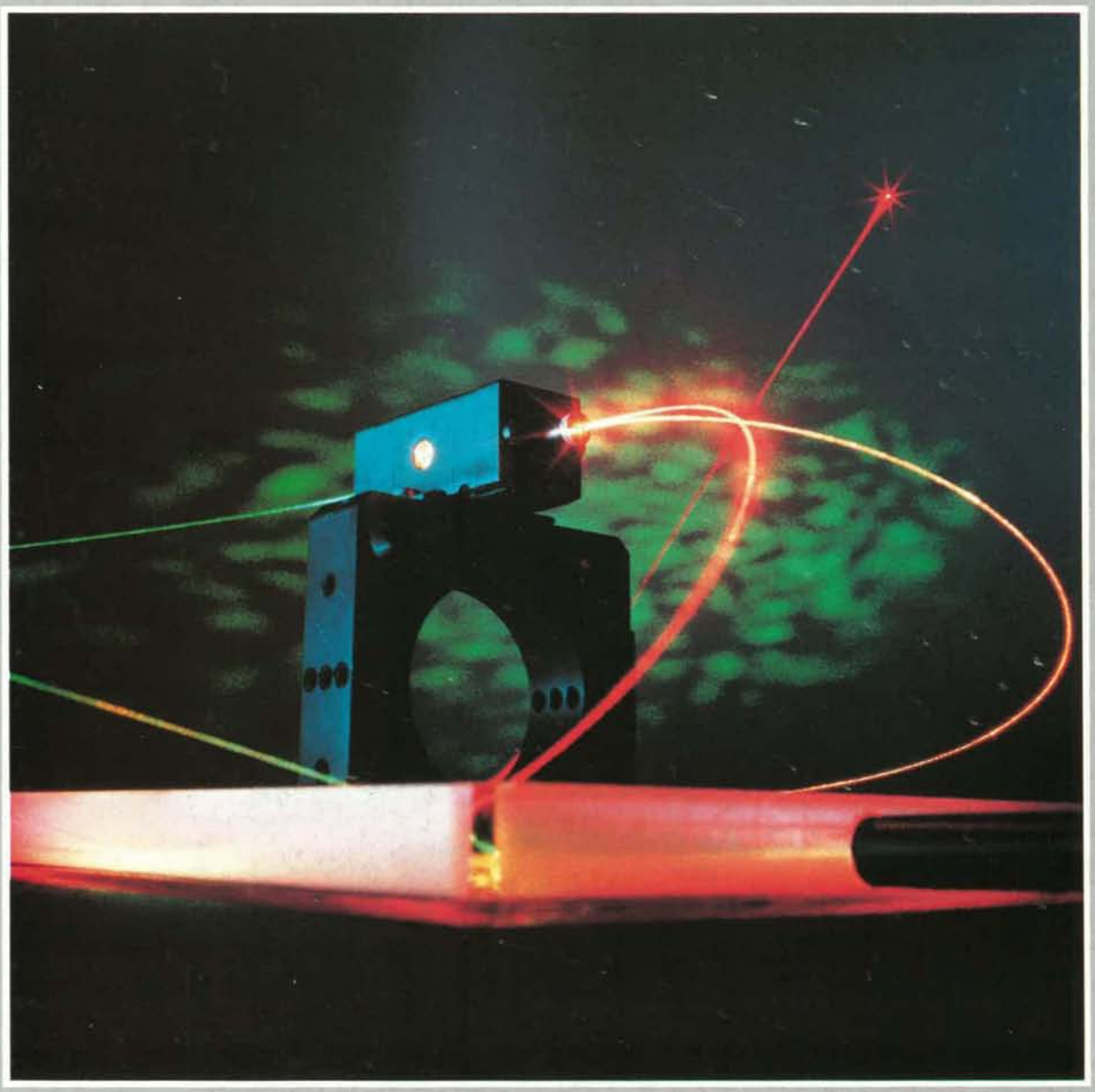
Hannover Messe 86 Nachschau

TECHNISCHE RUNDSCHAU

Wochenmagazin für innovative Technik, Indust

10 Jahre SAP
Seite 102

TR 23 3. Juni 1986 78. Jahrgang Fr. 3.– DM 4.– öS 30.–



Perspektive: Bedrohungsentwicklung

Vibrationsschweißen von Thermoplasten

Instandhaltung im Wandel

Glasfasern: Von der Substitution zur Innovation

Softwareergonomie

Vollkostenrechnung