Zeitschrift: Vermessung, Photogrammetrie, Kulturtechnik : VPK = Mensuration, photogrammétrie, génie rural

Herausgeber: Schweizerischer Verein für Vermessung und Kulturtechnik (SVVK) = Société suisse des mensurations et améliorations foncières (SSMAF)

Band: 83 (1985)

Heft: 2

Artikel: Zur Entwicklung interaktiver Programme und Systeme

Autor: Kuhn, W.

DOI: https://doi.org/10.5169/seals-232582

Zur Entwicklung interaktiver Programme und Systeme

W. Kuhn

In den vergangenen zwei bis drei Jahren sind verschiedene Modelle eines neuen Typs von Arbeitsplatz-Computern auf den Markt gekommen (Xerox Star, ICL Perq, Apple Macintosh), die eine radikal neue Benützer-Schnittstelle anbieten. Als Eingabemedium verwenden sie neben der Tastatur eine ‹Maus›, d. h. ein handliches Gerät, mit dem der Cursor rasch zu jedem beliebigen Punkt des Bildschirms geführt werden kann. Die Ausgabe erfolgt auf einen ‹Bit-Map› Bildschirm, d. h. eine Anzeige, die aus mehreren 100 000 einzeln ein- und ausschaltbaren Bildpunkten (Pixeln) besteht. Die Information ist in Fenstern gruppiert, die sich meistens überlagern können. Arbeitsschritte, soweit sie nicht durch ‹direct manipulation› (siehe Abschnitt 5) ausgeführt werden, sind in Menus zusammengefasst. Diese erscheinen durch Antippen mit der Maus entweder am oberen Bildschirm-Rand (‹pull-down› Menus) oder dort auf dem Bildschirm, wo gerade gearbeitet wird (‹pop-up› Menus).

Au cours des deux ou trois dernières années est apparu sur le marché un nouveau type d'ordinateur monoposte (Xerox Star, ICL Perq, Apple Macintosh), qui propose un interface-utilisateur radicalement nouveau. A part l'emploi d'un clavier comme moyen de communication, l'utilisateur dispose d'une souris, c.-à-d. un petit appareil maniable, qui permet de placer rapidement un curseur sur n'importe quels points de l'écran de visualisation. Cet affichage ou écran ‹Bit-Map› se compose de quelques 100 000 points, qui peuvent être enclenchés ou déclenchés individuellement. Les informations sont groupées dans des fenêtres, qui peuvent se superposer. Les différentes possibilités de travail sont contenues dans des menus. Au moyen de la souris, ceux-ci apparaissent soit au bord supérieur de l'écran (menus ‹pull-down›), soit à l'endroit, où l'on travaille actuellement (menus ‹pop-up›).

1. Einleitung

Am Institut für Geodäsie und Photogrammetrie sind in den vergangenen Jahren im Lehrbereich von Prof. Conzett einige grössere Programmpakete entwickelt worden. Drei davon dienen hauptsächlich den im Unterricht und den Diplomkursen anfallenden Entwurfs- und Berechnungs-Aufgaben: INTRA für den Entwurf und die Ausgleichung von Triangulations-Netzen, DATAUF für die Datenerfassung und -aufbereitung in Triangulationen, PRIMA für Matrizenoperationen.

Alle diese Programme haben eine wichtige Eigenschaft gemeinsam: Sie sind *interaktiv,* d. h. der Benützer arbeitet am Terminal, sieht das Resultat einzelner Arbeitsschritte und kann den Ablauf entsprechend steuern.

Die Qualität eines Programms misst sich heute vor allem daran, ob es einfach zu bedienen ist. Interaktive Programme, die richtig rechnen, sind nicht zwingend auch ‹gute› Programme. Die Frage, wie Programme möglichst *benützerfreundlich* zu gestalten sind, wird gegenwärtig von Informatikern und Psychologen intensiv erforscht (siehe z. B. Janda 1983). Ein Projekt an unserem Institut untersucht dieses Problem im Zusammenhang mit Landinformationssystemen.

Institut für Geodäsie und Photogrammetrie
ETH-Hönggerberg, CH-8093 Zürich
Separata Nr. 83

Die allgemeinsten Forderungen an interaktive Programme oder Systeme sind etwa, dass sie
- einfach zu bedienen
- einfach zu erlernen
- einfach zu erinnern

sein sollen. Je nach den erwarteten Benützerklassen (Anfänger, gelegentliche oder erfahrene Benützer) erhalten diese Anforderungen unterschiedliche Gewichte. Im Zweifelsfalle wird empfohlen, sich am unerfahrenen Benützer zu orientieren.

Es geht hier nicht darum, neue Theorien über Interaktivität zu erarbeiten oder Normen für die Programmierung aufzustellen. Vielmehr soll anhand unserer Erfahrungen[1] bei der Entwicklung und im intensiven Einsatz der erwähnten Programme (je über 100 Benützer) auf kritische Punkte in den folgenden Phasen der Entstehung interaktiver Programme aufmerksam gemacht werden: Dem Entscheid, was ein Programm tun soll, dem Vorgehen beim Entwurf, der Dialog- und Bildschirm-Gestaltung, der Beurteilung des Benützer-Verhaltens und der Programm-Dokumentation.

[1] Dieser Artikel beruht auf Erfahrungen und Erkenntnissen aus mehrjähriger Arbeit, an denen der Autor nur einen kleinen Anteil hatte. Für unzählige Anregungen und Hinweise danke ich Prof. Conzett, allen ehemaligen und heutigen Kollegen und den Studenten, von derern Erfolgen und ‹Fehlern› (siehe 10. Postulat) wir viel gelernt haben.

Zuerst werden zwölf allgemeine *Postulate* zur Interaktivität formuliert (Abschnitt 2). Dann folgen in Abschnitt 3 einige Bermerkungen zum *Vorgehen* beim Programm-Entwurf. Abschnitt 4 soll zeigen, dass für den Dialog-Entwurf und dessen Beurteilung nicht nur Tips und Tricks, sondern ingenieurmässige *Methoden* zur Verfügung stehen. Ein Hinweis auf die veränderte Rolle der *Graphik* (Abschnitt 5) leitet zu einem *Ausblick* auf die zu erwartenden Entwicklungen (Abschnitt 6) über.

Wir glauben, dass diese Betrachtungen, die keinen Anspruch auf Vollständigkeit erheben, auch für den Praktiker von Interesse sind. Sei es, weil er selbst mit der Entwicklung von Programmen zu tun hat, sei es als Anhaltspunkte, um Software auf dem Markt zu bewerten, oder als allgemeine Orientierungshilfe in der EDV-Diskussion.

2. Postulate zur Interaktivität

2.1 Postulat: Der Computer ist ein Instrument und hat sich dem Menschen anzupassen, nicht umgekehrt.

Herkömmliche Maschinen bewältigen eine bestimmte Aufgabe auf eine meist unveränderliche Weise und verlangen oft eine beträchtliche Anpassung vom Menschen. Im Gegensatz dazu erlaubt uns der Computer, insbesondere im interaktiven Einsatz, durch ein Programm die Lösung von Problemen den verschiedenen Anwendern anzupassen. Diese Chance wird bis heute viel zu wenig wahrgenommen. Schlagworte wie ‹computergerecht›, ‹Umstellung auf EDV› usw. werden häufig dazu missbraucht, dem Menschen Arbeitsabläufe aufzuzwingen, die ihm unvertraut sind und die sich von der zu lösenden Aufgabe her nicht begründen lassen. Der Computer verändert unsere Arbeitsweise. Wir haben es in der Hand, dafür zu sorgen, dass dies zu menschen- und problemgerechten Lösungen führt. Ein Beispiel dafür liefert das

2.2 Postulat: Manuelle Arbeitsabläufe zu programmieren bedeutet oft Zeit- und Geldverschwendung. Der Computer eröffnet neue Wege zur Problemlösung.

Programme sollen nicht einfach Berechnungs-Formulare automatisieren. Formulare sind Hilfsmittel für Lösungen von Hand oder mit dem Taschenrechner. Sie organisieren den Arbeitsablauf und die notwendigen Rechenkontrollen. Die Eintragungen sind redundant, d. h. die gleiche Information wird mehrfach aufgeschrieben, z. B. neben dem Wert

eines Winkels auch sein Sinus und sein Cosinus. Der Berechnungsvorgang lässt die zugrundeliegende Lösungsidee oft kaum erkennen.
Der Schritt vom Taschenrechner zum Computer ist wesentlich grösser als jener von der Logarithmentafel zum Taschenrechner. Er bedeutet den Übergang vom automatischen Rechnen, wo der Anwender weiterhin die Last der ‹Buchhaltung› trägt, zur Datenverarbeitung, wo ihm diese uninteressante Arbeit vom Computer abgenommen wird, der sie überdies viel zuverlässiger erledigt.
Bei einer sinnvollen Lösung mit dem Computer ist das Programm für die Organisation und die Hardware für die Rechenkontrollen verantwortlich. Somit kann sich der Benützer auf die Fragen konzentrieren, die sein Problem betreffen. Trotzdem soll es noch Programme geben, denen man zusätzlich zu einem Winkel auch noch dessen Sinus und Cosinus füttern muss ...
Nebenbei bemerkt: Reine Rechenkontrollen zu programmieren ist sinnlos. Heutige Computer machen etwa einen Fehler in 10^{10} Operationen. Ist ein Programm korrekt geschrieben, so braucht es keine Rechenkontrollen auszuführen, andernfalls nützen auch diese nichts. Sie können allenfalls zur Fehlersuche während der Programmierung dienen.

2.3 Postulat: Kein Programm kann alles. Wenn es 90 Prozent der Fälle korrekt erledigt, kann es in 10 Prozent der Zeit erstellt werden, die für ein 99prozentiges Programm nötig wäre. Vor allem ist es dann aber viel einfacher zu bedienen.
Die Benützerfreundlichkeit vieler Programme leidet am Ehrgeiz des Programmierers, für jeden noch so seltenen Spezialfall eine Lösung anbieten zu wollen. Die Befehle werden komplizierter und ihre Anzahl unüberblickbar. Die Entwicklungszeit kann derart wachsen, dass das Programm schon vor seinem Einsatz veraltet ist. Auch der Programmunterhalt wird erschwert.
Bei notwendigen Lösungen für Sonderfälle gilt die Regel ‹Einfaches einfach, Komplizierteres möglich›. Dies bedeutet, dass eine einfache Lösung für den Normalfall nicht durch die Möglichkeit zur Behandlung von seltenen komplizierteren Fällen erschwert werden soll. Das Meistern der Komplexität einer Aufgabe, ihre gedankliche Durchdringung, ist die erste Voraussetzung für jedes erfolgreiche Programm. Wenn sich Spezialfälle häufen, liegt das meist an der gewählten Lösung, nicht am Problem. Ein Überdenken des Lösungsweges führt dann oft zu unerwarteten Vereinfachungen und zum Verschwinden von Spezialfällen.

2.4 Postulat: Der Erfolg von Programmsystemen hängt davon ab, wie einheitlich die einzelnen Programme mit dem Benützer sprechen.
Es gibt für den Benützer nichts Verwirrenderes, als wenn die gleiche Aktion in verschiedenen Programmen unterschiedliche (leider oft entgegengesetzte) Reaktionen bewirkt. Die Konsistenz der Benützer-Schnittstellen mehrerer Programme in sich und untereinander hat einen entscheidenden Einfluss auf das Verständnis und die Lernbarkeit. Sie erlaubt dem Benützer Verallgemeinerungen, mit denen er den Aufbau von Operationen teilweise erraten, hauptsächlich aber leichter erlernen und erinnern kann.
Ein einfaches, konsequent durchgehaltenes Muster für Befehle ist deshalb sehr wichtig. Routinen für alle Ein- und Ausgabefunktionen sollen *einmal* erstellt und dann in allen Programmen als Bauelemente verwendet werden. Beim Ausbau bestehender Programmsysteme muss hier oft Einheitlichkeit gegen punktuelle Verbesserungen abgewogen werden.
Moderne Arbeitsplatz-Computer (siehe Kasten) bieten Bausteine für eine einheitliche Schnittstelle als Bestandteil des Betriebssystems an. Ohne diese Unterstützung ist Konsistenz in grösseren Anwendungen kaum zu erreichen.

2.5 Postulat: Werden gleiche Eingaben an verschiedenen Orten unterschiedlich interpretiert, soll der Benützer immer sehen, welche Interpretation (welcher Modus) gerade gilt.
Die Vielzahl möglicher Befehle in einem interaktiven System macht es meist unumgänglich, dass gewisse Tätigkeiten je nach Systemzustand unterschiedliche Auswirkungen haben, etwa das Drücken der RETURN-Taste oder eines Maus-Knopfs. Dies bedeutet keinen Widerspruch zum 4. Postulat, wenn der gültige Modus für den Benützer offensichtlich ist und wenn irrtümliche Eingaben einfach korrigiert werden können.
Auch alphanumerische Terminals bieten viele Möglichkeiten, Modi verständlicher darzustellen als nur durch eine Textzeile, z. B. indem die jeweils aktiven Befehle und Daten hervorgehoben werden (‹highlighting›).

2.6 Postulat: Der Benützer will sehen, nicht lesen oder sich erinnern müssen.
Einerseits ist die Informations-Menge, die der Anwender zwischen zwei aufeinanderfolgenden Handlungen erfassen kann, gering. Jedes Zeichen, das keine Information liefert (d. h. Unsicherheit vermindert), stört die Nachricht der anderen Zeichen. Floskeln und Dekorationen verschwenden die Zeit des Programmierers, des Anwenders und des Computers. Sätze erfasst man selten in einem Blick. Deshalb sind in vielen Fällen Stichwörter mit Symbolen (Pfeilen usw.) klarer als ausformulierte Sätze. Sie müssen aber einheitlich verwendet und dargestellt sein. Andererseits ist das Kurzzeit-Gedächtnis des Benützers beschränkt und wird durch die Lösung seiner Aufgabe beansprucht. Alle für den nächsten Arbeitsschritt benötigte Information soll deshalb auf dem Bildschirm vorhanden sein. Diese Forderung führt meist zur Darstellung der Befehle in Form von Menus, die sich den jeweils tatsächlich angebotenen Befehlen anpassen. Die Daten werden dem Benützer so präsentiert, dass er sie direkt bearbeiten und ihre Veränderung auf dem Bildschirm verfolgen kann.

2.7 Postulat: Der Benützer soll auf einen Blick erkennen können, was das Programm von ihm erwartet.
Ein Programm kann im wesentlichen auf drei verschiedene Arten zum Benützer sprechen: Es kann von ihm Eingaben verlangen (Antwort auf eine Frage, Wahl aus einem Menu, Ausfüllen einer Maske), ihn zu sonstigen Handlungen auffordern (Anschliessen eines Peripheriegeräts, Wechseln einer Diskette), oder es kann eine Meldung ausgeben (Fehler, Systemzustand). Ausgaben auf den Bildschirm sollen auf wenige Muster beschränkt und so gekennzeichnet werden, dass sofort klar wird, um welche Dialog-Form es sich handelt. Eine graphische Unterscheidung ist natürlich auch hier dem blossen Anschreiben vorzuziehen.
Systeme, die statt einer zeilenweise über den Bildschirm laufenden Ausgabe mehrere Fenster verwenden, erleichtern diese Typisierung sehr.

2.8 Postulat: Ein benützerfreundliches Programm lässt keine Ungewissheit aufkommen.
Die Geduld und das Selbstvertrauen des Anwenders im Umgang mit dem Computer sind beschränkt. Er erwartet auf einfache Befehle eine sofortige Reaktion, auch wenn das Resultat mehr Zeit benötigt. In Fällen, die er als schwieriger empfindet, sind etwas längere Antwortzeiten zulässig. Es darf aber nie Unsicherheit darüber aufkommen, ob das Programm eine Lösung sucht, sich in einer endlosen Schlaufe bewegt oder blockiert ist. Im Unterschied zum Menschen lässt der Computer nicht erkennen, ob er ‹nachdenkt› oder ‹nicht mehr weiter weiss›. Deshalb muss diese Information vom Programm geliefert werden. Eindeutige Symbole, etwa ein Cursor, der blinkt oder die Form ändert (Sanduhr, Biene, Buddha), um anzudeuten, dass ein Prozess abläuft, sind Meldungen (‹Bitte warten›) vorzuziehen. Wichtig für den Arbeits-

Interaktion mit modernen Arbeitsplatz-Computern

Einige wenige Entwurfsprinzipien sorgen für einfache und einheitliche Interfaces in allen Anwendungen. Diese Grundsätze lassen sich wie folgt zusammenfassen [Smith et al. 1982]:

- Das Modell, das der Benützer vom System erhält, verwendet *vertraute Konzepte:* Weit verbreitet ist die Schreibtisch-Metapher, mit der auf dem Bildschirm die gewohnte Arbeitsumgebung simuliert wird: Dokumente, Ordner, Briefkasten, Papierkorb usw. Alle diese Gegenstände erhalten eindeutige Symbole (‹icons›), und der Benützer kann damit die gewohnten Operationen ausführen, etwa einen Ordner öffnen oder ein Dokument wegwerfen.
Will er den Inhalt eines Dokuments anschauen oder verändern, so eröffnet das System dafür ein Fenster auf dem Bildschirm. Dieses zeigt einen Ausschnitt des Inhalts und kann an beliebige Stellen im Dokument bewegt werden. Fenster können sich überlagern, wie übereinanderliegende Papiere auf dem Schreibtisch. Das System lässt klar erkennen, welches Fenster ‹aktiv› ist, d. h. von den Operationen berührt wird.
- Alle benötigte Information, oder ein Zugang zu ihr, ist jederzeit *sichtbar.* Der Benützer muss sich keine Befehle merken und kann somit sein Gedächtnis voll für seine Aufgabe einsetzen.
Jede Operation führt sofort zu einem sichtbaren Resultat, das, falls nötig, rückgängig gemacht werden kann. Durch diese Transparenz und Korrekturmöglichkeit verliert der Benützer weitgehend die Angst vor Fehlmanipulationen.
- Die Arbeitsweise ist in allen Anwendungen *einheitlich.* Gleiche Handlungen haben an verschiedenen Orten sinngemässe Auswirkungen. Alle Operationen folgen dem gleichen Grundmuster, z. B. dem Objekt-Verb Paradigma: Zuerst wird das Objekt gewählt (durch Antippen mit der Maus) und dann die Operation damit ausgeführt (z. B. wird das Objekt mit der Maus über den Bildschirm ‹gezogen›, oder ein Befehl wird aus einem Menu gewählt).
- Die Bedienung ist *einfach,* auch in komplexen Anwendungen. Dies hängt eng zusammen mit der Einheitlichkeit. Ideal ist ein Minimum an wirksamen Operationen für jede Anwendung. Mehrere Möglichkeiten, das selbe zu erreichen, können verwirren. Trotzdem sollten dem erfahrenen Benützer Abkürzungen angeboten werden. Einfachheit lässt sich immer nur unter Nebenbedingungen maximieren.
- *‹What you see is what you get›:* Dokumente sehen auf dem Bildschirm genau so aus, wie sie nachher z. B. ausgedruckt werden. Der Text wird nicht durch Kontroll-Zeichen gestört, und das Resultat von Formatierungen, Schriftwechseln usw. ist sofort sichtbar.

Die Abbildungen 1 und 2 sollen anhand von ‹Momentaufnahmen› einer denkbaren Benützer-Schnittstelle für ein Triangulations-Programm einige der erwähnten Techniken moderner Interaktion illustrieren. Die Absicht ist, die gegenüber herkömmlichen Systemen völlig veränderte Arbeitsweise zu zeigen, deren Hauptmerkmal ist, dass alle relevanten Vorgänge sichtbar sind. Es geht also nicht darum, einen Entwurf für ein bestimmtes Programm darzulegen.

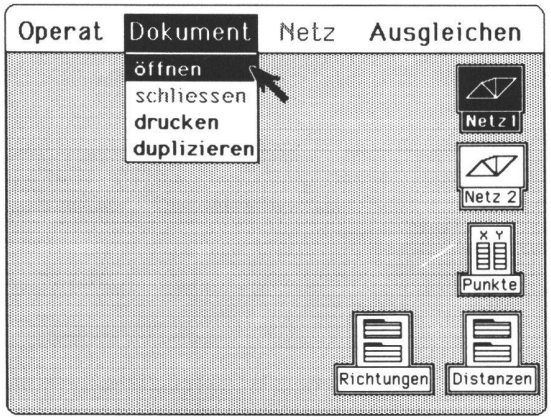
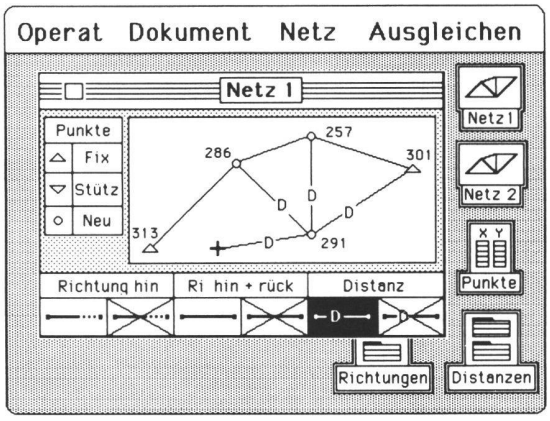


Abb. 1 Wir haben das Triangulations-Programm gestartet und ein Operat ausgewählt. Das System zeigt am rechten Bildschirmrand ein icon für jedes Dokument in diesem Operat. Wir können eines dieser Dokumente auswählen, um damit zu arbeiten, z. B. die Variante 1 des Netzplans. Dazu schieben wir den Cursor (geführt durch die Maus) in das betreffende icon (‹Netz 1›) und ‹klicken› dort mit der Maustaste. Dadurch wird dieses Dokument ‹aktiv›, was uns durch ein hervorgehobenes icon angezeigt wird. Alle folgenden Operationen beziehen sich also auf den Inhalt von ‹Netz 1›. Am oberen Bildschirm-Rand stehen in einer Zeile die Titel aller ‹pull-down›-Menus unserer Anwendung. Auf ähnliche Art, wie wir ‹Netz 1› auswählten, verlangen wir nun einen Arbeitsschritt im Menu ‹Dokument›: Wir führen den Cursor (Pfeil) zum Menu-Titel und drücken dort die Maustaste, worauf der Menu-Inhalt angezeigt wird. Nun wählen wir jenes Feld, das die gewünschte Operation enthält. Sobald wir die Taste dort loslassen, verschwindet das Menu, und diese Operation wird ausgeführt. Wenn wir den Netzplan anschauen wollen, wählen wir ‹öffnen›. Das Vorgehen entspricht dem ‹Objekt-Verb-Paradigma›: Zuerst wählen wir den Gegenstand (Netz 1) und dann die Operation (öffnen).
Das Menu bietet uns auch einen Ausdruck auf Papier oder eine zweite Speicherung der Netzplandaten an. Im jetzigen Zustand hat die Operation ‹schliessen› keine Bedeutung, da noch kein Dokument geöffnet ist. Das gleiche gilt, da noch kein Netz dargestellt ist, für das ganze ‹Netz›-Menu (das Operationen auf dem Netz-Graph anbietet, z. B. die Anzeige aller Distanzen oder der Punkte allein). Deshalb sind der Arbeitsschritt ‹schliessen› und der Menu-Titel ‹Netz› nur schwach angezeigt; sie können nicht ausgewählt werden.

Abb. 2 Als Resultat erscheint auf dem Bildschirm ein Fenster, das den Netzplan enthält und Editier-Funktionen in Form neuer icons anbietet. Wir können mit dem beschriebenen Verfahren z. B. das Symbol für einen Fixpunkt wählen und es direkt manipulieren: Mit heruntergedrückter Maustaste ‹ziehen› wir das Dreieck bis zu einem Punkt, den wir in der Ausgleichung festhalten wollen und lassen dort die Taste und damit das Symbol los. So sind die Punkte 301 und 313 als Fixpunkte gekennzeichnet worden. In der Abbildung wurde das Distanz-icon aktiviert. Wir schieben den Cursor, der nun die Form eines Strichkreuzes hat, zum Anfangspunkt der einzuführenden Distanz-Beobachtung und drücken dort die Maustaste. Die Abbildung zeigt den Zustand unterwegs zum Endpunkt, wo durch Loslassen der Taste die Distanz in den Netzplan eingefügt wird.

Um einzelne Beobachtungen aus dem Netzplan zu entfernen, stehen die durchgestrichenen icons zur Verfügung. Das ‹Netz›-Menu bietet zudem ständig einen UNDO-Befehl für die letzte Operation an. So müssen wir nie befürchten, den Netzplan durch eine irrtümliche Veränderung ‹verdorben› zu haben.
Zu beachten sind die im Hintergrund weiterhin angebotenen icons. Wir können damit jederzeit die Arbeit am Netzplan unterbrechen und in einem anderen Fenster (das je nach Bildschirm-Grösse den Netzplan überlagern wird) etwa die Liste der Richtungs-Beobachtungen oder der Koordinaten anschauen und verändern. Andererseits bietet uns das Menu ‹Ausgleichen› die Berechnung und Auflösung der Normalgleichungen für das gerade aktive Netz an. Die Wahl dieser Schritte würde neue Resultat-icons (z. B. für den Lösungsvektor und für die Qxx-Matrix) hervorbringen. Um ein Dokument, z. B. den Netzplan, zu schliessen, wählen wir entweder im ‹Dokument›-Menu den entsprechenden Schritt (er wirkt auf das zur Zeit aktive Fenster) oder klicken in der kleinen ‹Schliessbox› in der linken oberen Ecke des betreffenden Fensters.



rhythmus und für das Gefühl des Anwenders, die Lage zu beherrschen, ist eine durchdachte Unterteilung der Probleme in kurze, reversible Arbeitsschritte, mit denen der Ablauf überwacht und gesteuert werden kann.

2.9 Postulat: Programme, die dem Benützer Befehle erteilen, werden nur widerwillig benützt, verfehlen somit ihren Zweck und fördern eine Abwehrhaltung gegenüber dem Computer.
Die Gewohnheit des Programmierers, dem Computer Befehle zu erteilen, schlägt leider oft bis in den Dialog durch. Der Benützer soll aber davon überzeugt sein, dass *er* den Arbeitsablauf bestimmt und ihn das Programm dabei nur unterstützt.
Das bedeutet nicht, dass Aufforderungen des Programms mit umständlichen Höflichkeitsfloskeln (‹Würden Sie bitte . . .›) vernebelt werden sollen. Weder ein herablassendes noch ein unterwürfiges Programm, sondern eines, von dem der Benützer jederzeit etwas nicht Plangemässes (Unterbruch, Information über Systemzustand usw.) verlangen kann, gibt ihm das Gefühl, die Lage zu beherrschen.

2.10 Postulat: Sogenannte Benützer-Fehler gibt es kaum. Handlungen, die zu einem unerlaubten System-Zustand (‹Absturz›) führen, haben ihre Ursache meist in missachteten Interaktions-Prinzipien.
Zu Unrecht schiebt man Fehler der Anwender gerne auf deren Unerfahrenheit ab. Ein Programm, das nur von erfahrenen Benützern eingesetzt werden kann, ist ein schlechtes Programm. Entscheidend ist, dass der Benützer keine Angst vor Fehlern haben muss. Ein System, in dem keine Fehler möglich sind, ist aber unerreichbar. Das Ziel ist deshalb eine Minimierung der Möglichkeiten, Fehler zu begehen, und der Auswirkungen davon. Ideal, aber oft schwierig zu programmieren ist ein UNDO-Befehl, um mindestens die letzte Operation rückgängig zu machen.
Wenn das System dem Benützer ein klar erkennbares, einfaches und widerspruchfreies Modell von seiner Aufgabe vermittelt, so werden Fehl-Eingaben sehr selten und können vom Programm abgefangen werden. Bei einer Untersuchung am Programmpaket PRIMA konnten *alle* mehrfach beobachteten Fehlmanipulationen auf Schwächen des Entwurfs zurückgeführt werden.

2.11 Postulat: Fehlermeldungen sollen dem Benützer in seiner Sprache mitteilen, was er falsch gemacht hat und wie er es korrigieren kann.
Die vielzitierten Beispiele der Art ‹Err. 92745 – Acc. Viol.› brauchen leider nicht erfunden zu werden. *Dass* etwas schiefläuft, merkt der Benützer meist auch sonst; darüber, *was* schiefging und wie er Abhilfe schaffen kann, sagt ihm eine solche Meldung aber nichts. Wenn er diese Information in einem Handbuch (in welchem?) nachschlagen muss (wo?), wird er keine Freude am Programm haben, besonders wenn er auch dort keine verständliche Auskunft findet.
Gute Fehler-Meldungen sind kurz und konstruktiv. Sie erscheinen so schnell wie möglich nach dem fehlerhaften Ereignis, um wirkungslose Eingaben oder grösseren Schaden zu verhüten und um Korrekturen zu erleichtern.

2.12 Postulat: Dokumentation kann schlechte Programme nicht verbessern. Sie sollte nur in drei Fällen benötigt werden: Zur Einführung für den Anfänger, als Hilfe in schwierigen Fällen und als Grundlage für den Programm-Unterhalt.
Häufige Benützer-Fehler durch vermehrte Dokumentation beheben zu wollen ist meist erfolglos oder sogar kontraproduktiv.
Bedienungs-Anleitungen werden selten gelesen und noch seltener verstanden. Ein gutes Programm, eventuell mit einer HELP-Funktion, macht sie fast überflüssig.
Programm-Beschreibungen für den Programmierer werden selten nachgeführt. Ein klar geschriebener, kommentierter Programm-Code ist viel nützlicher.
Bei on-line-Dokumentation mittels einer HELP-Funktion bewährt sich ein zweistufiges Verfahren: Ein Überblick über alle Befehle mit ihrer Syntax (entsprechend den gedruckten Quick-Reference-Cards) und vertiefte Information zu den einzelnen Befehlen (je etwa eine Bildschirmseite). Eine gute HELP-Funktion ist aber sehr aufwendig zu programmieren und dort fehl am Platz, wo sie nur schlechte Programme ‹flicken› soll. Neueste Entwicklungen [Espinosa und Hoffman 1982] lassen vermuten, dass sie sich bei einfachen und konsistenten Interaktions-Modellen erübrigt.

3. Bemerkungen zum Entwurf interaktiver Programme

Über Entwurfsmethoden für Programme und über Projektmanagement gibt es ausführliche Literatur [z. B. Kimm et al. 1979]. Hier werden nur einige Besonderheiten hervorgehoben, die sich besonders auf die Interaktion auswirken.

3.1 Wo beginnen?

Sicher nicht am Terminal. Die Hauptarbeit des Programmierens geschieht mit Papier und Bleistift. Ein gründliches Verstehen der zu programmierenden Aufgabe und der herkömmlichen Lösungen steht am Anfang jedes Programm-Entwurfs. Es gilt dabei, die Problemanalyse nicht mit der Untersuchung bestehender Lösungen zu verwechseln, d. h. die mathematische Struktur des Problems vom Formelapparat des gewohnten Lösungswegs zu unterscheiden. Eine Untersuchung des grundsätzlichen Charakters der zu lösenden Aufgabe kann zu überraschend einfachen Lösungsmethoden führen. Diese kommen dem Problem und dessen Auffassung durch den Menschen meist näher als herkömmliche Lösungen, die mit beschränkteren Mitteln auskommen müssen und deshalb oft umständliche Methoden anwenden (Reihenentwicklungen, schrittweise Lösungen usw.).
Vor jeder Programmierung sind also etwa folgende Fragen zu beantworten: Um was für ein Problem handelt es sich? Welches ist sein mathematischer Hintergrund? Welche Lösungsmethoden gibt es? Welche Entscheidungen soll der Anwender treffen, welche werden automatisiert?

3.2 Das Rad nicht neu erfinden

So ausgefallen die Aufgabe und so originell die Lösungsideen sein mögen, mit grosser Wahrscheinlichkeit wurde das gleiche oder ein verwandtes Problem schon mehrfach gelöst. Ein Studium der Fach- und der Informatik-Literatur sowie Diskussionen mit Leuten, die ähnliche Aufgaben bearbeiten, können manchen Umweg vermeiden helfen. Ähnliches gilt innerhalb von Arbeitsgruppen: Verschiedene Programmierer kämpfen oft mit den selben Problemen, z. B. mit der Gestaltung der Ein- und Ausgabe. Individuelle ad-hoc-Lösungen kosten viel Zeit und sind meist schlechter als einmal gründlich durchdachte Werkzeuge. Sie führen in grösseren Projekten auch zu einer inkonsistenten Benützerschnittstelle, die verschiedene Programmierer-Handschriften trägt.

3.3 Wer sind die Benützer?

Eine möglichst genaue Kenntnis der erwarteten Anwender ist Voraussetzung, um ein Programm diesen anpassen zu können. Bevor der Dialog entworfen wird, sind also die Gewohnheiten und Erwartungen der zukünftigen Benützer genau abzuklären.
Unabhängig von der Anwendung haben ausserdem alle Benützer mit einigen Tätigkeiten und Konzepten grundsätzlich mehr Mühe als mit anderen. Hier eine Gegenüberstellung:

Einfach
konkret
sichtbar
kopieren und verändern
aus einer Liste auswählen
erkennen

Schwierig
abstrakt
unsichtbar
aus dem Nichts erzeugen
in einen Leerraum einfüllen
sich erinnern

Einfaches und Schwieriges für den Menschen (frei übersetzt nach [Smith et al. 1983]).
Während der Implementierung und vor der Inbetriebnahme helfen ausführliche Tests mit Versuchspersonen, viele Fehler frühzeitig zu erkennen. Oft genügen auch Papier- und Bleistift-Experimente, z. B. für die Reihenfolge einzelner Arbeitsschritte, für die Gestaltung des Bildschirms oder für die Formulierung von Fehlermeldungen. Der Kontakt mit den Benützern und die genaue Analyse ihrer Schwierigkeiten bleiben auch nach der Inbetriebnahme die wichtigste Quelle für Verbesserungs-Ideen.

3.4 Von oben nach unten, von aussen nach innen

In der Entwurfsphase ist ein modulares ‹top-down›-Vorgehen üblich und sinnvoll. Die Idee dahinter ist, auf einer hohen Abstraktionsebene, d. h. nahe an der Wirklichkeit zu beginnen und stufenweise die Auflösung für die Besonderheiten der Implementation zu erhöhen. Dies erlaubt, auf jeder Stufe Details auszuklammern, die dort keinen Einfluss haben. Angewandt auf interaktive Programme bedeutet dies, zuerst die Schnittstellen eines Systems zur ‹Aussenwelt› festzulegen: den Dialog mit dem Benützer und allenfalls die Kommunikation mit anderen Programmen oder mit einer Datenbank. Ein schriftlicher Programmentwurf, der den vollständigen Dialog mit allen Menus und (Fehler-)Meldungen enthält, eignet sich hervorragend zur Diskussion mit Mitarbeitern und Anwendern, bevor nur eine Zeile programmiert worden ist.

4. Methoden und Hilfsmittel für den Dialog-Entwurf

4.1 Spezifikations-Methoden

Das Bedürfnis, die Entwurfsphase zu formalisieren, hat eine ganze Reihe von mehr oder weniger formalen Methoden zur Beschreibung von Programmen hervorgebracht. Vielen von ihnen ist die Idee der Datenabstraktion gemeinsam [Isner 1982].
Formale Methoden für den Entwurf von Dialogen sind bisher aber weniger verbreitet. Besonders die erforderliche Konsistenz (4. Postulat) macht aber für grössere Projekte den Einsatz solcher Methoden unabdingbar. Sie ermöglichen, mit geringem Aufwand und anhand klarer Kriterien, Varianten zu studieren und frühzeitig Schwachstellen zu erkennen.
Die heute verwendeten Methoden lassen sich zwei Hauptströmungen zuordnen: Die einen verwenden Zustands-Diagramme, in denen Ein- und Ausgabe-Operationen die verschiedenen Zustände des Systems verbinden [Kieras und Polson 1983]. Die anderen betrachten die Interaktion als Sprache und definieren in Form von Grammatiken (Backus-Naur-Form) oder Syntax-Diagrammen die möglichen Sequenzen von Operationen [Reisner 1981].
Unsere Erfahrungen zeigen, dass formale Grammatiken einfach anzuwenden sind und eine klare Struktur aufweisen gegenüber den oft verwirrenden Diagrammen. (Klassische Flussdiagramme haben sich allzuoft als ein Hilfsmittel erwiesen, das die Komplexität eines Problems nicht nur vermindert, sondern oft sogar steigert.)
Schwierigkeiten bieten noch die Beurteilung der Konsistenz eines Dialogs und die Übersetzung der Spezifikationen im Programmcode. Die Konsistenz versucht man mittels zweistufiger Grammatiken [Turner 1984] zu formalisieren. Eine mögliche Lösung für das Übersetzungsproblem sind sogenannte Compiler-Compiler. Sie übersetzen formale Grammatiken in ausführbaren Programmcode, welcher die Eingaben der Benützer verarbeitet und die notwendigen Berechnungsroutinen aufruft.

4.2 Beurteilungs-Methoden

In engem Zusammenhang mit den formalen Spezifikations-Methoden stehen die Versuche, die Benützerfreundlichkeit von Dialogen zu quantifizieren, um Kriterien für den Vergleich von Entwürfen zu schaffen [Reisner 1983]. Es zeigt sich und ist intuitiv verständlich, dass die Anzahl der notwendigen Regeln in formalen Grammatiken ein gutes Mass für die Konsistenz einer Interaktion darstellt: gleiche oder ähnliche Operationen werden durch die selben Regeln beschrieben. Die Anzahl verschiedener elementarer Handlungen (etwa das Drücken einer Maustaste oder das Verschieben des Cursors) und die Länge der Sequenzen aus solchen Handlungen sind sodann wichtige Indikatoren dafür, ob ein System einfach zu erlernen und zu bedienen ist.
Andere Methoden verwenden die notwendige Anzahl Elementaroperationen und die Antwortzeiten des Systems, um die Zeit zu beurteilen, die ein erfahrener Benützer für das Lösen einer Aufgabe mit dem System benötigt.

4.3 Hilfsmittel

Der grosse und stereotype Aufwand für die Programmierung von Dialogen (50 bis 80% des Codes!) hat zur Entwicklung von Software zur Dialog-Generierung geführt, die dem Programmierer erlauben, in allen seinen Programmen Menus und Masken für die Ein- und Ausgabe mit wenigen Makro-Befehlen aufzubauen. Neben der grossen Zeitersparnis bewirken diese Werkzeuge vor allem eine erhöhte Konsistenz verschiedener Dialoge.
Natürlich hängen solche Hilfsprogramme stark von der verfügbaren Hardware ab (alphanumerische oder graphische Bildschirme, Tastatur, Maus usw.). Bei modernen Kleinsystemen werden sie oft als Software-Pakete mitgeliefert, oder das Betriebssystem enthält bereits die notwendigen Bausteine. Andernfalls ist der Aufwand für den Bau eines eigenen Masken- und Menu-Generators nicht viel grösser als für die Codierung des Dialogs eines einzelnen grösseren Programms. Zu den Werkzeugen für den Bau interaktiver Systeme gehören im weiteren auch Graphik- und Symbol-Pakete sowie die erwähnten Compiler-Compiler.

5. Die Rolle der Graphik

Unabhängig von den graphischen Aspekten einer Anwendung wird Graphik mehr und mehr zu einem Bestandteil *aller* interaktiven Systeme. Eigene Graphik-Prozessoren, hochauflösende Raster-Bildschirme und neuere Eingabe-Techniken, wie Maus oder elektronischer Griffel, ermöglichen eine grundlegende Veränderung des Graphik-Konzepts: von der reinen Illustration der Resultate alphanumerisch erteilter Befehle zu einem Modell des behandelten Gegenstands, das der Benützer am Bildschirm bearbeiten kann.
Dieses Modell sollte so nahe wie möglich an der Vorstellung, die der Benützer von seiner Aufgabe hat, liegen. Es spielt dabei keine Rolle, ob der Gegenstand ein eigentliches graphisches Gebilde (etwa ein Triangulations-Netz) oder ein simpler Text ist. Wesentlich ist, dass in beiden Fällen der Benützer direkt mit den Bestandteilen des Modells arbeiten, sie verändern, verschieben, erzeugen oder löschen kann (‹direct manipulation›). Dies vermittelt ihm das Gefühl, das System zu beherrschen. Zudem muss er keine umständlichen, verschlüsselten Befehle lernen, sondern kann viele seiner Operationen direkt ausführen (z. B. eine Linie zeichnen) und andere durch Menu-Wahl aufrufen. Die damit erzielte bedeutende Steigerung der Benützerfreundlichkeit erklärt sich teilweise mit dem Ursprung vieler dieser Ideen und Techniken in Video-Spielen.
Ein realistischeres Interface ruft aber auch wesentlich höhere Ansprüche der Benützer an die Vollständigkeit und Macht der angebotenen Operationen hervor. Entscheidend ist unter anderem, wie ‹intelligent› ein System auf die Operationen des Benützers am Modell reagiert, d. h. ob es für die Einhaltung

von Konsistenz-Bedingungen sorgt. Diese Frage geht aber über das hier gestellte Thema hinaus (siehe [Frank 1984]).

6. Ausblick

Moderne Arbeitsplatz-Computer, die mit Bit-Map-Graphik, Fenstertechnik, ‹icons› und flexiblen Menus heute neue Massstäbe für die Interaktion von Mensch und Computer setzen (siehe Kasten), sind nicht das Ende, sondern der Anfang einer Entwicklung. Bedeutende Fortschritte sind sowohl in der Handhabung und der äusseren Erscheinung von interaktiven Systemen als auch in ihrem inneren Aufbau und in den Entwicklungs-Methoden zu erwarten.

Flache und grössere Bildschirme, Farbgraphik mit höherer Auflösung, schneller bewegliche Bilder, eine bessere Integration verschiedener Anwender-Software (etwa Text- und Graphik-Editoren), wirksamere Anpassungsmöglichkeiten an verschiedene Benützergruppen, Erkennung und Synthese von natürlicher Sprache und von realistischen Bildern sind nur einige Stichworte zu den erwarteten sichtbaren Verbesserungen in der Beziehung zwischen Mensch und Maschine. Einen guten Eindruck von den bereits bestehenden Möglichkeiten geben uns die hochkomplexen Flugsimulatoren, die eine sehr umfassende ‹künstliche Realität› schaffen. Simulationen spielen eine zunehmend wichtige Rolle auch in vielen anderen Gebieten (Medizin, Maschinenbau usw.).

Im Bereich der internen Verbesserungen sind Betriebssysteme zu erwarten, die auf gleichzeitige Eingaben von verschiedenen Quellen (Tastatur, Maus, Sprache usw.) ausgelegt sind, im Gegensatz zu den heutigen Standard-Betriebssystemen, welche durchwegs noch auf der Idee eines einzelnen Eingabe-Stroms pro Benützer (Lochkarten!) aufbauen.

Die Methodik bei der Entwicklung interaktiver Systeme schliesslich wird ihren ‹Versuch und Irrtum›-Charakter mindestens teilweise verlieren und auf soliderer theoretischer Grundlage versuchen, die Benützerfreundlichkeit von Systemen schon im Entwurfsstadium analytisch zu beurteilen. Integrierte Programmier-Umgebungen, die, ihrerseits mit hoher Interaktivität, die erwähnten Hilfsmittel anbieten, werden die ‹alleinstehenden› Editoren, Compiler und Debugger ablösen. Die Ebene, auf der wir unsere Programme formulieren, wird ständig höher, was uns erlaubt, stärker von den Besonderheiten der Ausrüstung zu abstrahieren und immer mehr die ‹natürliche› Sprache der Probleme zu verwenden. Als Anwendungs-Programmierer stehen wir am Übergang von den Sprachen, mit denen die einzelnen Schritte einer Operation formuliert werden, zu solchen, mit denen nur noch das gewünschte Resultat beschrieben wird. Direkt ausführbare Spezifikationen von Programmen erlauben, innert kurzer Zeit Prototypen für den Dialog wie für den Rechenteil zu erstellen und mehrere Varianten zu prüfen. Die problemnahe Formulierung erhöht auch wesentlich die Korrektheit der Programme.

Dieser Artikel will Erfahrungen weitergeben, die wir beim Entwurf und Einsatz interaktiver Programme gemacht haben. Ein Teil davon ist in den Postulaten formuliert, die vor allem auf häufige Schwachstellen in solchen Programmen aufmerksam machen sollen. Unsere wichtigste Erfahrung ist aber, dass die einer Problemlösung zugrunde gelegte mathematische Struktur den grössten Einfluss auf eine gute Benützer-Schnittstelle hat. Wenn dort Unklarheit herrscht, nützen die besten Richtlinien für die Dialog-Gestaltung nichts. Wenn aber die zu programmierende Aufgabe, unabhängig von bestehenden Lösungen, klar herausgearbeitet wurde, schlägt sich das meist auch in einfach zu lernenden und einfach zu benützenden Programmen nieder.

Literatur:

Degano, P. und Sandewall, E. (Eds.): Integrated Interactive Computing Systems. Proceedings of the European Conference ECICS 82, Stresa North Holland 1983.

Espinosa und Hoffman: Macintosh User Interface Guidelines, Apple Computer Inc. 1982.

Frank, A.: Computergestützte Planerstellung – Graphik oder Geometrie? In: Vermessung, Photogrammetrie, Kulturtechnik 12/84

Isner, J.F.: A programming Methodology Based on Data Abstraction, In: Bulletin Geodesique vol. 56, No. 2, 1982.

Janda, A. (Ed.): Human Factors in Computing Systems. CHI'83 Conference Proceedings, ACM, Boston, December 1983.

Kieras, D. und Polson, P.G.: A Generalized Transition Network Representation for Interactive Systems. In: Janda 1983, pp. 103–106.

Kimm et al.: Einführung in Software Engineering. DeGruyter 1979.

Reisner, P.: Formal Grammar and Human Factors Design of an Interactive Graphic System. In: IEEE Transactions on Software Engineering SE-7, 1981, pp. 229–240.

Reisner, P.: Analytic Tools for Human Factors of Software. In: Blaser und Zoeppritz (Ed.): Enduser Systems and their Human Factors, Lecture Notes in Computer Science, vol. 150, Springer 1983.

Smith, D.C. et al.: Designing the STAR User Interface. In: Degano and Sandewall 1983, pp. 297–313.

Turner, S.J.: W-Grammars für Logic Programming. In: Campbell (Ed.): Implementations of PROLOG Edis. Horwood Ltd. 1984.

Adresse des Verfassers:
Werner Kuhn
Institut für Geodäsie und Photogrammetrie
ETH-Hönggerberg, CH-8093 Zürich
z.Z. University of Maine at Orono
Dept. of Civil Engineering
103 Boardman Hall, Orono ME 04469, USA

Die Doppler-Messkampagne SWISSDOC: Ein Beitrag zur Landesvermessung in der Schweiz

A. Wiget, A. Geiger, H.-G. Kahle

In den Monaten Juli und August 1984 hat das Institut für Geodäsie und Photogrammetrie (IGP) der ETH Zürich in Zusammenarbeit mit mehreren ausländischen Instituten Geländemessungen für die Doppler-Messkampagne SWISSDOC (Swiss Doppler Observation Campaign) durchgeführt. Das Ziel dieses satellitengeodätischen Projektes bestand darin, die Lage- und Höhenkoordinaten von ausgewählten Punkten des Schweizerischen Landestriangulationsnetzes mit Dopplermessungen an U. S. TRANSIT-Satelliten zu bestimmen.

Institut für Geodäsie und Photogrammetrie, ETH-Hönggerberg, CH-8093 Zürich,

Separata Nr. 82

Messmethode des TRANSIT-Systems

Die Methode der Dopplermessungen an den TRANSIT-Satelliten des U.S. Navy Navigation Satellite Systems (NNSS) wurde in dieser Zeitschrift bereits ausführlich beschrieben [1]. Sie soll hier nur kurz erwähnt werden.

Die Grundlage des TRANSIT-Systems bilden heute sechs Satelliten, die von den USA zwischen 1967 und 1979 in