Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

H.J. Adèr, D.J. Kuik, E. Opperdoes, B.F. Schriever

The use of conversational packages in statistical computing

Department of Mathematical Statistics Report MS-R8506 September

The use of Conversational Packages in Statistical Computing

H.J. Adèr
Department of Psychology, Free University,
P.O. Box 7161, 1007 MC Amsterdam, The Netherlands.

D.J. Kuik
Department of Medical Statistics, Free University,
P.O. Box 7161, 1007 MC Amsterdam, The Netherlands.

E. Opperdoes
Central Bureau of Statistics, P.O. Box 959, 2270 AZ Voorburg, The Netherlands.

B.F. Schriever
Department of Mathematics and Computer Science,
Free University, P.O. Box 7161, 1007 MC Amsterdam, The Netherlands.

It is not generally recognized that conversational computing demands a different way of using statistical analysis. However, special problems arise while applying statistical techniques repeatedly on the same dataset. The question is put whether unexperienced users should be protected against this kind of improper use of a conversational statistical package (CSP). In section 3 we list some formal aspects of conversational communication. Thereafter user-objectives in the use of the package are considered and consequences for the user-interface formulated. Finally some technical questions are treated.

1980 Mathematics Subject Classification: 62-04
Key Words & Phrases: conversational computing, statistical computing, evaluation of software, conversational statistical packages.
Note: This report will be submitted for publication elsewhere.

Report MS-R8506
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1. INTRODUCTION

In the past few years the interactive use of computers has increased enormously. The usefulness of conversational *) computing has been recognised earlier in the administrative field than in computational statistics and, although techniques that are most useble in an conversational environment like Exploratory Data Analysis (Tukey, 1977; Velleman & Hoaglin, 1981) have been introduced and implemented, the conception that conversational computing demands a different way to use statistical analyses is not at all generally accepted. This new way to use statistics raises questions on the proper use of techniques, and consequently on the development of packages which are not only foolproof but also reflect a sensible view on the way they should be used. Developers are usually not concerned about meeting methodological requirements for their products. About other - say ergonomical - requirements, no user to our knowledge has ever been consulted by any package developer. The term "user friendliness" has been thought of only recently and the quality of packages in this respect is often rather poor. (Userfriendliness is discussed by van Apeldoorn, 1983, and shortcomings in SPSS by Davis, 1983.)

Not too long ago the idea arose that one should evaluate the existing (batch) packages (Francis 1977,1982; Hext, 1982; Wilke 1982). Also, in Cognitive Psychological research, several aspects of Man Machine Interaction have been looked at, especially those relevant to the user interface as show the many articles in, for instance, the Journal of Human Machine Interaction and the Sigchi Bulletin of the ACM. But although literature in both fields gets more extensive, this did not result in substantial changes in any package. However, evaluation of statistical (batch) packages has increased the understanding of the pros and cons of their use (see Molenaar, 1984).

*) A subtle difference in meaning exists between the terms interactive and conversational: the first applies in all situations where use is made of interactive terminals to channel the contact between user and computer. This contact is called conversational if the interaction is a dialogue, i.e. if it takes an alternating question-response pattern.

We do not give a critical survey of existing packages. For batch-packages this has already been done by others (Francis, 1981; Lauro & Serio, 1982; Cable & Rowe, 1984) since there are not too many Conversatio-nal Statistical Packages around (see however Kuik (1984) about QUESTOR; Reid (1980) gives a list of packages used in the Britain), we felt we should think about the subject more generally.

We describe in general terms, some concepts that define the 'context' in which a "sufficiently good" package might exist. Therefore this work should be seen as an stocktaking of tools and as material for a discussion on concepts.

Hopefully, not only discussion is started by this article but also future design of Conversational Statistical Packages (CSP for short) is influenced by it.

Moreover, since this paper roughly formulates what to expect of Conversational Statistical Packages, it should be easier to formulate norms to be used to evaluate such packages. We intend, as a follow-up of this study, to make up such norms and to evaluate existing packages.

Most generally this article is meant as a working paper for all those people that are interested in the field of Computational Statistics. More specifically we adress ourselves to the people who meet CSPs in their professional career. Here one can make a subdivision in three: on one hand there are the developers of these CSPs. They may find some ideas which may help them in designing friendlier systems. On the other hand there are the users of such a package. They might be interested in the discussions the (statistical) problems arising while using a CSP. But mainly we address ourselves to a third group, the people who are responsible, possibly as an advising specialist, for the purchasing and installation of a package. We call them 'distributors' hereafter. They have to make up for specific requirements, related to the wishes of their user groups.

In the second chapter we formulate what kind of users might be expected to use a CSP. Some special problems that are caused by the interactive applying of statistical techniques on the same dataset are dwelled upon. This leads to considerations on the responsabilities of user, distributor and package-developer.

In chapter 3 the more formal aspects of conversational communication is treated.

In chapter 4 aspects of the user-objectives in his or her use of the package are considered, consequences for the user-interface are formulated.

In the fifth chapter some technical questions are treated.

Parts of this paper have been published before (Chapter 2: Kuik & Schriever, 1984; chapter 4: Adèr & Kuik, 1983; Adèr, Kuik & v.d. Veer, 1984).

The work on this article has been performed by a subgroup of the "ADSARA Subcommittee on Statistical Software". This subcomittee has an advisory task towards ADSARA, which stands for the board of users of Stichting Academisch Rekencentrum Amsterdam (SARA), the central computing centre of the University of Amsterdam, the Free University and the Centre for Mathematics and Informatics, all three in Amsterdam.

2. STATISTICS AND THE CONVERSATIONAL PACKAGE

2.1. User groups

Nowadays the applied statistician is not the only kind of user of statistical programs. In fact there are many different users of statistical packages orginating from all disciplines.

These user groups may be classified with respect to the following three characteristics:

1. application field (e.g. social sciences, economics, biometrics)
2. statistical ability
3. skill in package-handling

The first characteristic has consequences for the content of the CSP with respect to techniques, data structures and user language.

For example, for use in the social sciences principal component analysis should be incorporated, economists need time series analysis, whereas for the use in biometrics bioassay techniques should be present. Correspondingly, different data structures are indicated: typical are grouping data in biometrics, time series data in economics and categorical data in social sciences.

A lot has been written on human machine interfaces. All that is true in general, is true for the special case of statistical packages. The existing interfaces of statistical packages cannot be called examples of user friendliness (see for instance Davis on SPSS, 1983). Especially novice users suffer from this: ideally, for a package used in a course on statistics, special provisions should be made for the user interface.

The second and third characteristic ask for extra facilities for users with different needs for assistance.

This variety of intended customers makes it far from easy to construct a "general purpose" Statistical Package. One can take different views on the desirability of the construction of such a package. The user with a high level of the above mentioned skills will be glad to have a package at hand with which he can perform a scala of different tasks. However, packages like that tend to get rather enormous and usually ask for specific abilities of the third kind. Packages directed versus a special group of users have some advantages: the application field is no longer a problem, the statistical ability and skill in package-handling are well defined.

Generally speaking, a package well constructed in these respects might stimulate the application of appropriate statistical techniques considerably.

2.2. (Dis)advantages of interactively applied statistics

Statistical batch packages are often incorrectly used, due to the fact that they are so easily available to the user. It are not just a few who are tempted to run some programs first and think later, instead of the correct reverse procedure. In the case of conversational packages access is eased yet and, consequently, the danger of incorrect use becomes even greater.

Let us have a closer look at some important causes for incorrect use.

A general cause, to be met with all kinds of statistical analysis, may be called "departure from underlying assumptions". Depending on the (statistical) robustness of the technique, neglecting this point can entail completely absurd conclusions since in essence a wrong technique is applied. Because a CSP enables the user to perform many different techniques unhammered one after another, a more specific danger consists in testing sequentially without adjusting the level of significance. This may lead to what is known as "capitalization on chance": one continues performing statistical tests until a "significant" result is found. Another danger of doing many analyses on the same dataset is the dependence of the conclusions: a significant result may affect the "significance" of other results (in a purely deterministic way).

It is obvious that these dangers exist only in confirmative analyses, since here we try to give a statistical proof of some phenomenon, applying the very strict rules of statistical analysis, whereas on the other hand in exploratory analysis we often neglect these strict rules and search for phenomena of interest. *)

Thus far for the dangers. What might be said about their prevention? First of all: does the user wants to be protected? Molenaar (1984) does not favor this idea: "I do not think a package has the Biblical obligation to be its user's keeper, and many users will resent warnings, let alone refusals, in cases such as very unequal group variances or very small cell expectations."

The question whether protection should be given is one to be solved by the designer. One could think of a package offering the possibility to activate an warning mechanism against improper use, to be switched on (or off) by the user.

If one decides that protection should be given, is it possible to conceive an "ideal" CSP which would protect the user against all pitfalls of confirmatory analysis? The answer seems to be negative for two reasons:

*) As Tukey puts it, Exploratory Data Analysis and Confirmatory Data Analysis compare like the work of the detective and that of the judge: "Exploratory data analysis is detective in character. Confimatory data analysis is judicial or quasi-judicial in character.....
Unless the detective finds the clues, judge or jury has nothing to consider. Unless exploratory data analysis uncovers indications, usually quantitative ones, there is likely to be nothing for confirmatory data analysis to consider -." On the other hand: "Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone -- the first step." (Tukey, 1977)

Firstly, the construction of a warning mechanism is met with severe technical problems; in many cases underlying assumptions are hard to verify, while in others the assumptions are inherent to the experimental design (e.g. randomness of a sample) and cannot be checked on the data. A modest protection against dependent analyses is possible when the same technique and the same variables are concerned. Otherwise complications may occur e.g. when one uses two different techniques on the same dataset (Notice that this also is the case when model assumptions are tested first, after which the confirmatory technique is applied.) Paradoxically, it looks as if a correct statistical procedure can lead to incorrect use of the data!

Secondly, the package can read the data but it cannot read the user's mind; it does not know whether the user intends to do an exploratory or a confirmatory analysis. In some cases the user may wish to try several techniques on the same variables in order to "explore" the information that is contained in the data. Evidently, no statistical proof is yielded by such an exploratory analysis. It may only produce some hypotheses which can be tested later in a confirmatory analysis on a different (independent) dataset. One way to put such a two stage approach into practice is to take a relatively small, random sample from the dataset at hand and use it for an exploratory analysis and after that to perform a statistical test of the hypotheses on the remaining data. We therefore think a full protection system is impossible and even consider the suggestion of such a powerfull protection system harmful: the user may think that any statistical analysis that passes (or escapes) the system is correct, which in many instances simply is not true. The package should restrict itself to warnings about what can go wrong in applying the technique at hand. In this way the user is informed about the risks he takes and is better aware of his own responsibility.

In our opinion, one of the most important justifications for the existence of conversational statistical software is the possibility to do exploratory analyses; in a correct confirmatory statistical analysis the technique, the relevant parameters and the data are chosen in advance and therefore one or two (batch) runs are sufficient.

2.3. The distributor's responsibility

Although we argued above that there exists no complete protection against incorrect use, it is still possible to offer protection to some extent. The question arises to what degree the distributor should be held responsible for the incorrect use of his software. One can take a stand somewhere between the two extremes: either the user or the distributor is completely responsible. The user is imperfect and liable to error. Therefore he will be glad to have some protection against his own fallibility, as long as he can work in the way he wishes. On the other hand, it seems reasonable to demand from the distributor that the software he supplies offers some protection against incorrect use.

The following methods could be thought of:

1. The user can choose, according to his statistical knowledge, between two levels to use the package: advanced or non-advanced. After this level has been established, the package may select some programs according to their degree of safety and prohibit the use of others (I.e. an inexperienced user might be barred from using the more esoteric rotation methods in factor analysis.)
2. The decision to offer protection or not depends on the aim of the analysis (exploratory/confirmatory). A more detailed discussion of this method is given in the following section.

But there is a third method that in our view is the most important:

3. Keeping a record of the history of the dataset in which every action that operated on the dataset is recorded. In this way it is possible to trace the genesis of a new variable which is a function of some others, to signal the use of the same variable or subsets of the dataset in two separate analyses, etc.

   It is obvious that this feature offers many possibilities for protection *). A disadvantage is that keeping a complete record of all actions and frequently consult this database (which can become quite large) may be very storage and time consuming.

*) Also, when doing a confirmatory analysis in a scientific context, one is often obliged to record in his article the data, data transformations and techniques used (reproducibility of scientific studies).

However,when the above methods are combined a less complicated protection stystem may be sufficient.

2.4. Description of a moderate protection system

In this section we indicate a possible way to implement the ideas exposed in the previous section.

Let us consider any dataset and analyses that can be applied to (subsets of) this dataset. The central idea runs as follows: elements of the dataset always have one of the following two labels: USED or NOT USED. On the other hand, analyses are devided in two predefined classes: CONF(irmatory) or EXPL(oratory). If an anlysis of the first class (CONF) is applied to a subset that contains USED elements an errormessage is given, if all elements are NOT USED, they recieve the label USED afterwards; an EXPL(oratory) analysis can be applied to subsets with mixed (i.e. containing USED and NOT USED) elements , NOT USED elements recieving the label USED afterwards. One can describe this proces more formally (see appendix A).

In this way, the history keeping in function of the protection system is unnecessary: extra actions needed are the checking of the elements of a subset if an confirmatory analysis is due and the changing of labels after application of an analysis. It does not seem useful to us to indicate which variables have been involved, since that still does not guarantee the independence of test outcomes. Moreover, it is often too complicated.

Further, the output of the package is thought to reflect the nature (exploratory/confirmatory) of the results.

If the package has a way to ascertain the statistical ability of the user (see method 1 in 2.3.) the following extra features may be considered: the 'advanced' user has the possibility to override element labels (i.e. to use any subset for a confirmatory analysis, if he so wishes), while for the 'nonadvanced' user the package tries to check the applicability of a certain confirmatory analysis on the asked-for data subset.

3. COMMUNICATION

Data analysis - and in particular data exploration - leans heavily on the interpretation of numbers, tables and pictures. Any CSP should be so designed that it helps, not hinders this process. It should be "comfortable"; i.e. a user should neither feel bored nor tired after a long session. One relief from boredom can be found in fast response times. This, of course, is not only a question of package design. In section 5.1 we make some remarks about the relation between response time and package development.

Directly related to the user comfort is the way communication takes place between the CSP and the user. And this definitely is a question of package design. Several aspects of this communication are discussed in this chapter.

3.1. Flow of information

To perform the tasks as intended by the user, the CSP needs some specific information. (e.g. the analysis to perform; where to find the data; how to present the results; etc.) Therefore a flow of information exists from the user to the package. But then the question arises: who decides which kind of information is needed at a given moment? Since there are two different participants in the communication, it can be package- or user-controlled. Consequently, two different classes in forms of communication can be distinguished, each with its own logic and problems.

If the package is in control, the communication becomes query-like: the package poses questions and the user has to answer them (see also Kuik & Hasman, 1983; Kuik, 1984). When the user is in control, the communication becomes keyword-oriented: the package has to recognize certain key-phrases in the information the user has to offer. Aspects of this dichotomy and the forms it takes are discussed in section 3.2.

There also exists a flow of information from the package to the user. Mostly this concerns error messages, help facilities, comments, etc. This will be discussed in section 3.4.

For both directions of flow one may speak about the intensity of the communication. This is an important aspect of comfort, since intense (comprimized) communication takes a lot of concentration, and may become tiring after a period, while long-winded communication can become extremely boring. These and related concepts are discussed in section 3.3.

3.2. Forms of communication

In this section we describe the more common forms of communication. In a CSP usually more than one is used; mixed forms occur also. These 'basic' forms can be divided into two classes according to the dichotomy mentioned above: package or user controlled.

3.2.1. Package controlled forms

The basic forms of the package controlled forms are :

[a1] yes/no questions.
The package produces a question to which the user must answer either yes or no.

[a2] menu scheme (multiple choice questions).
This is an extension of [a1]; the package produces a question with several possible answers from which the user must select.

[a3] information directed questions.
The package asks for specific information such as a filename, the type of statistical technique to be used or the number of variables on a file.

[a4] form defined questions.
This form is only possible on a video display or a graphic terminal. The package displays a complete question form on the screen. The user has to fill in answers to all the questions at predescribed places and return the completed form to the package. For some questions the package has already suplied default values, which can be (but need not be) overwritten by the user.

[a5] position defined questions.
This form is also only possible on a video display or a graphic terminal. With cursor, hairlines, joystick, lightpen, mouse or some other means the user responds to a given question by returning one or more positions to the package. This allows a very flexible way of

data transmission and parameter setting. As an example consider a situation, in which one has a scatterplot of observations in twodimensional space. In such a plot outliers are easily detected; therefore the easiest way of deleting them is by pointing at them with the joystick (or light pen). Another example is the selection of the wanted option in a menu sceme with a joy stick.

3.2.2. User controlled forms

The basic user controlled forms are:

[b1] keywords and procedure type instructions.
The user gives the name of a procedure followed by the parameters, or gives a keyword followed by one or more values.

[b2] Instructions in a pseudo natural language.
This is in fact a more elegant presentation of [b1].The information about the next instruction is given by certain key expressions, which may be embedded in a language-like sentence.

[b3] Instructions in a natural language.
The user gives his instructions to the package in a natural language sentence. The package evaluates and interpretes this and reports his interpretation to the user, together with additional questions if some information is not clear or incomplete.

3.2.3. Example

Suppose we want to perform a Student t-test on the variable AGE, where the two groups to be compared are indicated by MALE and FEMALE. The data might be found on some external file DATA.
For the first group we combine the forms [a1] and [a3] (questions are in lowercase, answers in uppercase).

```
[a ] Which analysis do you want to perform ?
     T-TEST
     between which groups ?
     MALE,FEMALE
     on which variable ?
     AGE
```

on which file can I find your data ?
DATA

For the second class examples for all forms are given:

[b1] T-TEST,VAR=AGE,GROUP1=MALE,GROUP2=FEMALE,FILE=DATA.

[b2] RUN the T-TEST ON the variable AGE BETWEEN the two groups MALE and FEMALE OF file DATA.
Here the texts written in uppercase are the key expressions, the embedding, which is optional, is written in lowercase.

[b3] I WOULD LIKE TO KNOW IF THERE IS A DIFFERENCE IN MEAN AGE BETWEEN MALES AND FEMALES. COULD YOU PERFORM A T-TEST FOR ME ? THE DATA MAY BE FOUND ON THE FILE DATA.

In many situations the package-controlled forms can hardly work efficient on their own. A combination of [a2], [a4] and [a5] is then most comfortable.

Since the package has the initiative, a situation can easily arise in which the user doesn't know what to do or what the package wants from him. To alleviate this problem, extensive help facilities are needed.

To evaluate the forms in user controlled situations, the following considerations are to be made: In [b1] the user needs to know the exact order and format of the keyword instruction line. In the "natural language" case [b3] a large overload of typework is to be done. Therefore at the moment the form of pseudo natural language [b2] is preferable. When in the (near?) future acoustical input techniques become available, the natural language might be the preferred alternative.

A major problem in user controlled situations lies in the fact that the user now has to know everything: all available keywords, their parameters and the order in which they have to be called. So also in this situation there evolves a need for help facilities.

The package controlled situation is easiest on the less experienced users of a CSP, whereas for experienced users the user controlled situation is often preferable. Gilfoil (1981) reports an experiment in which a menu-driven dialogue is found appropriate for novice users. After approximately 16-20 hours of task experience transition to a command driven dialogue is preferred.

3.3. Variations in dialogue

Several variations in the dialogue are possible. The main types are the variation in input and output intensity ,such as:

Verbosity

This is the global amount of text the package produces around each instruction. This should be sufficient for good understanding but not too much more than necessary: it can be extremely tiring when sitting behind a video display, letters and numbers flash by constantly. The opportunity for the user to adjust the verbosity to his own needs provides a good comfort.

Density

This can be described as the amount of questions (from the package) or procedure and keywords instructions (from the user) required to describe an action. The density is closely related to the form of communication; for instance YES/NO questioning always has a higher density than the menu scheme. In some cases it is also related to the experience and (statistical) knowledge of the user; in a keyword-oriented package he may for example have defined a procedure in which several actions are combined. Thus, by typing his self-defined keyword, he can trigger a whole sequence of actions. This way he lowers the density.

Display inertia

This is the amount of information remaining on the screen after the user has given a command or answer. When there are many changes at these moments, the package has a low display inertia.

Thus, it is convenient for the user to be able to select levels for these three types of intensity. Lower levels make the analysis more compact, and hence the screen does not change much every time when input is required or output produced. This may be pleasant for the user; however there is a greater risk of forgetting to replace default values of important parameters and hence of making mistakes.

Other variations in the dialogue are for instance variation of the layout of the screen, variation in size of letters and figures, etc.

3.4. Aids and support

The package should be a guide for the user during the dialogue. In some cases it can even anticipate and therefore warn the user for errors to be made. Of course, a user who does not want to be helped, should be enabled to turn off this guidance.

Ingredients necessary for good support of the dialogue are:

Error handling

Messages should be clear and succint. They also should be polite and unintimidating; bells and other noises should be used sparingly. Errors should not lead to abrupt and unintelligible abortion of the session. Therefore errorhandling by the operating system should be prevented. The user should be given the opportunity to correct mistakes, for example by editing an incorrect command line, or by returning to the last-but-one question at will. The package should ask for confirmation of any command, that asks for drastic changes (such as deleting variables or files). It should allow for Murphy's law: any mistake that can be made, will be. The package should be forgiving: if some answer makes sense with a minor change, the package should do so (or ask for confirmation of the corrected line). It must be easy for the user to correct mistakes !

Comments

When (statistical) comments are called for (e.g. singularity of a matrix, too many missing values, non-Gaussian data), the package may do so with alacrity. Also other comments should be given as clear as possible, and as often as needed.
The way the comments are given will depend on the choosen level of intensity. The level on which they are generated may be influenced by the preselected level of the user (see the discussion in chapter 2). So an 'advanced' user can get other comments than a 'nonadvanced' user.

Help facilities

These should be as extensive as possible. Several kinds of facilities are indicated: clarifying the question, giving information about the possible answers, giving insight in the requested statistical method, switching verbosity or density, a mailing service, etc. They should be available at any time and in selectable (useful) portions.
For different users there should be given different amounts of information, related to the verbosity the user has selected; for nonadvanced users there should be tutorial information at all crucial points. There also should be facilities to provide help in correcting errors.

Technical aids

Video displays often have possibilities for technical manoeuvres: blinking, half intensity, inversing, even colour can be used. All thes aids should be used discriminatingly: it can be extremely tiring, trying to extract information from a display like a patchwork quilt. Especially in blocks of text, a surfeit of colour changes will not add to the informative content ! On the other hand, the judicious use of colour can turn an otherwise untransparant plot into a sensible picture.

Retracing

It should be possible for the user to make some steps back in his (explorative) statistical analysis. For instance consider the case that the desired dataset is obtained after some difficult manipulations. Suppose that we then applied the wrong statistical technique. In such cases it must be possible to return to the dataset without doing the manipulations all over again. Facilities to make steps backwards in the analysis do need a good recording of the history of the interactive session.
It is also convenient when the user can interrupt his interactive session and proceed an other time (e.g. the next day). For this aspect see also section 5.5.

3.5. Intelligent interfaces

Communication between human beings is facilitated by their common knowledge. The absence of this in man-machine communication is one of the main sources of the apparent lack of intelligence of the machine. If a CSP could be designed as an "expert system" using a "knowledge base" containing appropriate statistical notions, many of the desiderata mentioned before may be answered easily (see Chambers, 1982).

Recently, experimental packages have been developed that honor this idea (Gale & Pregibon, 1984; Pregibon & Gale, 1984). These "expert systems" react more intelligently on user input, even making suggestions about the line of action to be taken. Thus far this has been done only for regression analysis and the implementation puts heavy demands on storage, apart from requiring special terminals.

As a first step, one could think of an interface embedded in an more or less intelligent environment as has been implemented for the interactive programming language B (see Nienhuis, 1983) and is being investigaged for the Ada *) programming language (DoD, 1980; see also Kernighan & Mashey (1979) on the Unix Programming Environment). For the design of keyword-oriented CSP's many of these idea's might be useful. As an example, the suggestion mechanism might be very attractive, since the interface inevitaly makes use of a fair amount of keywords.

4. THE INTERFACE IN VIEW OF USER OBJECTIVES.

In this chapter our aim is to describe in a general way what characteristics a language between user and package should have in view of the objectives the user has in mind. At the moment, much work is done concerning the cognitive aspects of the user interface. Moran (1981) describes a method and a formalism to describe an interface (see also Saja, 1985). Here we try to discribe that part of the system which he calls the "Conceptual Component" (Task and Semantic Level).

*) Ada is a registered trademark of the United States Governement (Ada Joint Program Office).

The Task Level is described in the following way: "The user comes to the system whith a set of tasks he wants to accomplish. The purpose of the Task Level is to analyse the user's needs and to structure his task domain in a way that is amenable to an interactive system. The output of this level is a structure of specific tasks that the user will set for himself with the aid of the system." Additionally, the Semantic Level is thus described: "A system is built around a set of objects and manipulations of those objects. To the system these are data structures and macro's; to the user they are conceptual entities and conceptual operations on these entities. The Semantic Level lays out these entities and operations. They are intended to be useful for accomplishing the user's tasks, since they represent the system's functional capability. Thus, the semantic Level also specifies methods for accomplishing the tasks in terms of these conceptual entities and operations."

We will elaborate this idea and, using general terms, give a description of the objectives of a user that utilizes a conversational statistical package. We will also describe the implied requirements for the application language and the interface.

We first formulate what kind of basic concepts should be extant, and what actions the user should be able to perform.

Though the contents of this chapter are mainly concerned with the form of communication we called user controlled in chapter 3, most of the concepts and ideas have consequences for the querylike situation, too.

In almost any formal language strained relations exist between on one hand meaning and content of the actions one wants to express and, on the other hand the administrative information required by the language itself. These strains are more perceptible if the language is more designed with an eye on the special facilities and constraints of computing machinery. (As an example, think of somebody who wishes to calculate n!. The only thing he is interested in, really, is the final result. Most programming languages in this case would require specification of conceptually irrelevant details, such as that the computation concerns integers. However elegant the construction of the programming language, details remain details from the user's point of view.)

As stated by Koster (1979), choosing the basic concepts of a computer language is an art: the student of this art should be aware of the constructs that are used while stating a problem initially, in the naive stage where computational considerations are immaterial. The importance of keeping a language as close as possible to these concepts becomes even greater if the user of the language is less experienced in the handling of the machine. This is the case with our user: we may assume that he knows more about his application field than about statistics, and more about statistics than about computation.

A description of the objectives of a userlanguage (or -interface) in general terms doesn't seem to be regular practice. (Compare the development of the high-level language Ada (DoD 1978; Ichbiah et al. 1979): in that case design considerations have been explicitly expressed.) In this chapter we formulate design considerations without designing a userlanguage. In fact we only formulate some idea's that could be embodied in the design of a more or less ideal package. To formulate what we want without actually sketching a new language asks for the definition or description of some basic notions. This is done in the next section. Modes, Actions and Datastructures are the most important concepts, but some attention is paid also to the notions of "internal" vs "external" information, macro's and Flow-of-control. These notions are used in section 4.3 to describe user objectives.

4.1. Basic concepts

4.1.1. Modes

An attribute can be ascribed to an instruction sequence suggesting its intention. We call these attributes modes and we will recognize specification-, input-, throughput- and output- mode.

A few examples may demonstrate the use of these terms:

In specification-mode parameters are specified, data structures are defined or macro's declared.

Data are read in input-mode.

In throughput-mode, statistical computations are performed intermingled with data manipulation.

In output-mode the user can inspect his output and select parts of it.

One should not think of modes as consecutive phases in the interactive session. During the session modes alternate constantly. For instance, principal component analysis would include input of an external correl-ationmatrix (input-mode), producing principal factors (throughput-mode) inspection of the output (output-mode), computation of a rotation matrix (throughput-mode) and producing a plot of the variables on the main axes (output-mode).

Modes differ in degree of activity: we call specification-mode passive compared to the other three, that are labeled active. Here "passive" means that instructions given by the user only contain information and do not call for action. In the other categories direct action is taken.

4.1.2. Data structures

A data structure is considered to be any kind of information, (eventually) accessible to the CSP and of interest to the user. One could think of the raw data file, some matrix internal to the system, a subset of the variables, etc. But also the contents of a screen on the terminal could be considered a data structure.

4.1.3. Actions

The concept of data structure has meaningless without the concept of operators or procedures applicable to datastructures. We will use the term action in this connection. An action is activated by one user-instruction. If the user asks for a T-test, the action is the complete computing sequence, not the separate components, such as the computing of the mean of one of the two groups.

4.1.4. Flow of Control (FoC)

All actions take place in a certain order, which is defined by the information the CSP receives from the user. Sometimes a situation arises in which an action only may take place as long as a certain condition is fulfilled. We then say that the flow of control is controlled by this condition.

4.1.5. Macro's

In the specification-mode macro's may be declared. In 4.2 and 4.4.1 we go into the concept of "macro" in greater detail. For the moment we can think of a macro as a new, user-defined set of instructions, composed of a set of data structures and the actions to be taken upon them. A macro can be called by name at all appropriate instances.

4.1.6. Internal and External Information

From the user's viewpoint there are two kinds of data: the data "known" (in structure and location) to the CSP and data not yet introduced into it. The last kind of information is called "external". Upon recieving data, the CSP is likely to organize it into an easy-to-handle structure. Information reorganized by the CSP is called "internal". Even when the CSP stores it on some external medium to facilitate retrieval we speak of an internal file. The reader may compare this to system files or work files, as known in many statistical packages.

4.2. Macro's and Flow of Control

There is a difference between flow of control in the active modes and in specification-mode. In the active modes user instructions cause the system to act; flow of control alternates between user and package. Macro definition takes place in specification mode: the body is retained for further use, and no action is taken.

The possibility to specify a macro is one way in which the user could influence the density: in specification mode several actions can be combined into a macro. Like "normal" routines in a programming language a macro should contain, apart from its body:

- name, and
- parameters.

This user defined name is used to call the macro. Input-parameters provide the macro with information, information is delivered in the output-parameters. Later on during the session the specified macro name functions as keyword in the same way the standard keywords do. When called with actual parameters a set of instructions is executed that without the macro defition should have been called one by one.

4.3. User objectives

What objectives does the user have in mind while interacting with a CSP? One can differentiate between actions which the user aims at, such as performing some kind of analysis (we call this kind of actions substantial) and actions, that are necessary for the system to function.
This last kind of actions can be further divided in data manipulaton and administrative actions.
In most cases the user concentrates on substantial actions, whereas other actions, to him, are a not particularly interesting necessity.
One could say that the following categories (in order of interest):

- executing analyses
- data manipulation
- administrative actions

comprise the actions the user wants the system to perform.
Consequently, the CSP design should automatize, as much as possible, all actions needed in the second and third categories.

4.3.1. Executing analyses

Preceeding execution of an analysis the data structures needed should be specified. We can think of two kinds of data structures in this context: the data on which the analysis is performed and the parameter vector determining the process. This vector may be filled in by a routine that questions the user or it may be given as a (row of) parameters to the analysis to be activated. One should be able to pass this vector by name to the analysis routine.

In packages like STAP (1980) or BMDP (Dixon, 1981), external matrices can be entered when required. In this case we assume that some action is taken to transform the external file into an internal one before it is used. Therefore we can state that analysis act upon internal data files.
After the analysis is performed, data structures may remain, resulting from the diffent modes that have been passed through. Intermediate results

like matrices containing weights or a variance-covariance matrix can be made available for further use, their structure and name being recorded internally at the moment of their creation. All remaining datastructures should be accessible by name. This is more obvious for intermediate results as it is for input- and output-structures. The mentioned systemfiles provide an example of the first. *) Since the user wants to be able to handle output as flexibly as other datastructures, also output structures should be accessible by name.

4.3.2. Data manipulation

In specification-mode, data structures needed for the actions performed during the active modes are defined. The design of the user-interface should minimalize this need. For input, one could think of format specification or specification of the expected hierarchical structure of a file. Also a subfile structure can be imposed on an already existing file. Apart from the specification of new structures a notation should be available which answers the following requirements:

- One should be able to compose data structures from existing ones.
- One should be able to access substructures, i.e., indexing should be possible.

(For an example of such a system, see Adèr & van der Veer, 1982.) Throughput-mode also may require declarations: it should be possible to declare missing values or name new variables. During throughput-mode computation or recoding is performed sometimes altering or extending the internal file(s). We can formulate a variety of desires here: the existing user languages are rather backward here compared to higher programming languages. Some idea's may be found in 4.4.

The same requirements mentioned above apply: composition or indexing of newly formed structures should be possible.
Finally, we consider the meaning of data manipulation in output-mode. What are the user's objectives when manipulating output? We consider three kinds of actions the user is likely to perform:

*) In most existing packages handling more than one system file is impossible.

- inspection
- isolation of some parts to send to an output-device
- isolation of information, stripped of irrelevant fringe like headings and edges of tables, to be used in subsequent analyses.

Essentially, output should allow manipulation like any other data structure. The user should be able to skip through it easily. It should be addressable to be send to the printer or to be used as input to a subsequent analysis. A notational system should be present to enable this.

4.3.3. Administrative actions

In input-mode to fetch external files should be possible, as in output-mode to store files externally. In throughput-mode information about names or composition can be gained about the internal data structures. The user can ajust all kinds of parameters in this mode (e.g., intensity levels).

Additionally, it would be convenient to him to be able to indicate whether a task should be performed interactively or in batch.

4.4. Desirable features of the user-language

In throughput-mode all sorts of computations and recoding should be possible. We could formulate a whole scala of whishes on this point since facilities in the existing user languages seem rather primitive compared to the more sofisticated constructs existing in programming languages.

We mention only a few possibilities. Undoubtedly, the list can be extended.

The user should be able

1. to handle a list of variable-names as a datastructure. Think of a repeat-instruction of the following form:

   ```
   for dummyname in list do
       <statement with dummyname as a parameter>
   od.
   ```

   Also, indexing of such a list should be made possible (i.e. one should be able to speak of the 5th element of list).

2. to perform computations not only on variables but also on "cases", or: columns and rows should be accessible for computation, or: not only the internal file but also the transposed one should be manipulatable, or: indexing should be as flexible as possible.
3. to define rows of numbers by name. In this way recoding and indexing can be made easier. For instance:
   row r := (5,4,3,2,1);
   r **ind** cormat;

4.4.1. Suggestions concerning Macro's

Users with no programming experience will have difficulties in understanding the difference between the definition of a macro and the calling of it. To make this more transparant, it may be conceptionally helpful to have a keyword "PASSIVE" that forces the package into specification-mode. From thereon instructions can be entered that are not executed but combined into a macro definition. The last step will be the specification of the name and the parameters of the macro. Another keyword "ACTIVE" indicates that from then on keywords (eventually selfdefined macro names) should be executed immediately. When entering the package, ACTIVE is on.

Elegance of design and transparancy to the user would be served if the expressions used in the macro would be the same as those in the user language. Unfortunately, there are two valid reasons to prevent this. Firstly, inside the macro the use of conditionals should be made possible; for example, a question to be asked whenever the macro is activated might be stored with an option. In this way, queries can be defined by the user. Once a macro is activated, options indicated by the user set the conditionals and determine the flow of control.

Secondly, one can hardly escape the allowance of procedures written in a higher level programming language like FORTRAN or PASCAL to be used as macro's. This also entails that it should be possible to introduce the the text of the procedure from external file, i.e. not interactively.

We conclude this section with a few suggestions that may augment the usefulnêss of macro's.

Once a macro is defined one should be able to use it inside other macro's. As a consequence also recursive use should be considered. Of course this easier to state than to implement: termination of execution should be guaranteed. The issue asks for a careful analysis and eventually, the formulation of well defined restrictions.

Apart from the "usual" parameters like reals or integers that control the execution, also other data structures may function as parameters: a row of strings that answer the questions of a query sequence, or output produced previously. As in the case of recursion mentioned above, it may be necessary to restrict the use of macro's as parameters: the possibility of unwanted side-effects as well as the conceptional complexity to the user, necessitate careful consideration of this option.

5. OTHER ASPECTS

5.1. Response time, portability and machine dependency

There are three viewpoints, concerning these objectives, as described in the introduction: that of a user of a package, that of it's developer and finally that of the distributor.

From the user's viewpoint fast response times are of foremost importance. This will be dependent on the "matching" between package software and machine hardware, i.e. from the implementation of the package on the machine. Fastest response times will be realized when the package is written in assembler or in machine code.

This has an obvious drawback: the package will become nearly nontransportable, i.e. bound to the development site. Though a user will find no problem with this, the developer will also have other aims. He will want to lease (or sell) his product to other institutions. Thus for him portability is of the greatest interest. To obtain this goal the package should be machine independent to a large extent. Therefore the package should be written in a well-known higher programming language (i.e. like FORTRAN or PASCAL).

Unavoidably, though, some things are strongly dependent on the used machine(s), especially on the word length, but also routines for IO-handling, bit-manipulation, making graphic applications for the video display or the plotter, etc. The distributor would like to have those operations concentrated in as few routines as possible, to facilitate easy conversion of the package for his specific machine.

5.2. Reliability and maintenance

The package should be reliable in two ways. Firstly, there should be numerical reliability. For any computation a stable and accurate algorithm should be used, while the results should be given in a correct format: a result that is numerically accurate to its second digit only, should not be given in three digits, though a result, accurate in eight digits, may be given in four.

Secondly, the package should be proof against injudicious use, as has been noted in 3.3. Therefore it should have been extensively tested, both at the original development site and in the later working situation. Several test sets with and without pathological data should be available for this purpose. It should be easy to correct errors that are detected after the release of the package. Therefore the package should be easy to maintain and to patch. This implicates a modular structure, a richly documented source code and good documentation, the latter two available to the distributors. The users and the distributors should be able to defer to the developer for maintenance and quality-control. For patching it would be an advantage, if the distributor can fit in his own subprograms. This is one more argument in favour of writing the program in a well-known language and for extensive documentation.

5.3. Interfaces

In this article we recognize two different types of interface: the user interface, as described in chapter 4, and the package interfaces. The latter consist of software, i.e. special routines for purpose of making a link between the package and some external feature. They can be divided into three main groups:

1. The user-package interface, consisting of all routines that handle the communication between user and package, i.e. question asking routines, text evaluation, error messages, help facilities, etc. This is quite different from the user interface in ch. 4 !

2. The package-package interfaces, consisting of the linking routines to other packages. Of special importance are the routines that handle conversion of the respective package-internal-files (system files). Also there might be routines to call the other package(s) directly, if the operating system allows for such an action (as in RSX-11D). Two examples are P-STAT (to do explorative data analysis) and ISPAHAN (Gelsema, 1981).
3. package-library interfaces. These can be subdivided into two: one to the system library, and some to external software.
   a) As noted in paragraph 5.1 all operations that are dependent on the used machine should be concentrated in but a few routines, together forming the interface (though one might find it easier to seperate different tasks in different -sub- interfaces, i.e. one for plotting, one for bit-manipulation, and so on).
   b) For numerical routines it might be an advantage to obtain these from an external library, for instance Nag or IMSL. This way one may expect to have the most up-to-date algorithms available. Besides, one does not need a new release of the statistical package, when a new numerical routine has been developed. On the other hand the distributor -and indirectly the user- now is dependent on two developers, for availability, for compatibility, as well as for maintenance. To buy a package for statistical computing would mean to be obliged to buy a certain (numerical) library too.
   Therefore in our view the developer should include in his statistical package an interface to all external software. Any routine from an external library should be embedded in one package routine, so as to facilitate adaptation to new releases of the library. In this way it is also possible to replace such a routine by one, written by the distributor. Moreover, it should be possible to obtain the package as being completely self-supporting, i.e. it should have it's own routines for all operations, included in an external library. Otherwise it would be vulnerable in its dependence on the availability - and maintenance - of such a library.

5.4. Processing modes and batch jobs

There are three modes in which a package may process the given commands: as an interpreter, a translator or a compiler. In the first case it evaluates and executes one given line/command at a time. In the second case it translates one or more commands into a set of instructions in another higher language. In the third case it collects all commands first and then compiles and executes them.

Most CSPs process the commands in the interpreter mode: it is easier to implement and more flexible to use. The compiler mode makes macro building very easy, but requires more insight from programmer and user. In the translator mode it is easy to create "batch jobs".

Though no mention has been made of it as yet, it should be worthwile to have the option for batch processing. Especially when large (core memory, CPU time) or special (coloured ink plots, floppy disk output) requirements of system utilities will be made, it might be advisable not to do the analyses in an interactive way. It will be better to place a complete "job" in the input stream for batch processing. When the user is given this options, he also should be given the possibility to work on interactively, while his job is executing.

5.5. Protection against system crashes

The operational system that never crashes still has to be invented. In this section we will discuss some ideas about how to make the disadvantage-ous consequences of a crash as small as possible. We will suppose that during an interactive session a workfile exists which contains the actual information. Examples of such information are computed data, history, intermediate results and statements (when the package is in translation or compiler mode (cf. section 5.4)). During the session this file is continuously altered. To protect the user against losing the workfile it has to reside on external memory (disc, tape). The main disadvantage of this system is the relatively low speed of the data transports between internal and external memories, even if these transports are buffered. Therefore the costs of extra computing time must be weighed against wages that must be paid for the extra work caused by a crash, taking in account, of course, the probability of a system crash. It seems advisable to offer protection to inexperienced users with non-protection as an option. For experienced users the other way around is preferable, since protection will put a large claim on system resources!

APPENDIX A. FORMALIZED DISCRIPTION OF A PROTECTION SYSTEM

Consider a system $S = (C, P, A, L)$, in which C is a set of objects, called cases, P is the powerset of C, A a set of operations, called analyses. There exist two special subsets of A, AC and AE, $A = AC \cup AE$, $AC \cap AE = \emptyset$. Elements of AC are called confirmatory analyses, elements of AE are called exploratory analyses.

L = {"USED", "NOT USED"} is a set of two attributes. At any time, all elements of C are assigned one of those two attributes, so $C = CU \cup CN$, $CU \cap CN = \emptyset$ and CU contains elements with the attributes "USED", CN with elements with the attribute "NOT USED". The user can apply elements of A to P, possibly causing a change of attribute to some elements of C.

During a session the following steps are performed concerning S:

Initialyse. All elements of C are attributed the label "NOT USED", so $C = CN$, $CU = \emptyset$.

Choose. $a \in A$, $p \in P$ are chosen by the user. Then, a is applied to p.

Confirmative Check. If $a \in AC$ and $p \in P$, then:
- a. If $p \cap CU = \emptyset$ then all elements recieve the label "USED" afterwards. CU becomes $CU \cup p$, CN becomes $C - p$.
- b. If $p \cap CU \neq \emptyset$ then an errormessage is given and no action is taken.

Exploratory Check. If $a \in AE$ and $p \in P$ then CU becomes $CU \cup p$, CN becomes $C - p$.

Repeat or Finish. The user may direct control to the second step if he wishes so or end here.

Remarks:

1. The user has influence on the second step ("Choose") and the last step ("Repeat or Finish") only. However, there may be some function at his disposal that enables him to put elements of CU into CN at will, before the step "Choose" is taken. Since the CSP is not guarded against such action, no warnings are given in a case like that: these changes are for the responsability of the user.

2. a AC can consist of a sequence of confirmatory analyses of which the increase of the level of significance can be controlled.

REFERENCES

ADèR, H.J. & D.J. KUIK (1983). Talking Statistics without using bad language (user desires in Conversational Statistical Packages), In: Symposium bundel, Symposium Statistical Software, Utrecht, p.171-183.

ADèR, H.J. & G.C. VAN DER VEER (1982). Prelude-composing in Algol 68: Applications in Dataminipulation and User Languages, In: COMPSTAT 1982, part II (supplement): Short Communications Summaries of Posters, Edited by H. Caussinus, P. Ettinger & J.R. Mathieu, Physica-Verlag, Vienna.

ADèR, H.J., D.J. KUIK & G.C. VAN DER VEER (1984). User-aims while using Conversational Statistical Packages, In: COMPSTAT 1984, Summaries of Short Communications and Posters, General Computing Center, Czecho-slovak Academy of Science.

CABLE, D. & B. ROWE (1984). Software for statistical and survey analysis. Comp. Stat. & Data Analysis Vol. 2, 1, p. 81-92.

CHAMBERS, J.M. (1982). Analytical Computing: Its Nature and Needs. In: Proceedings in Computational Statistics, Part I. Physica Verlag, Vienna, p.22-29.

DAVIS, R (1983). User error or Computer error ? Observations on a Statistical Package. Int J. Man-Machine Studies 19, p.359-376.

DoD, DEPARTMENT OF DEFENCE (1978). Department of Defense STEELMAN require-ments for high-order computer languages.

DoD, DEPARTMENT OF DEFENCE (1980). Department of Defense Requirements for ADA Programming Support Environment "STONEMAN".

DIXON, W.J., M. B. BROWN, L. ENGELMAN, J.W. FRANE, M.A. HILL, R.I. JENNRICH & J.D. TOPOREK (1981). BMDP Statistical Software Manual, University of California Press, University of California.

FRANCIS, I. (1977). A Comparitive Review of Statistical Software. Interna-tional Association for Statistical Computing, New Delhi.

FRANCIS, I. (1982). A survey of statistical Software. Comp. Stat. & Data Analysis Vol I, 1, p.17-27.

GALE, A. & D. PREGIBON (1984). Constructing an Expert System for Data Analysis by Working Examples, In: COMPSTAT 1984, Proceedings in Computational Statistics, Physica-Verlag, Vienna, p.227-235.

GELSEMA. E.S. (1981). ISPAHAN, Users Manual (4th edition). Department of Medical Informatics, Free University, Amsterdam.

GILFOIL, D.M. (1981). Warming up to Computers: a study of the cognitive and affective interaction over time. In: Proceedings of Human factors in Computer systems, Gaithersburg, Maryland, p.245-250.

HEXT, G.R. (1982). A Comparison of Types of Database Systems Used in Statistical Work. In: Proceedings in Computational Statistics, Physica Verlag, Vienna, I, p.272-277.

ICHBIAH, J.D., J.G.P. BARNES, J.C. HELIARD, B. KRIEG-BRÜCKNER & B.A. WICHMANN (1979). Rationale for the Design of the ADA Programming Language, ACM, SIGPLAN Notices Vol. 14, 6.

KERNIGHAN B.W. & J.R. MASHEY (1981). The Unix Programming Environment. In: Tutorial: Software Development Environments, T. Wasserman (Ed.), IEEE Computer Society.

KOSTER, C.H.A. (1979). User Languages and Application Languages. Informatica / Computer Graphics, Faculty of Science, Nijmegen University, The Netherlands.

KUIK, D.J. & B.F.SCHRIEVER (1984). Statistics and Conversational Packages, In: COMPSTAT 1984, Summaries of Short Communications and Posters, General Computing Center, Czechoslovak Academy of Science.

KUIK, D.J. & A. HASMAN (1983). QUESTOR, a Conversational Statistical Package. In: MEDINFO '83. Proceedings of the Fourth World Conference on Medical Informatics, p.939-942.

KUIK, D.J. (1984). QUESTOR, a Conversational Preprocessor to BMDP. In: COMPSTAT 1984, Proceedings in Computational Statistics, Physica-Verlag, Vienna, p.329-334.

LAURO,N & G. SERIO (1982). Criteria for evaluating and comparing Statistical Software: a Multidimensional Data Analysis Approach, Statistical Software Newsletter, Vol 8, nr 3, p.102-119.

MOLENAAR, I.W. (1984). Behavioral Studies of the Software User, Comp. Stat. & Data Analysis Vol. 2, 1, p.1-12.

MORAN, T.P. (1981). The Command Language Grammar: a representation for the user interface of interactive computeer systems. Int. J. Man-Machine Studies 15, p.3-50.

NIENHUIS, A. (1983). _On the Design of an Editor for the B Programming Language_, report IW 248/83, Mathematisch Centrum, Amsterdam.

PEMBERTON, S. (1984). The B Programming Language and Environment, In: _CWI Newsletter_ 1, p.2-14.

PREGIBON, D. & A. GALE, 1984. REX: an Expert System for Regression Analysis. In: _COMPSTAT 1984, Proceedings in Computational Statistics_, Physica-Verlag, Vienna, p.242-248.

REID, A., J.S. LENNON & J.S. KNOWLES (1980). _Report on a Survey of Interactive Statistical Packages in British Universities and Polytechnics_, Inter University Software Committee, Aberdeen University Computing Center.

RSX-11D _Documentation Directory_. DEC-11-OXUGA-D-D. Digital Software Documentation.

SAJA, A.D. (1985). The Cognitive Model: An Approach to Designing the Human-Computer Interface. ACM, _SIGCHI Bulletin_ Vol. 6, 3, p.36-40.

SHAW, M., P. HILFINGER & W.A. WULF (1978). TARTAN-Language Design for the Ironman requirements: Reference Manual, ACM, _SIGPLAN Notices_, Vol. 13, 9, p.36-58.

STAP (1980). _Statistical Appendix_. User manual, Stichting Academisch Rekencentrum Amsterdam, Amsterdam.

TUKEY, J.W. (1977). _Exploratory Data Analysis_. Addison-Wesley, Reading, Mass.

VAN APELDOORN, J.H.F (Ed.) (1983). _Man and Information Technology; towards friendlier systems_, STT Publications 38, Delft University Press.

VELLEMAN, P.F. & D.C. HOAGLIN (1981). _Applications, Basics and Computing of Exploratory Data Analysis_, Duxbury Press, Boston, Mass.

WILKE, H. (1982). Evaluation of Statistical Software Based on Empirical User Research. In: _Proceedings in Computational Statistics_, Part I. Physica-Verlag, Vienna, p.442-446.

YESTINGSMEIER, J. (1984). Human Factors Considerations in development of Interactive Software, ACM, _SIGCHI bulletin_ Vol. 16, 1, p.24-27.