CANADIAN THESES ON MICROFICHE

THÈSES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

NL 339 (r. 85/09)

C 315-20316-1

National Library of Canada | Bibliothèque nationale du Canada | CANADIAN THESES ON MICROFICHE | THÈSES CANADIENNES SUR MICROFICHE

68223

NAME OF AUTHOR/*NOM DE L'AUTEUR* FREDERICK J. MARTIN

TITLE OF THESIS/*TITRE DE LA THÈSE* TOWARDS THE AUTOMATED EVALUATION OF INTERACTIVE SYSTEMS: A QUALITY-ASSESSMENT APPROACH

UNIVERSITY/*UNIVERSITÉ* SIMON FRASER UNIVERSITY

DEGREE FOR WHICH THESIS WAS PRESENTED/*GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* MASTER OF SCIENCE

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE GRADE*

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* NICK CERCONE

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* Oct. 2, 1984 SIGNED/*SIGNÉ*

PERMANENT ADDRESS/*RÉSIDENCE FIXÉ*

[redacted]

NL-91 (3-74)

TOWARDS THE AUTOMATED EVALUATION OF INTERACTIVE SYSTEMS:

A Quality-Assessment Approach.

by

Frederick J. Martin

B.A. (Hons.), Simon Fraser University, 1982

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

Computing Science

SIMON FRASER UNIVERSITY

July 1984

APPROVAL

Name: Frederick J. Martin

Degree: Master of Science

Title of Thesis: Towards the Automated Evaluation of Interactive Systems:

A Quality-assessment Approach.

Examining Committee:

Chairperson: Art Liestman

Nick Cercone,
Senior Supervisor

Brian Funt

Bill Turnbull

Dr. Shahid Hussain
External Examiner,
Vice-president, Planning and Services,
Microtel Pacific Research Ltd.

Date Approved: 84.8.15

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

TOWARDS THE AUTOMATED EVALUATION OF INTERACTIVE SYSTEMS:
A QUALITY-ASSESSMENT APPROACH.

Author: _____
(signature)

Frederick J. Martin
(name)

October 2, 1984
(date)

ABSTRACT

A system for performing automated quality-assessment evaluation of interactive computer systems is proposed and an implementation is discussed. After the need for evaluation is reviewed and shortcomings of existing evaluation methods are cited, the proposed alternative method is introduced. A major feature of the proposed system is the automated recording of user characteristics and user perceptions of system quality. A LISP implementation of the evaluation system is introduced and described. Test results are given, demonstrating the use of the system on a small-scale Expert System application. Conclusions are presented regarding the efficacy of the method and suggestions for future work are outlined.

ACKNOWLEDGEMENTS

My Senior Supervisor, Dr Nick Cercone, was the key motivating force behind this project. He has been available and helpful throughout my graduate-student experience and has always provided encouragement at precisely the times when encouragement was most needed. The other members of my examining committee, Doctors Shahid Hussain, Brian Funt, and Bill Turnbull, were generous with their time, and both fair and helpful with their criticisms and suggestions.

Dr. Bill Richards first proposed that an automated evaluation component be built into SFU's Automated Academic Advisor as a result of his assessment work on TELIDON. He helped focus my awareness of the need for this research, and suggested the 'spontaneous user comment' feature which appears in the program.

Ed Bryant gave freely of his time and expertise in introducing me to the wonders of UNIX, and generously sought to devise a reasonable interface between the available implementations of Prolog and LISP.

I am grateful to my colleagues and the administration at BCIT who granted me leave from some of my duties there so that I could work on this project.

Finally, and by no means least, I want to acknowledge the support of my wife Jacqueline, which has never wavered despite the considerable sacrifices involved in being married to a computing science student.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In general, techniques and procedures for systems evaluation are still somewhat primitive and ad hoc [Landau, 1982]. This is particularly so for the relatively new field of "expert" systems, which now tend to be designed with a major focus only on advancing the state of the art of system capabilities [Michaelsen, 1983]. Furthermore, systems evaluation is usually interpreted as meaning performance measurement rather than quality assessment, an interpretation which is becoming less adequate as the field evolves and matures [Carroll and Rosson, 1984].

When system evaluations are done poorly or are not done at all, there arise many possibilities for errors and inefficiencies. One frequent outcome of weak evaluation is that the system becomes a 'working non-solution' to the problem areas it addresses [Bair, 1982]. Indeed, Richards and Larimer [1984] cite the Canadian Telidon project as an unfortunate example of this phenomenon.

The present work describes an evaluation tool for interactive systems which is extensively automated, easy to implement, relatively portable, reasonably standardized, yet easily modifiable to address the unique needs of specific applications. The approach focuses strongly on human factors and user perceptions of system quality and system effectiveness, important areas which tend to receive minimal attention in the existing literature on systems evaluation. While the present project was implemented as an adjunct to artificial intelligence (AI) Expert Systems, there is no reason why the approach could not be applied to a wide variety of interactive systems.

1.1. Who needs to evaluate a Computer System?

There are various categories of people who might want to evaluate a system. These include: designers, who need to know how well their design goals were met by the actual system implementation; experts who have provided their knowledge want to know how well it has been captured in the system; prospective buyers must decide whether the system meets their needs better than other competing products do; accrediting bodies and professional societies must consider whether to permit practitioners to use a system; society as a whole must decide whether to condone use of some systems, such as expert systems; programmers want to know whether a particular system is easy to implement, modify, and extend; researchers will wonder whether a system incorporates novel features of theoretical interest, or reflects only well-known standard design approaches; finally, and of very great importance, users at various levels of proficiency want to know how well a system fits their needs.

1.2. What is Evaluation?

Clearly, with so many categories of people having a potential interest in evaluating a system, it is likely that the term 'evaluation' will have more than one meaning, depending on the evaluator's perspective. In a 1980 panel discussion of Software Metrics, David Weintraub is quoted as saying:

> "The implication of having metrics is that somebody somewhere wants to measure something, and my argument is, everybody involved in software thinks it's something different, and I think that everybody is right." [Perlis et al, 1981].

Among the well-known criteria which have been used for system and program evaluation are: program length, measured in lines of source code; program length, measured in characters of source code; frequency of occurrence of certain programming constructs, such as 'GOTO' or 'DO WHILE'; extent of indentation in source code; number and type of comments; modularity measures, such as size and number of modules; run times of batch systems; response times of interactive systems, and formal mathematical analysis of

algorithm efficiency and correctness.

1.2.1. Some Problems with a Typical Evaluation Criterion.

As might be expected, each of these measures has advantages and disadvantages. For instance, counting the lines or characters in the source code of two programs which purport to do the same task appears to be a useful tool for deciding which program is 'better'. The method is easily implemented, especially if automated, and provides a simple quantitative measure. However, there are numerous confounding issues. The smaller program may cram several statements onto each line, or it may omit comments, or it may use short variable names. Each of these strategies to reduce size would generally be seen as also reducing program quality. Is a small program really better? In some contexts, a larger program could be perceived by some people as a sign of high programmer productivity, but would merely indicate high costs to other people. A small elegant program, the source of great pride to its author, might be seen as an unreadable monster by a manager or maintenance programmer. Again, the perceived merits of the program depend on the perspective of the evaluator.

Furthermore, two programs which accomplish the same task but differ greatly in size may in large part reflect only the characteristics of different programming languages, rather than the competence of the programmers. Rephrasing the question of whether one program is better than the other to one that asks which language is better does not help, since a sparse language such as APL might be perceived as good by someone writing a program, while a verbose but self-documenting language such as COBOL may be perceived as good by someone who must read or maintain programs.

Finally, and of major importance, is the fact that the measure of program size, which may seem so important to programmers, language designers, and their managers, is probably of no interest whatsoever to the end user, who will have a completely different set of criteria for program evaluation. Although counting lines of code might yet be a useful

evaluation tool, clearly it must be used and interpreted with care. Similar comments could be made about the various other commonly used evaluation techniques. One feature which most of them lack, however, is the perspective of the end user.

1.2.2. Universal Evaluation Metrics Are Unlikely.

Basili [1981] categorizes six (partially overlapping) kinds of data of interest to the would-be evaluator: people factors, such as experience, morale, and capabilities; problem factors, such as the state of the problem definition, importance of the problem, and constraints placed on the solution; process factors, such as top-down design, programming language(s) used, and structured programming; product factors, including correctness, efficiency, and portability of code; resource factors, such as deadlines, budget and turnaround times; and tool factors, including database systems, automatic flowcharters, and the like. While Basili's list may or may not be comprehensive, it serves to show that system evaluation is a complex subject. It suggests also that the task of evaluating a system may always be somewhat ad hoc. If seemingly straightforward metrics, such as counting lines of code, are fraught with ambiguity, what hope is there for standardization on such inherently ambiguous issues as 'people factors'?

In contrast to the attempts at quantitative evaluation implied above is the observation that diffusion of a new system (user acceptance) is often based on the system's evaluation by near-peers [Landau et al, 1982]. The point is that a system will often succeed or fail, based on its word-of-mouth reputation spreading among users. In this view, a user will be influenced to try an unfamiliar system, not by formal evaluation results but by informal reports obtained from close colleagues who have used the system. Unfortunately, however, the user's assessment of a system may be very different from the designer's. As Damodaran and Eason [1981] put it:

> "Computer systems rise or fall on the ability and willingness of users to make effective use of them. Systems designers may feel they have created a perfect system for the users but if the users are not also convinced, their efforts may have been wasted."

The importance of such 'naive evaluation' should not be underestimated.

Finally, to further confound the problem of attempting to define evaluation, is the issue of temporal factors. Even if a set of metrics for system evaluation could be agreed upon, a system which scored well at one time might be rated much lower at another time, or vice versa. "Initial user desire for a simple system with few commands... gives way to dissatisfaction... and a demand for extended capabilities." [Landau et al, 1982]. Similarly, yesterday's state-of-the-art design methodology, such as program flowcharting, becomes today's quaint anachronism [Schneiderman et al, 1977]; new hardware capabilities make a previously satisfactory system appear unacceptable, and so on.

From this brief overview of the nature of evaluation, a few points should emerge. Firstly, evaluation quite legitimately means different things to different people. Second, evaluation may mean different things to the same person in different contexts. Third, the results of evaluation of the same system may legitimately differ over time. Fourth, the prospect of obtaining a comprehensive list of immutable, universal system evaluation criteria is remote. Fifth, evaluation is desirable, despite the various problems associated with it. Sixth, the evaluation process can legitimately address both subjective and objective issues. Finally, and of great importance to this thesis, is the point that the end user should have an active part to play in any formal system evaluation, since the users' informal 'word-of-mouth' evaluations often determine whether a system will succeed or fail in practical terms.

1.3. Automating the Evaluation Process

Given that evaluation is both difficult and desirable, are there gains to be realized by automating the evaluation process? Clearly, some components are easily automated, such as counting lines of code. Other components, such as gathering users' opinions, may seem to resist automation. Basili [1981] suggests that evaluation data-collection takes four basic formats: reporting forms; interviews; automatic collection using the computer system, and

automated analysis routines. The present work will show that it is feasible to automate much more of the data collection process than has generally been the case, including elements which would normally be handled through reporting forms or interviews.

In this thesis, a collection of automated data gathering procedures is developed to address the tasks of inter-system and intra-system evaluation. These procedures are intended to provide at least minimal evaluation support for most systems under most circumstances, but are easily modifiable and extendible to handle unusual situations. For systems which use a 'throw-away code' successive prototype style of development, the procedures can be activated in even the earliest prototypes, and can gather information which will affect future development decisions. The procedures remain in place throughout the application system's life cycle, providing guidance for development, maintenance, expansion, and eventual retirement of the system.

Most of the literature on computer system evaluation concentrates on issues relating to programming or machine efficiency and program correctness. Borovits and Neumann [1979], Ferrari [1978], and Ferrari and Spadoni [1981] typify this orientation. Perhaps, however, the computing community's tendency to measure 'internal' computing-related factors almost exclusively, rather than 'external' user-perceptions causes many, if not most, systems to be less effective than they could be. According to Carroll and Rosson [1984], this view is becoming more widely held:

> "But increasingly there is an awareness that mere performance is not enough, that it is not the same thing as quality. As computers become tools and toys for everyone, their ability to adapt to human needs and propensities becomes the principal determinant of real system quality.... In the next few years, the study of system quality should be one of the most active areas in computer science research."

The present work, while addressing some internal issues, focuses more directly on human factors, particularly those involving user perceptions of system quality. In particular, the process of interviewing users is extensively automated, leading to several possible advantages. Kaplan and Saccuzzo [1982] cite the following specific advantages of interac-

tive interviewing or 'testing': excellence of standardization; sequential administration, individually tailored; precision of timing responses; release of human testers for other duties; patience; and control of bias. Certainly, one more advantage is that the interviews will actually take place. Presently, interview plans are all too easily dropped under the pressure of budget and deadline constraints. Automated interviews can be done relatively cheaply and quickly. Haynes and Wilson [1979] point out that a

> "...very important attribute of computer assessment is that it may reduce the error variance associated with clinical interviews. Because the method of presentation is constant across subjects, variance attributable to interviewer fatigue, nonverbal cues, reactions to the subjects' responses, or interviewer bias might be reduced. Another major advantage is that analysis of data would be facilitated by utilizing software to summarize and analyze the subjects' responses."

The same authors report reassuring results concerning the validity of data obtained from automated assessment and the acceptability of the method to 'clients'. They cite:

> "high agreement coefficients between information elicited in the computer interviews and information elicited during psychiatric interviews and questionnaires. During the computer interview, however, alcoholics reported significantly greater alcoholic ingestion than during the psychiatric interview. This would suggest that, compared to personal interviews, computer interviews might provide more valid assessment data on such sensitive issues.... The subjects also rated the computer as highly acceptable."

Haynes and Wilson are of course reporting on studies done in a clinical setting, but there seems little reason to believe that their results are not generalizable to other situations. Miller [1970] expresses the extent to which users are willing to drop their defenses when dealing with computers:

> "I find it remarkable that an intelligent college student will let a machine tell him repeatedly that he is wrong without losing either heart or face; if a human experimenter told him the same thing, he would seethe with indignation."

Certainly Weizenbaum's classic experience with ELIZA strongly suggests that people are perfectly willing to deal with computerized interviewers [Weizenbaum, 1976].

By way of contrast, it is all too easy to imagine a user being interviewed by a system designer who becomes defensive in the face of any criticism of the system, who becomes sidetracked or forgets to ask key questions, who distorts user responses for any number of

reasons, or who just lacks the skills needed for effective interviewing. Such problems are well documented in the literature on interviewing. Sundberg [1977] cites many studies demonstrating problems of interviewer bias and inter-interviewer reliability, most of which disappear if the interviewer is a machine.

1.4. Summary of the Motivation for this Project.

Systems evaluation is desirable for many reasons, such as to guide development, to assess competing products, to avoid producing 'working non-solutions', to sense when maintenance is appropriate, and to help recognize when a system has reached the end of its useful life. Virtually everyone involved in any way with a system has a potential interest in one or more aspects of its evaluation, such as cost, efficiency, effectiveness, user-friendliness, or impact on society.

To date, most work in software evaluation focuses on internal, computing-related issues, such as program efficiency or maintainability. However a system will be a failure if it is not accepted by its user community, no matter how elegant its algorithms, how maintainable its source code, nor how efficient its use of resources. Therefore, the present work focuses strongly on external human-oriented issues, such as users' perceptions of system quality and effectiveness. It provides a way to gather information on such factors with relative ease and consistency.

CHAPTER 2

AUTOMATED EVALUATION GOALS: WHAT TO EVALUATE AND WHY?

2.1. The Ideal System Evaluator.

An ideal evaluation program will meet the needs of all the categories of people outlined in section 1.1. It will permit evaluation of a system at all stages of its life cycle, and will facilitate comparison of different systems.

Ideally, the criteria used in evaluation should be universally applicable, should be meaningful to at least one category of interested people, should be unambiguous, and should be easy to interpret. Data collected for evaluation purposes should be reliable, comprehensive yet non-redundant, and should be easily and inexpensively collected.

No doubt the reader can add to these lists, but it should already be apparent that an ideal evaluator could only be designed upon a mature, scientific, rigorous theoretical framework. Alas, the state of the art of system evaluation is far from this ideal. Browne and Shaw [1981] recommend that future experimental work in software evaluation be based upon traditional principles of science, where first considerations are principles which are invariant across all software. Unfortunately, no-one is yet delineating such principles. These authors correctly note that most present techniques of description, measurement, and evaluation are ad hoc:

> "Most of the analysis and data collection techniques haven't been generalized beyond the local system for which they were developed."

Denicoff and Grafton [1981] add that

> "...quantitative evaluation techniques are necessary for the evolution of software from an art form to an engineering discipline and eventually to qualifying as a science."

Such comments suggest that the ideal evaluator will not soon be realized.

2.2. Categories of Information to be Measured.

An evaluation system which meets the needs of some people may be inadequate for others. The following list is a collection of possible aspects of a system which could be worth evaluating: ease of use; development cost; cost to use; cost to maintain; cost to enhance; ease of maintenance; extensibility; user training requirements; system power and features; user friendliness; documentation quality and quantity; execution speed; memory utilization, and inherent usefulness of the application. Each of these terms may have different meanings and imply different criteria to different people, depending on their orientation.

Some data can be gathered automatically by an evaluator, with no requirement for a user to take any specific action, possibly even without the user's knowledge (although this of course raises ethical issues). Some possible topics for automatic monitoring include: delay/response times; number of queries; type of queries; number of ill-formed inputs; site, including terminal type; environmental features; such as temperature, lighting, and the like, and time of day.

Basili's list (Section 1.2.2) of six factors of interest to an evaluator serves as a useful starting point in attempting to identify what an automated evaluator should do. His heading 'people factors' may offer the most scope for an automated evaluation system, precisely because it is an area which traditionally has not received much attention and because it includes many of the issues of concern to an end-user.

If an automated evaluation system is activated as a constituent of an interactive application system, it can potentially serve all the categories of people identified in Section 1.1. It can serve those who deal directly with the system, such as designers, programmers and end-users, by eliciting their comments about the application system, by monitoring their interactions, and by gathering data about them. The system can also serve those who deal with it at arm's length, such as professional societies or prospective buyers, by provid-

ing information in a standardized form to help them in their decision making.

Assuming that evaluation logic can be interfaced with an application system so that the former is always available whenever the latter is running, the evaluator will potentially be able to gather data from each and every person who interacts with the application system.

Some types of 'people factor' data which could be gathered routinely are: user profiles; user competence or mastery of the system under evaluation; user opinions on various system-specific issues; user opinions of the system's strong and weak points; effects of external factors such as time-of-day upon user performance; user estimates of the cost to them of various system problems, or conversely, of the value to them of proposed changes, and user opinions of the inherent value of a system. The potential value of much of this information should be obvious, and I shall not comment extensively on how it could be used by various categories of people. However, some remarks on user profile information may be useful.

2.2.1. Comments on User Self-reports of Attitudes

Psychologists have developed a variety of questionnaires and tests which purport to identify or measure a person's traits, attitudes, personality, and so forth [Kaplan and Saccuzzo, 1982, Aiken, 1982]. Naturally, these tools focus strongly on the individual person, since they are used to help make decisions about an individual's life. They are used to assess factors such as degree of pathology, appropriateness of behavioral counselling, or suitability for employment. Many issues are raised when questionnaires are used under such conditions, including issues of ethics, test validity, and reliability of subject self-reporting. Indeed, Cattell [1973] strongly challenges the widely-accepted practice of self-reporting on the grounds that it is too vulnerable to faking.

The present work uses attitudinal self-reporting in a different way, with an emphasis upon the computer system under study and upon overall user group behavior

rather than upon the individual user. The system requires each user to provide a name or identifier by which the system can recognize the individual in the future. Under most circumstances, a designer should ensure that real names are not given, and that the user's anonymity is preserved with respect to other users, system designers, project managers, and so forth. If users do not have assurance of anonymity, there is every reason to believe that they could give distorted answers to questions.

The intent of gathering user self-reports is to seek factors which influence user perceptions of a system and user behavior with respect to the system. One can advance many intuitively-appealing hypotheses in these areas, such as: "User education level will correlate positively with user effectiveness", or "User mood will affect user opinions of the system's value". Without gathering information on user education, mood, and the like, there is no way of testing such hypotheses.

For most situations of interest to the computer system designer, there is no need to determine whether a user's opinions reflect innate traits which make up a 'personality' (a topic of great interest and much debate by psychologists). While a designer might ideally want to 'learn all about' a user by giving various ability tests, intelligence tests, personality tests, and so forth, such detail is clearly inappropriate under most circumstances. The designer will care mainly about how people describe their feelings when using the system, and the extent to which the system influences these feelings and/or the feelings influence the user's performance on the system. Furthermore, the computer user is unlikely to welcome extensive questioning about mood, motives, attitudes, and, so forth in the middle of a 'productive' interactive computer session. Any questioning done should be on a much smaller, less intrusive scale than the typical psychological test battery (the California Psychological Inventory, a typical personality test, contains 480 items).

2.2.2. What User Profile Information to Gather?

Some typical data might include: name; age; sex; user category (eg student, administrator, or faculty member for a university environment); education level; computer-training level; familiarity with system (or similar systems); frequency of computer use; frequency of system use; alone or accompanied; mood or frustration level during system use, and reasons for using the system.

For a particular application, someone must decide whether a user's real name is to be recorded, or some unique code which preserves anonymity. Although this is an interesting issue in its own right, it is not addressed further here. Some of the items listed, such as age and sex, are relatively static and need only be recorded once. Others, such as mood or reasons for using the system, may vary from session to session and should therefore be sampled more than once for each user.

The motivation for measuring all of these people factors is generally prediction, rather than explanation, a motivation which is entirely appropriate for a young science [De Millo and Lipton, 1981]. For instance, it would be valuable to discover whether age or sex differences correlated significantly with error frequency for a system, even if one had no hypothesis as to the underlying cause of the phenomenon. Such information can still have significant practical implications. For instance, Chrysler [1978] found that programmer age and years of formal education were the most significant predictors of project development time. Such a finding could strongly influence project staffing policies.

2.3. Some Possible Approaches to System Evaluation

Among the best-known evaluation techniques are the 'internal' computing-related measures, such as program size and speed. This thesis will not address any such measures which examine source code. After thus eliminating many of these internal measures from discussion, two important standard approaches which remain are the questionnaire and the user interview.

When these approaches are used, they may take various forms. For example, a designer may devise, administer and evaluate responses to questionnaires which address predefined topics or categories. Alternatively, one could use an open-ended, undirected questionnaire approach. The latter could, for instance, ask all users to describe the three best and three worst features of a system. Responses would be analyzed, classified as to type, and evaluated. Another designer might use a mixture of these two approaches. Such a mixture could include directed questions covering areas known or assumed to be relevant to all systems or to a particular system, while leaving room for open-ended questions or spontaneous user comments. Alty and Coombs [1981] report a mixed technique in which users are asked to respond to a verbally administered multiple-choice questionnaire, combined with a Rogerian non-directive interview. After obtaining a user's (forced choice) response to a question, the interviewer would reflect back the answer to the user and await any elaborating remarks. While these methods have the potential for yielding a great deal of interesting data, they are used relatively seldom in practice because of their cost, complexity, and their time-consuming nature. The present project automates these methods in an effort to make them much more practical than has previously been the case.

Another method of evaluation is to have the system 'observe' user behavior, automatically collecting data on response times, utilization of various features, error patterns, and the like. This approach is fairly common and is also used in the present project. I interpret the word 'user' very broadly, as anyone who interacts with the application system. Thus, the evaluation program can record behavior patterns and perceptions of programmers, expert users, novice users, 'normal' end-users, and so on.

Still another method is to have the system prompt users from time to time, asking for user opinions on various areas of interest. For instance, an evaluator could ask several times per session:

"At this point, what do you like best (least) about the system?"

followed by such prompts as:

"Please rate the following aspects of the system - give grades of A+ to F."

This general approach can be used in the present program, but I am not aware of its use elsewhere.

A related approach would allow users to interject comments, opinions and suggestions at any time while interacting with the system [Richards and Larimer, 1984]. This feature also appears in the present program.

Three other standard evaluation methods are worth mentioning. The first, in which a system is run and its behavior compared to the behavior of human experts in case study situations [Michaelsen, 1983], is commendable and should no doubt be part of any expert system's checkout procedures. A second method, in which experts experiment with the system at will and report problems as they are discovered [Michaelsen, 1983], is also worthwhile, and is entirely compatible with the present program. Finally, a system can be run for some time by 'real or realistic' users, whose opinions are then jointly solicited in a 'focus group discussion'. This method allows spontaneity of interaction and the possibility of discovering concensus on issues of interest. It could be used to seek explanations or clarifications of results gathered by the present program.

2.3.1. Some Problems with Evaluation

An inevitable side-effect of incorporating an automated evaluator into an application system, is that the system which will be evaluated is the hybrid thus formed, rather than the 'pure' application system. It seems highly likely that the benefits to be obtained from doing the evaluation will strongly outweigh the disadvantages caused by this phenomenon. Indeed, if the notion of built-in evaluators becomes generally accepted, application systems without them will be perceived as incomplete or abnormal.

A related problem which arises with virtually all forms of evaluation involving overt observation is the Hawthorne effect. This phenomenon was first recorded by Roethlisberger and Dickson [1939] who noted that the mere fact of paying attention to workers in a productivity study caused changes in productivity. No doubt such unwanted confounding effects could be present for an automated system evaluator, especially for users encountering the system for the first time. However, I suggest that such effects would tend to disappear over time, as the presence of the evaluator becomes taken for granted. Such a question is a worthy topic of investigation, but is not explored here.

2.4. Questions Raised or Explored in the Present Work

Reflections of the kind outlined so far led to the formulation of several questions to be explored in this thesis or in future work arising from it. Firstly, can a system participate significantly in its own evaluation? In other words, can evaluation logic be built into an application system in such a way that evaluation information is gathered whenever the system is used? Clearly this question can be answered affirmatively in a trivial sense, by just having the system count the number of users, or some such measure. I mean the question in a deeper sense.

A second, related question asks whether evaluation logic can be usefully broken into general (application-independent) and custom (application-specific) segments? Are there categories of information which should be gathered for any and all application systems? If so, perhaps appropriate programs can be created once to be used universally.

This leads to another related question. Can evaluation logic be transported easily from system to system? Even though the desired information is the same for two systems, there may be technical problems in embedding the same evaluation logic in different systems. These problems could be caused by differences in languages, language dialects, operating systems, and the like. One way to explore the feasibility of transportable evaluation logic is to attempt implementation on different machines, operating systems,

source languages, and application systems. I had hoped to use two such applications for testing the evaluation program, namely an Automated Academic Advisor (AAA) prototype system, written in PROLOG and called SHADOW [Hadley, 1983] and my own small-scale expert system program for microcomputer selection, written in LISP and called Selectmicro [Martin, 1983]. As time passed, it became obvious that the work needed to allow a LISP program to run concurrently and interface with a PROLOG program would be substantial, and should remain outside the scope of the present project. Thus, the main test vehicle for this project became the LISP program Selectmicro. Of course, applying evaluation logic to application programs raises the obvious questions: "How well do these application systems perform? What do their users like and dislike about them? Generally, what are their strengths and weaknesses?" Detailed answers to these questions are not sought in this thesis, except for illustrative purposes to demonstrate the capabilities of the evaluation system.

Another interesting question asks whether the evaluation logic can be 'turned on itself'. What do users think of the evaluation logic which will be developed for the present project? Can such information be gathered by means of the program itself?

If the feasibility of automated evaluation data-gathering can be demonstrated, many questions can be explored and hypotheses tested regarding the effects of changes to a system or to its environment. Sometimes an apparently trivial change can have profound effects, as exemplified by Weinberg's [1971] anecdote concerning the effects of the relocation of a computer centre coffee machine on the pattern of user interactions.

Finally, questions arise as to whether user opinions and question responses match or differ from actual user behaviors as they deal with a system? When users say they want a certain feature added to a system, will they actually use such a feature if it is provided? Which categories of user provide valuable inputs - which categories (if any) are unreliable? These latter questions, and a plethora of related issues belong in the domain of rou-

tine use of the evaluation program, rather than the domain of its development. While they are obvious future extensions to this thesis, they are not included in it.

All of the questions outlined and many related ones can be addressed, and some of them answered definitively, by attempting to implement an automated evaluation package, by testing it, and by reporting on its performance.

CHAPTER 3

DESIGN OF AN AUTOMATED EVALUATION SYSTEM

3.1. Desirable Features of an Automated Evaluation Package

The primary goal of the present work was to devise and demonstrate the feasibility of a data-gathering 'package' which could be easily affixed to application programs. Among the features considered to be appropriate or desirable in the present implementation were:

(1) Relatively little automated interpretation of the data gathered. The evaluator should produce results suitable for analysis by programs such as SPSS. Eventually, as the characteristics of the various data and the specific needs of designers become better understood, more of the data interpretation could be automated.

(2) A substantial kernel of standard (application independent) questions to be asked of users.

(3) Facility to add custom (application specific) questions.

(4) A kernel of standard automated measurements (eg user response times).

(5) Facility to add custom automated measurements.

(6) User ability to enter comments at any time, so that opinions and ideas can be captured before they are forgotten.

(7) Some analysis of user comments as they are received.

(8) Comprehensive logging of user/system dialogue for later human or machine analysis.

(9) Prompt for users' insights on how to fix problems.

(10) Portability - ie transferability to various application systems, ideally across language and operating system boundaries. Such portability would facilitate comparison of application systems, a task which is normally very difficult.

(11) Facility to tally and categorize error messages from the application system to the user.

All of these features have been embodied in the present work, to a greater or lesser extent. Features five and seven are present only in rudimentary form. Feature eight is best implemented directly by the operating system when running under UNIX, and thus becomes redundant. All other features are fully implemented.

3.2. Overview of System Design

The present system is written in a dialect of LISP called Franzlisp. All tests were run from interpreted (not compiled) code running on a VAX computer under UNIX 4.2 bsd at Simon Fraser University (SFU). Source listings for the evaluation system appear in Appendix A.

3.2.1. The Application Designer's Tasks

When an application designer wants to use the evaluation system, he[1] first runs it in stand-alone 'designer mode' in order to set up the appropriate parameters for his needs. Normally, this would be done only once, but it could be done from time to time over the life of an application, in response to changing needs. The main effects of a designer-mode run are to initialize and save the property list of a variable EVALSTUFF, and to modify and save the various questionnaires which are to be given to users. Each of these main effects will now be reviewed briefly.

3.2.1.1. Properties of EVALSTUFF

The designer-mode portion of the evaluation program leads the designer through six 'phases'. In each phase, properties of the variable EVALSTUFF are set. Figure 3.1 shows the format of this property list. At the end of each phase, the designer may backtrack and redo the phase or may move on to the next phase.

[1] Throughout the following discussion, reference is made to application program designers using male pronouns. This only reflects the absence of suitable neuter pronouns in English and is not intended to imply that any or all designers are necessarily male.

Figure 3.1. Format of Evalstuff Property List

EVALSTUFF Property Name	Phase Where Set	Normal Use / Sample Value
application	1	application program id / selectmicro
evalfilename	1	stored evaluation info / semisave
mainevaluator	1	name of evaluator / (Fred Martin)
password	1	rudimentary protection against tampering / x
otherevaluators	1	others who can change parameters / (Nick Cercone)
logfilename	1	file for saving user dialogue / semilog
tallyerrors	1	flag shows whether to tally errors / t
errorid	1	one-byte code which identifies errors / 99
tallyerrortype	1	separate errors by type code after errorid / t
errorstuff	1	tallying error data (See figure 3.1a.)
questfilename	2	file containing questionnaires / evstdquest
ansfilename	2	file containing user answers / semianswers
audiences	2	audiences for all questionnaires / (end_user designer)
timingstuff	3	timing questionnaire delivery (See figure 3.1b.)
dspecs	3	summary of designer-specified triggers / (5 10 83)
endofrun	4	one-byte code triggers end-of-run questionnaires / 255
usercomments	5	identifies user comments / (comment complaint beef)
commentphrases	5	key phrases to seek in comments (See figure 3.1c.)
commentfate	5	what to do with user comments / intercept
commentfilename	5	file for saving user comments / semicomments
timesystempauses	6	flag shows whether to time system delays / t
timeuserpauses	6	flag shows whether to time user delays / t
timesessionlength	6	flag shows whether to time user session / t
timingfilename	6	file to hold timing information (See figure 3.1d.)
knownusers	evuser	known users and relationships (See figure 3.1e.)
sessionstart	"	the time a run started / 6554382
lastread	"	the time the last read occurred / 6554385
lastwrite	"	the time the last write occurred / 6554390
systempauses	"	number and size of system pauses (See figure 3.1d.)
userpauses	"	number and size of user pauses (See figure 3.1d.)
lastactivity	"	the time the last i/o activity occurred / 6554385

In phase one, the designer is asked to identify himself, the name of his application program, and the name of a file where the evaluation data may be stored. If this evaluation file is empty, the designer is invited to select a password, otherwise the designer must reproduce the password which was stored earlier in the evaluation file. This password scheme is intended to provide protection against casual tampering with the evaluation parameters. The designer may set up different evaluation files for the various application systems under his control. Once identified as a valid designer-user, the designer may

specify the names of other people who will be allowed to use the system in designer-mode. These people are also subject to the password constraints outlined above.

Next, the designer may specify a log file for keeping a detailed log of user dialogue. This feature is redundant in a UNIX environment, being replaced by the UNIX SCRIPT feature. Finally, the designer will indicate whether the evaluation system is to tally error messages from the application system to the user. If so, the tallies are kept under the property name ERRORSTUFF, the format of which is shown in Figure 3.1a in Backus-Naur-Form (BNF).

In phase two, the designer may build, modify, and/or inspect the evaluation questionnaires. Discussion of this process is left to the next section. Three more properties of EVALSTUFF are set in phase two. The first contains the file name where the questionnaires are saved. The designer may use the standard default name (EVSTDQUEST) or provide his own. The second property specifies the file name where user answers to questionnaires are to be stored. Again, the designer may choose a standard name (EVSTDANS) or provide his own. Finally, the program examines all existing questionnaires and builds a list of all possible user categories (such as enduser, programmer, or buyer), which is saved as property AUDIENCES. This list is used during application program runs, to ask a first-time user to identify his role with respect to the application system.

Figure 3.1a. Format of Errorstuff

Item		Form
errorstuff	::=	(error*)
error	::=	(errortype count)
errortype	::=	nil | integer 0...255
count	::=	integer

Where * means 0 or more occurences.

Phase three allows the designer to specify the timing of the various questionnaires. Each questionnaire may be given upon startup of an application run, at designer-triggered times during the run, or at end-of-run. The BNF format of TIMINGSTUFF, the property in which this timing information is stored, is shown in Figure 3.1b.

Phase four sets only the property ENDOFRUN. This is a one-byte code which the application program will 'print' to signal the end of the application run, at which time it will have the side-effect of causing end-of-run questionnaires to be given to the user.

In phase five, the designer specifies whether he wants to allow users to enter spontaneous comments. He may specify any number of 'trigger-words' which, if read as the first element of a user response, will be interpreted as the beginning of a comment. He may choose the name of a file where such comments will be stored for later inspection. He may also specify whether a comment will be passed on to the application program, or intercepted and 'swallowed' by the evaluation logic. Finally, he may specify whether comments should be scanned for key words or phrases. This latter feature is not strongly supported in the present version of the program, and is largely left as a future enhancement. If thoroughly developed, the feature would tally comments by such categories as praise, criticism, remarks concerning response times, and so forth. Figure 3.1c shows the BNF format of the property COMMENTPHRASES.

Figure 3.1b. Format of Timingstuff

Item		Form
timingstuff	::=	(questionnaire_timing*)
questionnaire_timing	::=	(questionnaire_number timing_info)
timing_info	::=	(timing_code*)
timing_code	::=	s | e | dspec
dspec	::=	an integer 0...255
dspecs	::=	(dspec*)

Where * means 0 or more occurences.

Figure 3.1c. Format of Commentphrases

Item		Form
commentphrases	::=	(phrasecategory*)
phrasecategory	::=	(phrasecatnumber phrasecatheader phrasecatbody)
phrasecatnumber	::=	a positive integer
phrasecatheader	::=	a character string
phrasecatbody	::=	(phrase*)
phrase	::=	a character string

Where * means 0 or more occurences.

The sixth and final phase allows the designer to specify whether to record timing information during user runs. Presently, the system can time a user's overall run length, and the number and length of 'system pauses' and 'user pauses'. These pauses are defined as follows:

A system pause is the interval from the previous i/o activity to a write.

A user pause is the interval from the previous i/o activity to a read.

Timing information is stored on a separate file, the name of which may be specified by the designer during phase six. The BNF format of timing file records is shown in Figure 3.1d.

Figure 3.1d. Format of Timingfile

Item		Form
timingfile	::=	(runtiming)
runtiming	::=	(userid systempauses userpauses sessionlength)
systempauses	::=	pauseinfo
userpauses	::=	pauseinfo
pauseinfo	::=	(number min max total) | nil
number	::=	integer seconds
min	::=	integer seconds
total	::=	integer seconds
sessionlength	::=	integer seconds | nil

A few other properties of EVALSTUFF are set, not in designer-mode, but when an application program is running. These include KNOWNUSERS, which keeps track of the identity of all users of an application in order to tell whether to administer first-time-user or repeat-user questionnaires to them. Figure 3.1e shows the format of KNOWNUSERS. Also included are various properties which record the timing of i/o activities during an application run.

3.2.1.2. Question Formats

The program allows a designer complete freedom in setting up questions, in that a question may be arbitrarily long, and may accept free-form or forced-choice answers from a user. In the standard questionnaires, most questions use a dichotomous (yes/no) format, in an effort to minimize user time spent in answering them. Other questions use a forced-choice rating scale, offering the user four or five alternative answers. Still other questions allow the user to make any response at all.

Many other question formats are possible, including numerical rating scales [Sundberg, 1977], adjective checklist scales [Gough, 1960], polar adjectives [Coombs and Alty, 1981], and Likert agreement scales [Likert, 1932]. The relative merits of these and other formats can best be explored over the long-term use of the system.

Figure 3.1e. Format of Knownusers

Item		Form
knownusers	::=	(userinfo*)
userinfo	::=	(user_id relationships)
user_id	::=	a character string
relationships	::=	(relationship*)
relationship	::=	designer | knowledgesource | buyer | programmer | enduser | Audience_descriptor

Where * means 0 or more occurences.

3.2.1.3. Questionnaire Contents

During phase two of a designer-mode run, the designer may examine, add, delete, or replace questionnaires and their contents. In general, the evaluation program treats the designer's terminal as a 'glass Teletype' and does not take advantage of features such as terminal page memories or windowing. It can thus be used on a wide variety of hardware, since it is not set up to take advantage of the idiosyncracies of a particular terminal type. A potential disadvantage of not utilizing advanced features such as windowing is that maintenance of large questionnaires may become awkward as information scrolls off the designer's screen. This potential problem has been minimized by breaking questionnaire contents into 'categories' of questions, and by 'focusing' on a single category at a time, rather than upon an entire questionnaire.

The reader may find figure 3.2 useful while reading the following discussion of a progression through a questionnaire's contents. Figure 3.2 is a BNF representation of the formats of questionnaires. (Answer-file record formats are very similar to questionnaire formats and are shown in BNF notation in figure 3.3.). Early in phase two, the designer is shown a list of all the questionnaires which presently exist and is invited to focus on a particular questionnaire or add a new one. If the designer chooses to focus on an existing questionnaire, he may then focus on the questionnaire's header, its constituent categories, or the characteristics of its intended audience, in order to examine or change these values, or he may choose to delete the entire questionnaire.

Focusing on the categories of a given questionnaire causes the program to display the header lines of all existing categories in the questionnaire and then invite the designer to focus on one of them or to add a new category. If he chooses an existing category, he may then focus on the category's header or its constituent questions in order to examine or change them, or he may choose to delete the entire category.

Figure 3.2. Questionnaire Format

Item		Form
Questionnaires	::=	(Questionnaire*)
Questionnaire	::=	(Questionnaire_number Audience_info Questionnaire_info)
Questionnaire_number	::=	a positive integer
Audience_info	::=	(Audience_experience Audience_type)
Audience_experience	::=	(Experience*)
Experience	::=	first_time | repeat
Audience_type	::=	(Audience*)
Audience	::=	all | designer | knowledgesource | buyer | programmer | enduser | Audience_descriptor
Audience_descriptor	::=	a character string
Questionnaire_info	::=	(Questionnaire_header Categories)
Questionnaire_header	::=	a character string
Categories	::=	(Category*)
Category	::=	(Category_number Category_info)
Category_number	::=	a positive integer
Category_info	::=	(Category_header Questions)
Category_header	::=	a character string
Questions	::=	(Question*)
Question	::=	(Question_number Question_info Aux_questions)
Question_number	::=	a positive integer
Question_info	::=	(Question_header Detail_lines Comment_flag Answer_values)
Question_header	::=	a character string
Detail_lines	::=	(Detail_line*)
Detail_line	::=	a character string
Comment_flag	::=	T | NIL
Answer_values	::=	(Answer_range*)
Answer_range	::=	(Answer_value Answer_value)
Answer_value	::=	NIL | a character string | a number
Aux_questions	::=	(Auxiliary*)
Auxiliary	::=	(Category Answer_triggers)
Answer_triggers	::=	(Answerrange*)

Where * means 0 or more occurrences.

Figure 3.3. Answer File Record Formats

Item		Form
Answer_file	::=	(Questionnaire_answers*)
Questionnaire_answers	::=	(User_id Questionnaire_a)
User_id	::=	a character string
Questionnaire_a	::=	identical to the Questionnaire format above, except Question_info is replaced by Answer_info and auxiliary triggers are always nil.
Answer_info	::=	(Question_header nil Comment Answer)
Comment	::=	a character string | nil
Answer	::=	a character string | a number

Where * means 0 or more occurences.

Similarly, focusing on the questions of a given category causes the program to display the first lines of all existing questions in the category and then to invite the designer to focus on one of the questions or add a new question. If he chooses an existing question, he may then focus on its header (the first line), its body (the remaining lines), its comment trigger, its permissible answer values, or its auxiliary questions, in order to examine or change any of these values, or he may choose to delete the entire question. The comment trigger of a question is a switch that the designer may set so that the evaluation program will ask a user to comment or expand upon his answer to the question. Answer values for a question constitute an optional collection of permissible user responses. By manipulating answer values, the designer can cause the evaluation program to edit user responses to questions and accept only valid responses.

Auxiliary questions were added to the question definitions at the suggestion of Nick Cercone. They provide an optional powerful recursive substructure to a question. A brief example should serve to illustrate this potential. One might wish to ask a multiple-choice question of a user, such that if a user answered 'a', 'b', or 'c' one would just go on to the next question, but if the user answered 'd', one might want to ask a few extra questions, and if the user answered 'e' one might ask a different group of extra questions. These extra

questions are implemented as auxiliary questions. Each question (including an auxiliary question) may have any number of categories of auxiliary questions. A category of auxiliary questions will be asked of a user when the user replies appropriately to its parent question. The designer may focus on an auxiliary question category and its contents in the same way as he focuses on any other category, as described above.

After any component has been inspected and/or modified, the program allows further changes at the same level, or return to a higher level. Among the options possible at most levels is the ability to replicate an item (such as a question, category, or even an entire questionnaire) in order to make the nucleus of a new item. Thus, the designer can quickly and easily access and modify any desired questionnaire components. In practice, the entire designer-mode run can be completed in well under five minutes, unless the designer is making extensive revisions to the standard questionnaires.

3.2.2. Changes to the Application Program

From the outset, it seemed clear that the application system designer should be able to activate the evaluation system with the least possible disruption to his application program. This goal was in potential conflict with the need for the evaluation logic to intercept all input-output (i/o) instructions in order to sense user comments, time various events, and log i/o activity to a separate logfile. The method used is as follows:

(1) The application designer inserts a single line at the beginning of his program of the form (INITEVAL filename), where the filename refers to a file containing all pertinent evaluation customization parameters. This is the file in which the property list of EVALSTUFF has been saved by the designer-mode run described previously. INITEVAL senses that it is in user/application-mode rather than designer-mode, which causes it to give any relevant start-of-run questionnaires to the user and perform various initialization functions.

(2) The application designer changes all i/o instructions to their evaluation system counterparts. For instance, READ becomes EVREAD and WRITE becomes EVWRITE. Fortunately, such changes can be implemented easily by means of global editor commands in most modern systems, and can easily be undone, if desired, by the same means.

(3) Optionally, the designer may insert a line of the form (EVPRINC trigger) at any point(s) in his program in order to trigger mid-run questionnaires. He may do this at will, using the same trigger values in order to present the same questionnaires more than once, or using different trigger values in order to present different questionnaires. EVPRINC is the evaluation system equivalent of the normal PRINC instruction.

(4) Optionally, the designer may insert codes at the beginning of his error messages to the user, so that the evaluation logic can detect and categorize such messages. These codes are stripped from the messages and are never seen by the user.

(5) The designer inserts a single line at the end of his program of the form (EVPRINC endtrigger), where the endtrigger causes the evaluation logic to administer any end-of-run questionnaires, and to save the results of the user run on appropriate files.

In practice, all these changes can be made to a typical application program in a matter of minutes, and can be undone equally quickly, if necessary. Figure 3.4 shows a trivial LISP program listing, where the program contains all the necessary modifications to activate the evaluation system. In the designer-mode run for this hypothetical application (not shown), the designer specified that: mid-run questionnaires would be triggered by the character '%' (decimal 37); error messages would begin with '*' (decimal 42), and end-of-run questionnaires would be triggered by '!' (decimal 33). Comments are prefaced by ';' in the listing.

Figure 3.4. 'Tinytest'. A Program to Use the Evaluation System.

```
; This program is just a shell for testing the evaluation system.

(defun tinytest ()
 (prog ()

; First load the main evaluation logic. It will load anything else it needs.

 (load 'evmain)

; Now set up the evaluation environment and give any start-up questionnaires.

 (initeval 'tinysave)

; Display a message using the modified function evmsg rather than msg.

 (evmsg "Tiny here, just to print something." N)

; Demonstrate reading, using evcleanread rather than cleanread.
; The evaluation system is noting timing of i/o activities and
;  monitoring inputs to see whether they are user comments.

 (evmsg "Let's do a few reads. Reply stop when you've had enough." N)
 loop
 (cond ((null (equal (evcleanread) 'stop)) (go loop)))

; Mid-run questionnaire(s) can be triggered at the designer's request.

 (evmsg "Tiny again. I'll try to trigger some mid-run stuff now." N)
 (evprinc (ascii 37))

; The designer has told the evaluation logic that messages starting with
;  an asterisk are to be tallied as error messages.

 (evmsg "Tiny again. Now let's send a couple of dummy error messages." N)
 (evmsg *1 "Here is an error of type one." N)
 (evmsg *2 "Here is an error of type two." N)

; The designer now signals end-of-run by displaying the pre-arranged code.

 (evmsg "Tiny here. Time to trigger end of run stuff." N)
 (evprinc (ascii 33))
 (evmsg "Tiny signing off now." N)))
```

CHAPTER 4

DIALOGUES WITH AN EVALUATION SYSTEM

4.1. A Dialogue With an Imaginary Evaluation System

The following annotated dialogue fragment represents an early attempt to define the kinds of interactions which could occur between a system designer and the evaluation program. The designer is specifying the type of evaluations to be performed. The evaluation program will note these specifications and carry them out when an 'ordinary' user activates the system. In this sample dialogue, the designer's inputs are indicated by **bold face** type. First, the program identifies the system to be evaluated, the designer-user, and various miscellaneous information.

(initeval nil)
This program helps evaluate other programs.
Do you want instructions? (Y/N).
no
PHASE 1: General initialization.
What file can be used to save run-independent evaluation data?
foofile
foofile already exists. (C)lear it, (A)dd to it, or (N)ew filename?
C
foofile cleared.
What is the name of the system or program to be evaluated?
foosystem
This program will recognize you in future as the main system evaluator:
Please enter your name.
John Doe
Please choose a password.
xxxxxxxxx
Again please, to be sure.
xxxxxxxxx
Will you allow some other person to change evaluation rules? (Y/N)
Y
Please provide the name of another evaluator.
Jane Doe
The new evaluator can choose a password when (s)he uses the program.
Will you allow some other person to change evaluation rules? (Y/N)
No

What file can be used to log user runs?
foolog
foolog created.
Tally error messages from the system to the user? (Y/N)
y
Enter numeric value of character to identify an error message (0...255).
0
Tally error messages by a type code following this character? (Y/N)
ok
PHASE 1 complete.
Reenter phase 1 to change anything? (Y/N)
no

Next, the program sets up a questionnaire to be given to first-time system users. It was not yet realized that different user types might warrant different questionnaires.

PHASE 2: Questions for first-time system users.
Give a questionnaire when a first-time user signs on to foosystem? (Y/N)
yes
Do you want to see a summary of the new user questionnaire? (Y/N)
y
The questionnaire for first-time users includes:
1. Introductory message and request for help from user.
2. user name, age and sex.
3. user computing experience level.
4. user education level.
5. user motivation (ie stated reasons for using the system).
6. user system preparedness (ie how well briefed on the system).
7. user mood (eg level of contentment, frustration, anger).
8. user alone or accompanied.
9. user report on site conditions.

Enter number (1 to 9) to see details of a category (0 if none wanted).
9

Now, the designer may browse through the contents of a category. The menu prompting style is comprehensive but bulky. It was not yet obvious that this sequential style could be annoyingly time-consuming.

Site Conditions (View as many questions as you wish):

The noise level here is:
1. Much too quiet.
2. Somewhat more quiet than I would like.
3. Just about ideal for my purposes.
4. Somewhat more noisy than I would like.
5. Much too noisy.

Question is forced choice response.
Acceptable answers are:
1,2,3,4,5.
There are no auxiliary questions to this question.
Use this question 'as is'? (Y/N)
ok
View next question? (Y/N)
ok
The temperature here is:
1. Much too cold.
2. Somewhat more cold than I would like.
3. Just about ideal for this kind of work.
4. Somewhat more warm than I would like.
5. Much too warm.
Please add a comment to explain why you chose number x above.

Question is forced choice response.
Acceptable answers are:
1,2,3,4,5.
There are no auxiliary questions to this question.
Use this question 'as is'? (Y/N)
no
(C)hange or (D)elete the question or adjust acceptable (R)esponses?
D
Question temperature here deleted.
View next question? (Y/N)
yes
The lighting here is:
1. Much too bright or glaring.
2. Somewhat more bright or glaring than I would like.
3. Just about right for this kind of work.
4. Somewhat more dim or dark than I would like.
5. Much too dim or dark.
Please add a comment to explain why you chose number x above.

Question is forced choice response.
Acceptable answers are:
1,2,3,4,5.
Level 1 auxiliary question(s) present. View auxiliary? (Y/N).
yes
Descending to lower level:

This situation has existed for:
1. Over a year.
2. Six months to a year.
3. Three to six months.
4. Under three months.
5. Don't know.
Question is forced choice response.
Acceptable answers are:
1,2,3,4,5.
There are no auxiliary questions to this question.
Answers triggering this auxiliary are:

```
1,2
Use answer triggers 'as is'? (Y/N)
y
Use this question 'as is'? (Y/N)
y
View next auxiliary question? (Y/N)
no
Add an auxiliary question? (Y/N)
no
```

At this stage, the possibilities for confusion arising from nesting of auxiliary questions are noticeable. This example only descended one level, but the questions "View next auxiliary question?" followed by "Add an auxiliary question?" may already puzzle the designer-reader. A quick succession of such questions from a nested depth of three or four levels would no doubt be very confusing. The example continues.

```
Returning to higher level

Use this question 'as is'? (Y/N)
yes
View next question? (Y/N)
no
Add a question to category site conditions? (Y/N)
n
You may look at another category now, if you like.
Do you want to see the categories in the questionnaire? (Y/N)
n
Enter number (1 to 9) to see details of a category (0 if none wanted).
0
Delete an entire category? (Y/N)
no
Add a new (custom) category? (Y/N)
no
View entire (modified) questionnaire now? (Y/N)
no
Questionnaire saved on file foofile.
PHASE 2 complete.
Reenter phase 2 to change anything? (Y/N)
no
```

The example continues. As with the first-time user questionnaire, it was not immediately obvious that different types of users might warrant different repeat-user questionnaires.

```
ep PHASE 3: Questions for repeat system users.
```

ep Give a repeat user questionnaire when a repeat user signs on? (Y/N)
yes
Do you want to see a summary of the repeat user questionnaire? (Y/N)
y
The repeat user questionnaire includes:
...
and so forth.

The preceding dialogue fragment illustrates the intent of the designer-mode portion of the evaluation system, but contains several features, some minor and some substantial, which deserved alteration. For instance, a single system password for all authorized designers should suffice, rather than having each designer create his own password. Next, and more importantly, it seemed unwise to tie the first-time or repeat-user attribute directly into a questionnaire. A designer may want to give a particular questionnaire to first-time users at one point, and give the same questionnaire to repeat users at another point. A third and substantial area of improvement is in the display of questions. The format illustrated is slow, bulky and (worst of all) sequential in nature. Next, it became apparent that different user classes (buyer, programmer, etc) often warranted different questionnaires. The actual system implementation benefited from the lessons learned in developing imaginary dialogues such as the preceding example.

4.2. An Actual Dialogue With the Evaluation System.

The following annotated dialogue took place between a designer and an implementation of the evaluation program. Again, the designer is specifying the type of evaluations to be performed. The evaluation program notes these specifications and carries them out when an 'ordinary' user activates the application system. The application system in this example is Selectmicro, a small-scale expert system which helps consumers pick a microcomputer appropriate to their needs. A brief description of Selectmicro appears in Appendix B. The evaluation program can be invoked stand-alone, as in this example, or from within a program. In the latter case, it is passed the name of a file containing the evaluation specifications established in an earlier stand-alone run (as was done in the 'tinytest'

example of Figure 3.4). Note that the amount of effort expended by the designer is substantially reduced compared to the example of section 4.1. The program begins with an (optional) explanation of its function. Designer inputs are shown in **bold face** type.

```
% lisp
Franz Lisp, Opus 38.79
-> (load 'evalmain)
[load evalmain]
t
-> (initeval nil)
Initeval. © Copyright 1984, Fred J. Martin
This program helps evaluate other programs.
Do you want instructions? (y/n) y
Thank you for using the evaluation package.

The program you have called can be used in two ways.
It can set up an evaluation environment, or conduct an evaluation.

When called with a null parameter list, (as you have just done),
it assumes that you are a system designer who will provide rules
for having your application program(s) evaluated.

This program will ask you various questions, and will use your
answers to define the evaluation criteria. Among other questions,
it will ask you to provide a filename where it can store the
information you supply. After you finish the present initialization
run, you will be able to have any program evaluated by inserting
the statement '(initeval filename)' as the first instruction in
your program. The filename will be the name you choose in this run,
namely the file where initeval stores the information you are about
to provide.

Press s to stop the display, any other key to see more.
x

This run consists of six distinct phases.
Phase 1 does general initialization, asking you for various filenames.
Phase 2 lets you browse the user questionnaires, altering them at will.
Phase 3 gathers information from you about questionnaire timing.
Phase 4 asks how to identify end of run for your application.
Phase 5 asks you about spontaneous user comments.
Phase 6 allows you to specify automatic data gathering parameters.

In all cases, just reply to the prompt messages.
They will lead you through all six phases.
```

The program now leads the designer through its general initialization phase. It will recognize that file 'semisave' is empty, and that this must therefore be a new application.

Phase 1: General Initialization.
What is the name of the application system or program to be evaluated?
selectmicro
What file can be used to save run-independent evaluation data?
semisave
Please enter your name.
Fred Martin
This program will recognize you in future as a valid system evaluator
for selectmicro.
Please choose a password.
x
Again please, to be sure.
x
Will you allow someone else to change evaluation rules? (y/n) **y**
Please provide the name of another evaluator.
Nick Cercone
Will you allow someone else to change evaluation rules? (y/n) **n**
What file can be used to log user runs?
semilog
Tally error messages from your system to the user? (y/n) **y**
Enter numeric value of ascii character which will precede
error messages from your system to the user (0...255) **0**
Tally error messages by a type code following the error identifier? (y/n) **y**
Phase 1 complete.
Reenter Phase 1 to change anything? (y/n) **n**

Next, the program allows questionnaire creation or modification. Since no questionnaires exist prior to this example run, this phase becomes fairly time-consuming. Once 'standard' questionnaires have been created, this phase should normally be very brief. In this example: question one shows a brief (single-line) question with a designer-specified allowable answer range; question two shows another simple question with individual designer-specified allowable answers; question three shows a multiple-line question where the user must provide an answer from a given range, but will also be asked to comment (expand upon) his answer; questions four through six are omitted for the sake of brevity.

Phase 2: Questionnaire Contents.
What file holds the questionnaires?
Enter filename or choose (s)tandard file default.
s
Save questionnaire answers on what file?
Enter filename or choose (s)tandard file default.
s
The following questionnaires presently exist:
nil
Enter number of questionnaire to focus on:

Use 0 if none, use 999 to add a new questionnaire. **999**
Questionnaire number 1
Should questionnaire be given to (f)irst-time users, to
(r)epeat users, or to (b)oth? **f**
What is the intended target audience type for this questionnaire?...
(a)ll system users, (n)o audience type known yet, or (s)pecific subgroups.**a**
Please enter questionnaire description.
User Characteristics
Add a category to questionnaire (User Characteristics)? (y/n) **y**
Please enter category description.
Basics
Add a question to category Basics? (y/n) **y**
Question number: 1
Please enter the first line of the question.
Please enter your age at your nearest birthday.
Enter next line. Reply nil if no more lines.
nil
Ask respondent to comment on answer? (y/n) **n**
Add an allowable answer for this question? (y/n) **y**
(s)ingle value or (r)ange of values? **r**
Please enter low value. Use nil if none. **3**
Please enter high value. Use nil if none. **123**
Add an allowable answer for this question? (y/n) **n**
Current question is:
(Please enter your age at your nearest birthday.)...
Add an auxiliary question category to this question? (y/n) **n**
Finished with auxiliary question category for question
(Please enter your age at your nearest birthday.).

Reverting to higher level.
Add a question to category Basics? (y/n) **y**
Question number: 2
Please enter the first line of the question.
What is your sex? Please reply m or f.
Enter next line. Reply nil if no more lines.
nil
Ask respondent to comment on answer? (y/n) **n**
Add an allowable answer for this question? (y/n) **y**
(s)ingle value or (r)ange of values? **s**
Please enter value.
m
Add an allowable answer for this question? (y/n) **y**
(s)ingle value or (r)ange of values? **s**
Please enter value.
f
Add an allowable answer for this question? (y/n) **n**
Current question is:
(What is your sex? Please reply m or f.)...
Add an auxiliary question category to this question? (y/n) **n**
Finished with auxiliary question category for question
(What is your sex? Please reply m or f.).

Reverting to higher level.

Add a question to category Basics? (y/n) **y**
Question number: 3
Please enter the first line of the question.
What is your computing experience? Answer with the appropriate letter.
Enter next line. Reply nil if no more lines.
a Never used a computer before now.
Enter next line. Reply nil if no more lines.
b Occasionally use computers (less than once a month).
Enter next line. Reply nil if no more lines.
c Often use computers (one to ten times a month).
Enter next line. Reply nil if no more lines.
d Regularly use computers (eleven times a month to daily).
Enter next line. Reply nil if no more lines.
e Constantly use computers (more than once a day).
Enter next line. Reply nil if no more lines.
nil
Ask respondent to comment on answer? (y/n) **y**
Add an allowable answer for this question? (y/n) **y**
(s)ingle value or (r)ange of values? **r**
Please enter low value. Use nil if none. **a**
Please enter high value. Use nil if none. **e**
Add an allowable answer for this question? (y/n) **n**
Current question is:
(What is your computing experience? Answer with the appropriate letter.)...
Add an auxiliary question category to this question? (y/n) **n**
Finished with auxiliary question category for question
(What is your computing experience? Answer with the appropriate letter.).

Reverting to higher level.
Add a question to category Basics? (y/n) **y**

.
. Questions four, five, and six omitted.
.

Question seven illustrates the creation of an auxiliary question category. If a user indicates that his mood is not "relaxed, contented, or happy", then the program will ask an auxiliary question to try to discover the cause of the user's problem.

Question number: 7
Please enter the first line of the question.
What is your present mood or frame of mind?
Enter next line. Reply nil if no more lines.
Choose the letter that comes closest to describing how you feel now.
Enter next line. Reply nil if no more lines.
a Relaxed, contented or happy.
Enter next line. Reply nil if no more lines.
b nervous, apprehensive, or anxious.
Enter next line. Reply nil if no more lines.
c angry, upset, or frustrated.
Enter next line. Reply nil if no more lines.

nil
Ask respondent to comment on answer? (y/n) **n**
Add an allowable answer for this question? (y/n) **y**
(s)ingle value or (r)ange of values? **r**
Please enter low value. Use nil if none. **a**
Please enter high value. Use nil if none. **c**
Add an allowable answer for this question? (y/n) **n**
Current question is:
(What is your present mood or frame of mind?)...
Add an auxiliary question category to this question? (y/n) **y**
Please enter category description.
more on bad mood
Add a question to category (more on bad mood)? (y/n) **y**
Question number: 1
Please enter the first line of the question.
Please indicate the main cause of your present mood.
Enter next line. Reply nil if no more lines.
a Personal problems (eg marital, family, health).
Enter next line. Reply nil if no more lines.
b Problems related to work or school (eg workload, argument with boss).
Enter next line. Reply nil if no more lines.
c Problems related to your surroundings (eg weather, noise).
Enter next line. Reply nil if no more lines.
d Problems related to this program or system.
Enter next line. Reply nil if no more lines.
nil
Ask respondent to comment on answer? (y/n) **n**
Add an allowable answer for this question? (y/n) **y**
(s)ingle value or (r)ange of values? **r**
Please enter low value. Use nil if none. **a**
Please enter high value. Use nil if none. **d**
Add an allowable answer for this question? (y/n) **n**
Current question is:
(Please indicate the main cause of your present mood.)...
Add an auxiliary question category to this question? (y/n) **n**
Finished with auxiliary question category for question
(Please indicate the main cause of your present mood.).
Reverting to higher level.
Add a question to category (more on bad mood)? (y/n) **n**
Add an answer value to trigger this category? (y/n) **y**
(s)ingle value or (r)ange of values? **s**
Please enter value.
c
Add an answer value to trigger this category? (y/n) **n**
You may add another auxiliary category now, if you want.
Current question is:
(What is your present mood or frame of mind?)...
Add an auxiliary question category to this question? (y/n) **n**
Finished with auxiliary question category for question
(What is your present mood or frame of mind?).

Reverting to higher level.
Add a question to category Basics? (y/n) **y**

.
. Questions eight and nine omitted for the sake of brevity.
.

Add a question to category Basics? (y/n) **n**
Add a category to questionnaire (User Characteristics)? (y/n) **n**
The following questionnaires presently exist:

Number	Header	User Characteristics	
		Experience	Type
1	(User Characteristics)	(first_time)	(all)

Enter number of questionnaire to focus on:
Use 0 if none, use 999 to add a new questionnaire. **0**
Write questionnaires to what file?
Enter filename or choose (s)tandard file default.
s

Phase 2 complete.
Reenter phase 2 to change anything? (y/n) **n**

The questionnaire(s) developed to date have now been safely stored on disk and the program moves on to collect other information from the designer. Throughout these examples, note that the evaluation program gives fairly long, detailed prompt messages but requires only simple (often one-character) responses from the designer. Designers are expected to change questionnaire contents infrequently, so the detailed prompt messages are appropriate. If these detailed prompts were seen often, they could become annoying.

Phase 3: Questionnaire timing.
The following questionnaires presently exist:

Number	Header	When Given
1	(User Characteristics)	not specified.

Enter number of questionnaire to focus on:
Use 0 if none. **1**
Questionnaire 1 is presently given at:
time not specified.
Give questionnaire at (s)tartup, under (d)esigner control,
or at (e)nd of run?_
Enter any combination of s, d, and e, separated by spaces, or
(n)o change. **s**
Questionnaire 1 is presently given at:
startup.
Give questionnaire at (s)tartup, under (d)esigner control,
or at (e)nd of run?_
Enter any combination of s, d, and e, separated by spaces, or
(n)o change. **n**
Enter number of questionnaire to focus on:
Use 0 if none. **0**
Phase 3 complete.

```
Reenter phase 3 to change anything? (y/n) n
Phase 4: End of Run Specifications.
What character will you send to signal end of run and trigger
any end-of-run questionnaires?...
Enter numeric value of ascii code (0...255) 255
Phase 4 complete.
```

Here, the designer may choose words which will be interpreted as comment triggers. For instance, if a user response begins with the word 'comment', it will be intercepted by the evaluation logic and saved on file 'semicomments'

```
Phase 5: Spontaneous User Comments.
Present trigger keywords for user comments are:
nil
Enter revised keywords, or the word nil to disallow spontaneous
user comments, or (n)o change to the present list.
comment complaint beef
Present trigger keywords for user comments are:
(comment complaint beef)
Enter revised keywords, or the word nil to disallow spontaneous
user comments, or (n)o change to the present list.
n
Scan user comments for words or phrases of interest? (y/n) n
Should user comments be (i)ntercepted by the evaluation system,
or (f)orwarded to your application program? i
Save user comments on a separate file? (y/n) y
Please enter filename: semicomments
Phase 5 complete.
Reenter phase 5 to change anything? (y/n) n
Phase 6: Automatic Data Gathering Parameters
The evaluation system can time system pauses.
This feature is presently set off.
Toggle this feature? (y/n) n
The evaluation system can time user pauses.
This feature is presently set off.
Toggle this feature? (y/n) n
The evaluation system can time the length of user sessions.
This feature is presently set off.
Toggle this feature? (y/n) n
Save timing information on what file?
semitiming

Phase 6 complete.
Reenter phase 6 to change anything? (y/n) n

Initeval completed.
```

The program is now complete and control is returned to LISP. The example continues, showing how the designer might return at a later date to modify the questionnaires or change other parameters. Since the 'semisave' file now exists, the designer is subjected to a password check before he is allowed access to the evaluation data.

```
-> (exit)
%
% lisp
Franz Lisp, Opus 38.79
-> (load 'evalmain)
[load evalmain]
t
-> (initeval nil)
Initeval. © Copyright 1984, Fred J. Martin
This program helps evaluate other programs.
Do you want instructions? (y/n) n
Phase 1: General Initialization.
What is the name of the application system or program to be evaluated?
selectmicro
What file can be used to save run-independent evaluation data?
semisave
semisave already exists: (u)se it anyway or (n)ew filename? u
Please enter your name.
Fred Martin
[load semisave]
Please enter password. x
Will you allow someone else to change evaluation rules? (y/n) n
What file can be used to log user runs?
semilog
Tally error messages from your system to the user? (y/n) y
Enter numeric value of ascii character which will precede
error messages from your system to the user (0...255) 99
Tally error messages by a type code following the error identifier? (y/n) y
Phase 1 complete.
Reenter Phase 1 to change anything? (y/n) n
```

The designer will add a second questionnaire. Question one shows some of the power of auxiliary questions. If a user reply indicates a noisy environment, an auxiliary question is asked to try to identify the source of the noise. If the answer to this auxiliary question indicates that computer equipment is the noise source, an auxiliary question to this question is asked, to try to identify the particular piece of noisy equipment.

```
Phase 2: Questionnaire Contents.
What file holds the questionnaires?
Enter filename or choose (s)tandard file default.
```

s
[load evstdquest]
Save questionnaire answers on what file?
Enter filename or choose (s)tandard file default.
s
The following questionnaires presently exist:

Number	Header	User Characteristics	
		Experience	Type
1	(User Characteristics)	(first_time)	(all)

Enter number of questionnaire to focus on:
Use 0 if none, use 999 to add a new questionnaire. **999**
Questionnaire number 2
Should questionnaire be given to (f)irst-time users, to
(r)epeat users, or to (b)oth? b
What is the intended target audience type for this questionnaire?...
(a)ll system users, (n)o audience type known yet, or (s)pecific subgroups.**a**
Please enter questionnaire description.
Physical Surroundings
Add a category to questionnaire (Physical Surroundings)? (y/n)**y**
Please enter category description.
Basics
Add a question to category Basics? (y/n)**y**
Question number: 1
Please enter the first line of the question.
The noise level here is:
Enter next line. Reply nil if no more lines.
a Much too quiet.
Enter next line. Reply nil if no more lines.
b Somewhat more quiet than I would like.
Enter next line. Reply nil if no more lines.
c Just about ideal for my purposes.
Enter next line. Reply nil if no more lines.
d Somewhat more noisy than I would like.
Enter next line. Reply nil if no more lines.
e Much too noisy.
Enter next line. Reply nil if no more lines.
nil
Ask respondent to comment on answer? (y/n)**n**
Add an allowable answer for this question? (y/n)**y**
(s)ingle value or (r)ange of values?**r**
Please enter low value. Use nil if none.**a**
Please enter high value. Use nil if none.**e**
Add an allowable answer for this question? (y/n)**n**
Current question is:
(The noise level here is:)_
Add an auxiliary question category to this question? (y/n)**y**
Please enter category description.
Noise Source
Add a question to category (Noise Source)? (y/n)**y**
Question number: 1
Please enter the first line of the question.
What is the main source of the noise?

Enter next line. Reply nil if no more lines.
a Computer equipment.
Enter next line. Reply nil if no more lines.
b Other equipment.
Enter next line. Reply nil if no more lines.
c Voices.
Enter next line. Reply nil if no more lines.
d Traffic or other outside noise.
Enter next line. Reply nil if no more lines.
e Other sources.
Enter next line. Reply nil if no more lines.
nil
Ask respondent to comment on answer? (y/n)**n**
Add an allowable answer for this question? (y/n)**y**
(s)ingle value or (r)ange of values?**r**
Please enter low value. Use nil if none.**a**
Please enter high value. Use nil if none.**e**
Add an allowable answer for this question? (y/n)**n**
Current question is:
(What is the main source of the noise?)...
Add an auxiliary question category to this question? (y/n)**y**

Now the second level of auxiliary question(s) is added. Questions at this level will only be asked when a user triggers the level above it, and also gives appropriate answers to the parent auxiliary question.

Please enter category description.
Source of equipment noise
Add a question to category (Source of equipment noise)? (y/n)**y**
Question number: 1
Please enter the first line of the question.
Which piece of equipment is mainly responsible for the noise?
Enter next line. Reply nil if no more lines.
a Printer(s).
Enter next line. Reply nil if no more lines.
b Keyboard(s).
Enter next line. Reply nil if no more lines.
c Terminal hum or whine.
Enter next line. Reply nil if no more lines.
d Other sources.
Enter next line. Reply nil if no more lines.
nil
Ask respondent to comment on answer? (y/n)**n**
Add an allowable answer for this question? (y/n)**y**
(s)ingle value or (r)ange of values?**r**
Please enter low value. Use nil if none.**a**
Please enter high value. Use nil if none.**d**
Add an allowable answer for this question? (y/n)**n**
Current question is:
(Which piece of equipment is mainly responsible for the noise?)...

Add an auxiliary question category to this question? (y/n)**n**
Finished with auxiliary question category for question
(Which piece of equipment is mainly responsible for the noise?).

Reverting to higher level.
Add a question to category (Source of equipment noise)? (y/n)**n**
Add an answer value to trigger this category? (y/n)**y**
(s)ingle value or (r)ange of values?**s**
Please enter value.
a
Add an answer value to trigger this category? (y/n)**n**
You may add another auxiliary category now, if you want.
Current question is:
(What is the main source of the noise?)...
Add an auxiliary question category to this question? (y/n)**n**
Finished with auxiliary question category for question
(What is the main source of the noise?).

Reverting to higher level.
Add a question to category (Noise Source)? (y/n)**n**
Add an answer value to trigger this category? (y/n)**y**
(s)ingle value or (r)ange of values?**r**
Please enter low value. Use nil if none.**d**
Please enter high value. Use nil if none.**e**
Add an answer value to trigger this category? (y/n)**n**
You may add another auxiliary category now, if you want.
Current question is:
(The noise level here is:)...
Add an auxiliary question category to this question? (y/n)**n**
Finished with auxiliary question category for question
(The noise level here is:).

Reverting to higher level.
Add a question to category Basics? (y/n)**y**

The designer goes on to add more questions and another entire questionnaire, the details of which are omitted for the sake of brevity. The example continues as phase 3 begins.

Phase 2 complete.
Reenter phase 2 to change anything? (y/n) n
Phase 3: Questionnaire timing.
The following questionnaires presently exist:

Number	Header	When Given
1	(User Characteristics)	startup.
2	(Physical Surroundings)	not specified.
3	(Summary of user impressions)	not specified.

Enter number of questionnaire to focus on:
Use 0 if none. **2**

```
Questionnaire 2 is presently given at:
time not specified.
Give questionnaire at (s)tartup, under (d)esigner control,
or at (e)nd of run?...
Enter any combination of s, d, and e, separated by spaces, or
(n)o change. d
What value will you send to trigger this questionnaire?...
Enter numeric value of ascii code (0...255) 35
Questionnaire 2 is presently given at:
designer triggers with 35.
Give questionnaire at (s)tartup, under (d)esigner control,
or at (e)nd of run?...
Enter any combination of s, d, and e, separated by spaces, or
(n)o change. n
Enter number of questionnaire to focus on:
Use 0 if none. 3
Questionnaire 3 is presently given at:
time not specified.
Give questionnaire at (s)tartup, under (d)esigner control,
or at (e)nd of run?...
Enter any combination of s, d, and e, separated by spaces, or
(n)o change. e
Questionnaire 3 is presently given at:
end of run.
Give questionnaire at (s)tartup, under (d)esigner control,
or at (e)nd of run?...
Enter any combination of s, d, and e, separated by spaces, or
(n)o change. n
Enter number of questionnaire to focus on:
Use 0 if none. 0
Phase 3 complete.
Reenter phase 3 to change anything? (y/n) y
```

As shown below, it is convenient to reenter phase 3 to double-check that the various questionnaires will be given at the desired times. Phases four and five are omitted for the sake of brevity. The example then continues with phase six.

```
Phase 3: Questionnaire timing.
The following questionnaires presently exist:

Number   Header                          When Given
1        (User Characteristics)          startup.
2        (Physical Surroundings)         designer triggers with 35.
3        (Summary of user impressions)   end of run.

Enter number of questionnaire to focus on:
Use 0 if none. 0
Phase 3 complete.
Reenter phase 3 to change anything? (y/n) n

.
. Phases four and five are omitted.
```

```
Phase 6: Automatic Data Gathering Parameters
The evaluation system can time system pauses.
This feature is presently set off.
Toggle this feature? (y/n) y
OK
The evaluation system can time user pauses.
This feature is presently set off.
Toggle this feature? (y/n) y
OK
The evaluation system can time the length of user sessions.
This feature is presently set off.
Toggle this feature? (y/n) y
OK
Save timing information on what file?
semitiming

Phase 6 complete.
Reenter phase 6 to change anything? (y/n) n
File semisave exists. OK to overwrite? (y/n) y
Initeval completed.
```

The program is now completed. Before leaving LISP, the designer displays the property list of EVALSTUFF in order to check its contents.

```
-> (plist 'evalstuff)
(timesessionlength t timeuserpauses t timesystempauses t
 timingfilename semitiming commentfilename semicomments
 commentfate intercept
 usercomments (comment complaint beef objection)
 endofrun 255
 timingstuff ((1 (s)) (2 (35)) (3 (e)))
 ansfilename evstdans questfilename evstdquest
 tallyerrortype t errorid 99 tallyerrors t logfilename semilog
 evaluators ((Fred Martin) (Nick Cercone)) password x
 evalfilename semisave application selectmicro)
-> (exit)
%
```

4.3. A User-run of a Program with the Evaluation System

The following annotated dialogue excerpt reflects the interactions between a user who is trying the Selectmicro program with the evaluation package attached to it. Selectmicro immediately passes control to the evaluation logic, since the statement '(initeval 'semisave)' has been inserted at the beginning of Selectmicro. The evaluation logic

reads the file 'semisave' and discovers that John Doe is a new user of Selectmicro. It therefore gives fairly detailed introductory remarks. Had John Doe been a repeat user, he would have been 'welcomed back' to Selectmicro and reminded about the comment feature. For the sake of clarity, outputs from the evaluation program (as opposed to those from Selectmicro itself) have been prefixed by 'ep'. User inputs are shown in **bold face** type.

-> **(selectmicro)**
ep Please enter your 'evaluation identification code'.
ep If you do not have such a code, then enter your name.
John Doe
ep According to the system files, you have not used selectmicro with the
ep evaluation feature before.
ep The evaluation feature is a program which is attached to selectmicro.
ep It gathers information from users like yourself on a variety of topics.
ep This information is passed on to the designers of selectmicro so that they
ep can further improve the system.
ep Please help improve selectmicro by accurately answering the questionnaires
ep which will be presented to you. Thank you for your help.
ep Ready to proceed?
yes
ep From time to time as the program is running, you may want to make a comment
ep about it. You may enter a comment any time the program is waiting for a
ep reply from you. Any comments you make will be passed on to the system
ep designers. If you start any reply with a keyword from the following list,
ep your reply will be interpreted as a comment:
ep (comment complaint beef).
ep After you enter your comment(s), the program will still be waiting for your
ep real reply to whatever question it asked you.
ep
ep Thank you again for your cooperation.
ep Ready to proceed?
yes
ep Which of these terms describe(s) your relationship to selectmicro?...
ep (enduser programmer designer buyer knowledgesource)
ep If you are not sure which one applies to you, you are probably
ep an enduser.
ep Enter all the names that apply, separated by spaces.
enduser
ep Are you satisfied that your entry is complete and correct?
y

The evaluation program now checks to see whether there are questionnaires to be given at start-of-run to a first-time enduser. It finds an appropriate questionnaire and

administers it. It edits the user's responses to many of the questions, as instructed by the designer. Question two shows how it deals with a user-response which fails the edit. The questions continue until there are no more start-of-run questions to give to this type of user.

```
ep Questionnaire 1.
ep question 1.
ep Please enter your age at your nearest birthday.
29
ep question 2.
ep What is your sex? Please reply m or f.
x
ep Inappropriate reply. Please enter new answer.
m
ep question 3.
ep What is your computing experience? Answer with the appropriate letter.
ep a Never used a computer before now.
ep b Occasionally use computers (less than once a month).
ep c Often use computers (one to ten times a month).
ep d Regularly use computers (eleven times a month to daily).
ep e Constantly use computers (more than once a day).
c
ep Please expand your answer - just a line to clarify it.
I sometimes use VISICALC at work.
ep question 4.
ep Please indicate your highest completed level of formal education.
ep Answer with the appropriate letter.
ep a grade school.
ep b high school.
ep c college or technical diploma.
ep d undergraduate degree.
ep e graduate degree.
c
ep question 5.
ep Please indicate your motivation for using this system.
ep Just summarize the main reason(s) on a single line.
I'm planning to buy a microcomputer.
ep question 6.
ep How thoroughly were you briefed on this system before you used it?
ep a No briefing at all.
ep b Casual, informal explanation.
ep c Detailed explanation, possibly with written documentation.
ep d Formal course of instruction.
ep e Other.
b
ep Please expand your answer - just a line to clarify it.
Fred just told me a bit about the program.
ep question 7.
ep What is your present mood or frame of mind?
ep Choose the letter that comes closest to describing how you feel now.
```

ep a Relaxed, contented or happy.
ep b nervous, apprehensive, or anxious.
ep c angry, upset, or frustrated.
c
ep auxiliary question 1.
ep Please indicate the main cause of your present mood.
ep a Personal problems (eg marital, family, health).
ep b Problems related to work or school (eg workload, argument with boss).
ep c Problems related to your surroundings (eg weather, noise).
ep d Problems related to this program or system.
b
ep question 8.
ep As you answer these questions, are you:
ep a Alone
ep b Accompanied by one person
ep c Accompanied by more than one person
a
ep question 9.
ep Please give your impressions of the site in which you are using the system.
ep Just enter a single-line summary of your impressions.
The room is noisy, crowded and dirty.
ep End of Questionnaire 1.

Control is returned to Selectmicro and the user sets about choosing the best computer(s) for his needs. Much of the Selectmicro dialogue is omitted in this example, since it is not particularly relevant. The evaluation logic is silent throughout most of this dialogue, but it is monitoring all the i/o as it occurs.

This program will help you select a microcomputer.
Please wait while knowledge base is loaded.

. Detail omitted for the sake of brevity.

Are you familiar with the program?**y**
Please enter your name.**John Doe**
Select a microcomputer?**y**
Are you considering any particular machines? (y/n)**y**
I'd like to know what they are.
Candidate name?**apple iie**
I am already familiar with: (apple iie)
If you do not have a new candidate, key nil.
Candidate name?**nil**
Candidate not added.
Do you want to see all my rules (selection criteria)?**n**
Shall I apply the rules to the candidates?**y**
Remaining in contention are: 101 candidates.

Show them anyway? (y/n).**n**
Ready to proceed?**y**

The evaluation logic is about to detect a spontaneous user comment. It will save the comment on a special file and will not pass it on to Selectmicro, as instructed by the designer earlier.

For your needs, is the following an essential requirement?
(must be capable of running under the operating system cp/m-80)
comment The program seems very slow.
ep Comment noted. Please reply to the original question.
y
(rule 20) eliminates the following: 50 candidates.
Show them anyway? (y/n).**no**
Ready to proceed?**ok**
(rule 20) will be ignored in remaining evaluations.
Remaining in contention are: 51 candidates.
Show them anyway? (y/n).**n**
Ready to proceed?**y**
For your needs, is the following an essential requirement?
(crt display must not be built into the micro cabinet)
explain crt display
I cannot yet explain (crt display)
I've noted your request and I'll see what I can find out.
Meanwhile, please continue as best you can.
Please answer yes or no to the original question.
yes
(rule 12) eliminates the following: 25 candidates.
Show them anyway? (y/n).**no**
Ready to proceed?**ok**
(rule 12) will be ignored in remaining evaluations.
Remaining in contention are: 26 candidates.
Show them anyway? (y/n).**n**
Ready to proceed?**y**

Another spontaneous comment will appear. The evaluation logic will save the comment in a file, as before. Selectmicro continues in this vein, asking questions and eliminating candidates until only a single candidate remains, whereupon it sends a signal to trigger mid-run questionnaires. The evaluation logic will check whether there are such questionnaires to be given to this type of user. It finds and administers Questionnaire Two.

For your needs, is the following an essential requirement?
(base configuration must be less than $4000)
comment The program seems to be much faster now.
ep Comment noted. Please reply to the original question.

yes
(rule 33) eliminates the following: 14 candidates.

.
. List of candidates and following dialogue omitted.
.
Remaining in contention are: 1 candidates.

Ready to proceed?**y**
No further candidates to evaluate.

ep Questionnaire 2.
ep question 1.
ep The noise level here is:
ep a Much too quiet.
ep b Somewhat more quiet than I would like.
ep c Just about ideal for my purposes.
ep d Somewhat more noisy than I would like.
ep e Much too noisy.
e
ep auxiliary question 1.
ep What is the main source of the noise?
ep a Computer equipment.
ep b Other equipment.
ep c Voices.
ep d Traffic or other outside noise.
ep e Other sources.
a
ep auxiliary question 1.
ep Which piece of equipment is mainly responsible for the noise?
ep a Printer(s).
ep b Keyboard(s).
ep c Terminal hum or whine.
ep d Other sources.
a
ep question 2.
ep The temperature here is:
ep a Much too cold.
ep b Somewhat more cold than I would like.
ep c Just about ideal for this kind of work.
ep d Much too warm.
c
ep question 3.
ep The lighting here is:
ep a Much too bright or glaring.
ep b Somewhat more bright or glaring than I would like.
ep c Just about right for this kind of work.
ep d Somewhat more dim or dark than I would like.
ep e Much too dim or dark.
c
ep End of Questionnaire 2.

Control returns to Selectmicro. The user chooses to exit the program rather than to explore any of its other features. Just before Selectmicro exits, it signals the evaluation logic to give any end-of-run questionnaires. Questionnaire Three is appropriate for this type of user, so it is presented to him.

Ready to proceed?**ok**
You can help improve this program, if you like.
Just add more knowledge, to help the next user.
Have you knowledge to add?**ok**
Add a candidate?**no**
Add a rule?**no**
Add an explanation?**no**
View user history/statistics?**no**
Save status and return to UNIX?**ok**.
.
. Some detail of Selectmicro omitted here.
.
ep Questionnaire 3.
ep question 1.
ep What has been the best feature of the system you have just been using?
ep Just enter a single-line description of the feature.
The number of computers represented is very impressive.
ep question 2.
ep What has been the worst feature of the system you have just been using?
ep Just enter a single-line description of the feature.
The program was very slow in the beginning.
ep question 3.
ep Generally, how would you rate the system's response times?
ep a Much too slow.
ep b Somewhat more slow than I would like.
ep c Just about ideal as far as speed goes.
ep d Somewhat faster than I was comfortable with.
ep e Much too fast for my needs.
b
ep question 4.
ep Did the system meet your original expectations?
yes
ep Please expand your answer - just a line to clarify it.
I had no strong expectations. The system seems pretty good.
ep question 5.
ep Would you use the system again?
no
ep Please expand your answer - just a line to clarify it.
I got the information I needed. I don't need the program again.
ep question 6.
ep Would you recommend the system to a friend?
yes
ep question 7.
ep Did answering these questions interfere much with your use of the program?
no

ep End of Questionnaire 3.
Program selectmicro completed.
%

These examples have shown how questionnaires can be developed and administered by the evaluation system. The questionnaires used were deliberately kept small for illustration purposes. The standard questionnaires actually developed for use with the system are considerably more detailed. Figure 4.1 shows highlights of their composition. The complete contents of the standard questionnaire file EVSTDQUEST appear in Appendix D. The following fragment of dialogue shows how a designer can focus on details of a questionnaire. The designer's responses are shown in **bold face** type.

Enter number of questionnaire to focus on:
Use 0 if none, use 999 to add a new questionnaire. **6**
Focus on which aspect of questionnaire 6?...
(h)eader, (c)ategories, user (e)xperience, user (t)ype,
(d)elete entire questionnaire, (r)eplicate this questionnaire
to make the nucleus of a new one, or (n)one. **c**
The following categories are present in questionnaire:
Remarks re ease of learning.

Number	Header
1	Easy to learn?

Enter number of category to focus on:
Use 0 if none, use 999 to add a new category.**1**
Focus on which aspect of category 1?...
(h)eader, (q)uestions, (d)elete entire category,
(r)eplicate this category to make the nucleus of a
new one, or (n)one. **q**
The following questions are present in category:
Easy to learn?

Number	Header	Auxiliaries?	User Comment?	Answer Values
1	How often would you say you have used this application system?	yes	no	yes

Enter number of question to focus on:
Use 0 if none, use 999 to add a new question.**1**
Focus on which aspect of question 1?...
(h)eader, (b)ody, (c)omment flag, (a)uxiliaries,
(p)ermissible answer values,
(d)elete entire question, (r)eplicate this question
to make the nucleus of a new one, or (n)one. **a**
These auxiliary questions are present for question 1.

```
Category    Category
Number      Header
1           Ease of learning.

Enter number of auxiliary category to focus on:
Use 0 if none, use 999 to add a new auxiliary.1
Focus on which aspect of auxiliary category 1?...
(c)ategory contents, (a)nswer triggers,
(d)elete entire auxiliary, (r)eplicate this auxiliary
to make the nucleus of a new one, or (n)one. c
Focus on which aspect of category 1?...
(h)eader, (q)uestions, (d)elete entire category,
(r)eplicate this category to make the nucleus of a
new one, or (n)one. q
The following questions are present in category:
Ease of learning.

Number  Header                                  Auxiliaries?  User Comment?  Answer Values
1       How easy or difficult was this system to learn?   no            yes            yes

Enter number of question to focus on:
Use 0 if none, use 999 to add a new question.0
Focus on which aspect of category 1?...
(h)eader, (q)uestions, (d)elete entire category,
(r)eplicate this category to make the nucleus of a
new one, or (n)one. n
```

The designer chose not to look further at the body of question one, which laid out the forced choice list of responses permitted to the user. Note that the user will be asked to comment additionally after giving his forced choice response to this question.

Figure 4.1 Contents of Standard Questionnaires

Number	Header	User Characteristics	
		Experience	Type
1	Describe users	(first_time)	(all)
2	User state on startup	(first_time repeat)	(all)
3	Describe the technical user.	(first_time)	(designer programmer)
4	Naive user expectations.	(first_time)	(enduser buyer)
5	Reasons for system re-use.	(repeat)	(all)
6	Remarks re ease of learning.	(repeat)	(enduser)
7	User remarks re errors.	(first_time repeat)	(enduser)
8	Technical user status report.	(repeat)	(designer programmer)
9	Buyers' opinions.	(first_time repeat)	(buyer)
10	Opinions of knowledge sources.	(first_time repeat)	(knowledgesource)
11	Summary of user impressions	(first_time repeat)	(all)

CHAPTER 5

USER-TESTING THE EVALUATION SYSTEM

5.1. Introduction

Once a collection of standard questionnaires was developed, it was appropriate to test the evaluation system with real users and a real application program. Selectmicro was chosen as the application vehicle because it was of manageable size, and was believed to be error-free and capable of attracting potential users. The plan was to invite users to take a 'free-trial' run of Selectmicro without particularly drawing their attention to its evaluation component, in order to determine whether:

users would willingly answer the questionnaires;

users would give reasonable answers, rather than 'clowning' with the system;

users would object to the questionnaire contents or style;

users would take advantage of the 'comment' feature;

the evaluation logic would interfere with application program performance, and

whether the evaluation logic itself was error-free.

It was assumed that favorable answers to these questions from a substantial user sample would indicate that the automated evaluation process was essentially sound. In the event, the results obtained were very encouraging, although the number of users recruited was lower than had been anticipated. In the following discussion, the reader should keep in mind the fact that results obtained in these limited tests may or may not be generalizable to other user populations.

5.2. Method

5.2.1. User Subjects

User subjects were recruited from 'night-school' computer students at the British Columbia Institute of Technology (BCIT). Most of these students were taking their first computer course (Introduction to Data Processing) although some were taking computer language courses. During the week of June 11, 1984 instructors read to their classes the notice shown in figure 5.1.

Figure 5.1. Notice Used to Recruit User-subjects.

THINKING OF BUYING A MICROCOMPUTER?

I am a BCIT day-school instructor in the computer technology, presently on leave to finish a Master's degree in Computing Science. I have written a program which helps consumers choose the best microcomputer for their needs. The program will eventually be marketed by a local firm.

One feature of the program is that it learns to behave more intelligently and respond faster the more often it is used. I need some people to run the program so that it will 'mold itself' to the behavior of real users, and so I can have user opinions of the program as part of my thesis research.

In exchange for your answering a few questions, you can get some free advice on which computer(s) might be best for you from among the hundred or so models known to the program. I will be making the program available at BCIT on Saturday June 16, Sunday June 17, and Monday evening, June 18.

Interested? Just fill in the signup sheet outside room 2n331, then come to 2n331 at the scheduled time. If you want to try the program but the times provided are not convenient, leave a note in my assignment box outside 2n331, indicating your preferences. If there is enough interest, I will try to schedule another session. Thanks for your help!

Fred Martin.

Fourteen people signed up to try the program. In the event, two people did not appear, one other was crowded out of his scheduled time slot by another person running late, and one person asked to try the program although he had not filled in the signup form. Thus, a total of twelve people actually used the program for the tests. A brief subject profile is shown in figure 5.2, using data obtained by the evaluation program itself.

As shown, the subjects ranged in age from 21 to 40, with a mean age of 30.9. Three were female and nine were male.

5.2.2. Procedure

A Hazeltine 1500 CRT terminal was set up at a desk in an office module at BCIT. The terminal was connected by a 1200-baud modem to a VAX computer at SFU. While I started the program and activated the UNIX SCRIPT logging facility, a user was asked to read and sign a form entitled "Conditions for free trial use of Selectmicro program". I then explained that the program would "ask some questions, such as your age and sex, in order to get a profile of our users." I keyed in the user's evaluation identification code (a simple code based on the day of the week and the order of the users' arrival on that day), and then yielded the terminal to the user, after pointing out some of its keyboard idiosyncra-

Figure 5.2. Profile of User-subjects.

Age	Sex	Highest Completed Education Level
36	m	M Sc.
40	m	High School
31	m	Technical Diploma - Electronics
28	m	High School - Many post-secondary courses, but no degree
26	m	Grade School - "Did not finish university"
27	f	High School - various post-secondary courses, but no degree
34	m	Undergraduate degree - Architecture, construction management
37	m	Technical Diploma
34	f	Accounting Certification
30	f	High School - some university courses
21	m	High School
27	m	High School - attended first year university

cies. I watched the user for a few moments, and made sure that the user understood the comment feature. I then retreated, saying that I would be available if there were any problems.

5.3. Results

All twelve users completed the test, including all the evaluation questionnaires. No user expressed a desire to break off the process before it ended (not even the user who was accompanied by a spouse and two-year-old son). Two users explicitly asked whether they could try the program again when they had more time. There was no evidence of flippant or insincere answers to the questions. No users found any of the questions annoying or offensive. Two of the twelve users stated that answering the questionnaires interfered with their use of the Selectmicro program. Five of the twelve users took advantage of the comment feature. Eight of the twelve thought the comment feature worthwhile, with the other four not giving an opinion about it. No user criticised the comment feature. No errors were discovered within the evaluation logic, however some problems with Selectmicro were unearthed.

In addition to the results summarized above which address the major questions asked at the outset of this chapter, the evaluation program produced a wide variety of other data. Some of this information is of general interest, some is relevant to the designer of Selectmicro, and some hints at possible avenues of future research. Figure 5.3 summarizes the key data. A brief expansion of this information follows.

Eleven of the twelve users reported that they had received a "casual, informal explanation" of the system before they used it. The twelfth user reported that he had received "no briefing at all."

Predictably, none of the users perceived computers as "a threat." Two saw computers as "a fact of modern life which you must become used to", eight saw them as "a useful tool", while the remaining two saw them as "a great boon to mankind."

Figure 5.3. Summary of Key User Responses from Test Runs

User id.	sa1	sa2	sa3	su1	su2	su3	su4	su5	mo1	mo2	mo3	mo4	% yes
Would recommend	y	y	y	y	y	y	y	y	y	y	y	y	100
Need for system	y	y	y	y	y	y	y	y	y	y	y	y	100
Temperature ok	y	y	y	y	y	y	y	y	y	y	y	y	100
Seating ok	y	y	y	y	y	y	y	y	y	y	y	y	100
Workspace ok	y	y	y	y	y	y	y	y	y	y	y	y	100
Easy to use	y	y	y	n	y	y	y	y	y	y	y	y	92
Info. complete	y	y	y	y	y	y	y	y	y	n	y	y	92
Noise level ok	y	y	y	y	n	y	y	y	y	y	y	y	92
Lighting ok	n	y	y	y	y	y	y	y	y	y	y	y	92
Fast enough	y	n	y	n	y	y	y	y	y	y	y	y	83
Would use often	y	n	y	n	y	y	y	y	n	y	y	y	75
Accurate	n	n	y	n	y	y	y	y	y	y	y	y	75
User goal met	y	n	n	n	y	y	y	y	y	n	y	y	67
Used comment	n	n	y	y	n	y	n	n	y	y	n	n	42
Excess jargon	n	n	n	n	n	y	n	n	n	n	y	n	17
Evaluation interfered	y	n	n	y	n	n	n	n	n	n	n	n	17
Mood change	n	n	n	n	y	n	n	n	n	n	n	n	8
Physical change	n	n	n	n	n	n	n	n	y	n	n	n	8
Offended	n	n	n	n	n	n	n	n	n	n	n	n	0

Ten of the twelve users reported their mood as "relaxed, contented or happy." The other two were "nervous, apprehensive, or anxious."

Curiously, nine of the twelve users reported that they would use Selectmicro often, if it were available.

All twelve users said they would recommend Selectmicro to a friend.

Ten of the twelve reported that the system's response times were fast enough for their needs. The program was run on a timesharing system, with occasional interference from other users.

Eleven of the twelve found the system "easy to use."

Two of the twelve thought Selectmicro used technical jargon unnecessarily.

Questions about documentation quality and quantity produced generally favourable responses. These answers were presumably based on the on-screen 'documentation' which Selectmicro uses, since these users had been shown no hard-copy program documentation.

In the first user sessions of Saturday, June 16, two of the three users reported that they did not believe Selectmicro's results were accurate. Selectmicro obtains its knowledge about microcomputers from users, and someone had told it that a particular computer cost over $4000 but did not cost over $3000. This rather glaring error was corrected after the Saturday sessions. In the following sessions, eight of the nine remaining users reported that the system was accurate, as far as they knew.

Eleven of the twelve users reported that Selectmicro's data was complete, as far as they knew.

Only eight users reported that they had met their goal during their session with Selectmicro. This result is interesting given that all twelve users said that they would recommend the program to a friend.

Predictably, all twelve users believed that there was a need for a program such as Selectmicro.

User predictions as to the expected future life of Selectmicro varied, but eight of the twelve predicted a life of "more than three years" for it.

Users generally agreed that the test site was satisfactory. Eleven were satisfied with the noise levels and lighting, and all twelve found the temperature, seating and workspace satisfactory. (There had been no attempt to make any of these features less than satisfactory).

One user reported a mood change between the beginning and end of his test session. His mood improved.

One user reported a change in physical condition during the session, namely the disappearance of allergy symptoms believed to have been caused by paper-handling before the session began.

A summary of timing information obtained from the evaluation timing file appears as figure 5.4. The total time spent in system pauses ranged from 1040 seconds for the fastest user to 2864 seconds for the slowest, and the total time spent in user pauses ranged from 324 seconds to 1934 seconds. The longest user session was 4995 seconds (an hour and 23 minutes), while the shortest session was 2013 seconds (about 33.5 minutes).

Various other results were obtained which addressed the perceived quality of Selectmicro at a level of detail which is inappropriate for discussion here. Readers wanting more detailed information should contact the author, since the entire user dialogues were logged and can be retrieved if necessary.

5.4. Discussion of User Tests

5.4.1. Comments on Results of Interest

The tests reported do not investigate the effectiveness of the questionnaires aimed only at repeat-users, nor those designed for programmers, buyers, expert knowledge-

Figure 5.4. Timing Results from User Runs

User ID.	System Pauses				User Pauses				Overall
	Count	Min	Max	Total	Count	Min	Max	Total	
sa1	2086	0	121	1956	113	0	98	728	3271
sa2	1071	0	131	1093	68	0	92	598	2276
sa3	1368	0	97	1040	75	0	70	431	2013
su1	1077	0	187	2749	117	0	277	1934	4995
su2	866	0	151	1291	65	0	80	790	2999
su3	1359	0	136	2864	89	0	125	769	4343
su4	1787	0	863	2749	84	0	113	910	4671
su5	2678	0	324	1957	117	0	48	324	3074
m01	2580	0	417	1915	93	0	66	415	2981
m02	1270	0	236	2033	89	0	70	554	3046
m03	1560	0	320	2264	84	0	64	487	3319
m04	1706	0	267	2019	84	0	72	502	3146

(All times given are in seconds.)

sources, and so forth. It is also possible that user opinions were coloured by the sheer novelty of the situation. However, within the important but limited context of first-time end-users of a system, the results suggest that automated evaluation and quality assessment procedures are feasible, that they will be accepted by users, will be treated seriously even when users are given a minimum of direction, and that they need not interfere significantly with a user's main goals in interacting with an application system. The data obtained from user comments led directly to detecting an error in Selectmicro as well as some more subtle weaknesses in its performance (such as its lack of information on low-priced 'home' computers).

Interestingly, the users were very tolerant of the physical site conditions where the tests occurred. There was considerable glare from overhead fluorescent lights which only one user found objectionable. Similarly, the keyboard was much too high for comfortable long-term use, but this was not mentioned as a problem by any of the users, even though many of them were touch-typists. I speculate that these motivated but 'casual' users would have very different site requirements from full-time users such as data-entry personnel.

One of the twelve users (su1) appeared to have a very different orientation from the rest. Perhaps significantly, he had considerable computing experience. He used the comment feature extensively, entering spontaneous comments after almost every question. He was one of the three users who would not use Selectmicro often if it were available; one of two who did not find the system fast enough; the only one who did not find the system easy to use; the only one who did not find Selectmicro accurate after the $3000/$4000 error (discussed previously) was corrected; one of the four users who did not meet their goal, and one of the two users who found that answering the questionnaires interfered with his use of Selectmicro. His session was the longest, and was almost 2.5 times the length of the shortest user session. I suggest that system designers who ask opinions con-

cerning system quality on an occasional basis rather than systematically, might receive such skewed responses and draw inappropriate conclusions about their systems, particularly if they solicit opinions from fellow computer personnel, rather than from a 'real' end-user population.

It is difficult to know why nine of the twelve users reported that they would use Selectmicro often if it were available, since it is not the kind of program which merits frequent use. Possibly, users did not attend to the word 'often' and were just trying to convey positive feelings about the program. On the other hand, most Selectmicro users do want to cycle through the program several times, changing their answers to its questions in order to see the effects. Perhaps they interpret this as using the program often. This explanation is supported by the fact that two of the users spontaneously asked for future appointments so they could try the program again.

It is interesting that none of the users found any of the questions offensive or annoying, even though some of the questions were rather intrusive (eg age, mood, physical state). I take this result to be supportive of the argument that the automated evaluation process used is basically sound. However, more tests with other users would help increase the confidence level with which this claim could be made. I speculate that users would have answered far more personal questions if asked [Milgram, 1963], and I suggest that professional computing societies define bounds to deter overzealous system designers from asking inappropriate questions in their evaluation systems. The questions regarding changes in mood or physical state were intended for situations in which users remained at a terminal for some time and might be inclined to report problems arising from stress or poor ergonomics. The Selectmicro tests were probably too brief to uncover any such patterns.

The 'spontaneous user comment' feature was well-received and in my opinion should be a standard component of interactive systems. The eight user replies to the question "What do you think of the comment feature?" were (verbatim): useful if required;

good; great idea; excellent idea; good idea; great; OK; and useful. Again, further tests with other users would indicate whether this favorable pattern is robust.

5.4.2. Some Apparent Correlations in the Results

As indicated earlier, the evaluation system produces a wealth of data which it does not attempt to analyze, since analysis is often best left to human interpretation. One obvious analysis technique is to subject various data of interest to scrutiny using a statistical evaluation package, such as SPSS. Some SPSS correlations from the present results are offered here as samples of the avenues which could be explored. Normally, it is of course inappropriate to go on a 'fishing expedition' looking for correlates among large numbers of data elements since many of the apparent correlations will be solely due to chance. This is particularly true for the relatively small sample size (n=12) of the present tests. Nonetheless, any apparent correlations may serve to stimulate development of new hypotheses under which the correlations can be more rigorously tested. With these cautions in mind, the reader may find the selected correlations in figure 5.5 of interest. They are offered merely to indicate how evaluation data might be used. Complete correlation results for probabilities no greater than p=0.05 using Pearson's r test are shown in Appendix C. Alphabetic data were converted to interval data where possible to provide a basis for correlation calculations.

Figure 5.5. Correlations of Possible Interest from Test Results

Item	Correlated With (n=12)		
Perceived accuracy	ease of use 0.5222 p=0.041	speed adequate 0.7746 p=0.002	would use often 0.5556 p=0.030
Age	education 0.5476 p=0.033	excessive jargon -0.5926 p=0.021	
Interference	lighting adequate -0.6742 p=0.008		
Sex	Used comment feature 0.6831 p=0.007		

Those users who believed Selectmicro to be accurate also tended to find it easy to use, to find its speed adequate, and to say they would use it often if it were available. This suggests (but certainly does not prove) the possibility of a 'halo effect'. Perhaps if a user is pleased with the results of some key variable, the satisfaction may spill over and colour his overall impressions of the system. Such a hypothesis would be worth exploring.

Age is correlated well with level of education in this student sample, as one might expect. Interestingly, older users tended not to see Selectmicro as using jargon unnecessarily compared to younger users. This could suggest that older users have larger vocabularies or that they may have a higher tolerance for technical terms. The latter notion may be worth investigation, particularly since all the users sampled were adults and one might not have predicted any particular pattern on these variables.

The one user who found the lighting inadequate also found that answering the evaluation questionnaires interfered with his use of Selectmicro. Despite the impressive correlation value (0.6742 $p=0.008$), these numbers are probably due to chance and serve to illustrate the perils of 'fishing expeditions' to look for correlations. Nonetheless, they raise the interesting issue of the extent to which a negative factor outside a system proper (such as glaring lights) could colour a user's opinion of a system to the extent that he would not use, buy, or recommend it. Such questions definitely deserve investigation.

Finally, all the female users tried out the comment feature, while many males did not. If this result were replicable over other trials, it would prove interesting.

CHAPTER 6

CONCLUSIONS

6.1. Answers to Questions

Among the questions raised at the outset of this project was the issue of whether evaluation logic can be built into an application system such that evaluation information is gathered whenever the system is used. Clearly this is the case, at least within the context of the test results of Chapter five.

Next was the question of whether there are categories of information which should be gathered for any and all application programs. The questionnaires illustrated in Chapter five are a first attempt to define such information. Extended use with a variety of application systems would no doubt reveal areas needing improvement.

On the issue of portability of evaluation code, the question is only partially answered. The present evaluation programs can be used easily with any application system written in Franzlisp. For other UNIX-supported languages, appropriate interfaces would have to be written. If a designer does not need support for the timing, mid-run questionnaire, and user-comment features, he can run small stand-alone LISP programs to generate start-of-run questionnaires before his application run and end-of-run questionnaires after it. For non-UNIX systems, other LISP dialects are sufficiently similar to Franzlisp that conversion should not be too onerous. In the worst case, for systems that do not support LISP at all, at least the ideas remain portable.

The question was posed as to whether the evaluation logic could be 'turned on itself' to gather user opinions of the evaluation program. To some extent, this occured during the tests of Chapter five. There seems no doubt that such testing is feasible.

As expected, the results shown in Chapter five demonstrate the potential for using the evaluation system to investigate various hypotheses concerning user behavior and system performance. Naturally, this one small-scale test of a single application only hints at the kinds of evaluation which could be done on a larger scale and with a variety of applications.

6.2. Known Limitations and Problems

While developing the standard questionnaires the delete feature was used only rarely. Only as the project neared completion was it noticed that a delete operation sometimes leaves a null-list 'residue' of the deleted object. When correcting this bug, provision should be made for renumbering lists of numbered items to avoid gaps in numbering after a deletion.

The prompt messages that lead a designer through creation of auxiliary questions are still more awkward than they should be. Some cosmetic program changes are in order.

The idea of capturing user comments in a separate file needs refinement. Users have a tendency to enter comments with no contextual explanation. A comment such as "Oops, I meant to answer number 3." is distressingly uninformative unless one refers to the log file from the user run.

The resolution used in the timing logic is at the level of a single second. Much finer resolution would be needed for some applications, such as comparing user performance on different keyboard layouts. In the same vein, provision should be made in the questionnaire answer file for storing the length of time a user took in answering a question.

Care should be taken to ask for user advice on correcting problems. This is done only occasionally with the present questionnaires.

6.3. Practical Applications

Obviously, the system is designed with practical implications in mind, and many have been implied throughout this paper. One promising application area which is somewhat oblique to the main function of the system consists of using the evaluation system on an application program which is essentially empty. The evaluation system could then metamorphose into a pure questionnaire administrator and could be used in a variety of situations, such as for giving psychological tests, performance tests, student examinations, job application fact-gathering, opinion polling, and so forth.

Another tangential area of application is to use the evaluation system as a performance measurement tool for data-entry personnel. Promotions or productivity bonuses could be made contingent upon processing a certain number of transactions at a predefined rate and accuracy level, as measured by the evaluation system. Of course, such applications can easily be misused and must be approached carefully.

In time, it may become possible to predict a user's future performance on a system based on early use patterns, as recorded by the evaluation system. Such information could be used in hiring or training environments.

6.4. Directions for Further Work

Obviously, both the form and contents of the questionnaires should be subject to refinement and extension. Which questions do designers find most useful over time? Which qustions (if any) come to annoy users over time? Some possibilities for alterations to question form were cited in section 3.2.1.2. The validity and reliability of many of the questions should be investigated using tools such as factor analysis [Gorsuch, 1974], so that the questionnaires will gradually be refined into truly scientific measures.

Despite the remarks in section 3.2.1.3, which explain why the present implementation uses a 'glass Teletype' format of user interaction, future versions of the system should

use modern terminal features such as mice and windowing where feasible. This could help smooth the interface between the evaluation system and the application under evaluation and minimize distractions caused by the evaluation questions.

A set of strategy guidelines should be developed for application designers who want to use the system. Possibly a cyclical approach would be appropriate, whereby designers ask many questions for a new system or a new user group, then correct problems, remove most of the questions until new problems are reported, add questions to help pin down the new problems, correct these problems, remove the extra questions, and so forth.

Thought should be given to the best ways of integrating evaluation logic with existing Job Accounting and Project Control systems. To what extent do these systems overlap? How can all appropriate information be gathered without requiring users to enter the same data repdundantly?

The present feature for tallying error messages can easily be extended to allow tallying of any arbitrary features of an application program, such as use of Help commands, or access to particular records or instructions.

Consideration should be given to having the evaluation system sense a user's skill level and report it to the application program so that the latter may change the level of detail in its prompts or menus.

The basic tools developed in this project can be used to explore any number of research areas. A few were outlined at the end of section 2.4. Others include examination of the effects of ecological variables which influence user behavior. Moos [1973] cites many such variables, including temperature, barometric pressure, cyclonic and anticyclonic storm patterns, and oxygen, nitrogen, carbon dioxide, and ozone concentrations in the atmosphere.

Temporal patterns could also be investigated. Is productivity of computer users lower on Mondays and Fridays than at midweek? Are there productivity cycles during the day?

If so, what variables can be manipulated to optimize such patterns? To what extent does the presence of the evaluation system cause a Hawthorne effect? No doubt the reader can imagine many other such applications.

APPENDIX A

PROGRAM LISTINGS

1.1. Program Structure - Role of Major Modules

INITEVAL	
EVDESIGNER	EVUSER

EVDESIGNER	
PHASE1	Initialize
PHASE2	Set questionnaire contents
PHASE3	Set timing data
PHASE4	Set end of run trigger
PHASE5	Set user comment handling
PHASE6	Set automatic data gathering

EVUSER	
INITQUESTIONNAIRES	Get questionnaires from disk file
GIVEQUAIRES	Give appropriate questionnaires to users

1.2. Evmain

```
; ------------------------------------------------------------
; This file contains the 'main' portion of the evaluation system.
; Specifically, it includes the function 'initeval' which is invoked to set
;  evaluation parameters and/or to do evaluations.
; It also includes the functions which are common to both 'designer mode' and
;  to 'user mode'.
;
; The rest of the evaluation system is contained in the two files 'evdesigner'
;  and 'evuser'.
;
; If you want to read the source code to follow the program logic, the best
;  place to start is the function 'initeval' in this listing.
;
; All functions are listed alphabetically by name.
;
; ------------------------------------------------------------
;
; addaquote returns a list containing a copy of the input,
;         preceded by the word 'quote'.
(defun addaquote (someexpr)
  (list 'quote someexpr))
```

```
; anstriggers returns a copy of the answer_triggers portion of an auxiliary.
(defun anstriggers (anaux)
  (lastelement anaux))


; catheader returns a copy of the header portion of a category.
(defun catheader (acategory)
  (caadr acategory))


; catnumber returns a copy of the number of a category.
(defun catnumber (acategory)
  (car acategory))


; catquests returns a copy of the questions portion of a category.
(defun catquests (acategory)
  (lastelement (lastelement acategory)))


; cleanread reads from the keyboard and returns either a single atom, or
; a list of the words which make up a user response.
; It exists so that multi-word entries can be made without the user worrying
; about double quotes or parentheses.
(def cleanread
  (lambda nil
          (prog (temp)
                (setq temp (readsymbols))
                (cond ((greaterp (length temp) 1) (return temp))
                      (t (return (car temp)))))))


; clearruntiming sets the various run timing properties of 'evalstuff' to
;  prepare for timing a new run.
(defun clearruntiming ()
  (putprop 'evalstuff (systime) 'sessionstart)
  (putprop 'evalstuff (list 0 999999 0 0) 'systempauses)
  (putprop 'evalstuff (list 0 999999 0 0) 'userpauses)
  (remprop 'evalstuff 'lastread)
  (remprop 'evalstuff 'lastwrite))


; diskin is a simplified version of the MTS diskin function
; side-effect: sets the value of global variable inport.
(defun diskin (filename)
  (cond ((probef filename)
         (setq inport (infile filename))
         (load filename)
         (close inport))
        (t nil)))


; evdesigner sets up an evaluation environment for later use by application pgm.
; inputs: none
; outputs: various files are set up
(defun evdesigner ()
  (msg "This program helps evaluate other programs." N)
  (msg "Loading rest of program now." N)
  (load 'evdesigner)
  (msg "Do you want instructions? y/n: ")
  (cond ((yesorno (cleanread)) (printinstructions)))
  (setq evalstuff (phase1)))
```

```
(setq evalstuff (phase2))
(setq evalstuff (phase3 questionnaires))
(setq evalstuff (phase4))
(setq evalstuff (phase5))
(setq evalstuff (phase6)))


; evuser handles the actual giving of questionnaires to users.
; input: the file name where 'evalstuff' has been saved.
; returns: nil
; side-effect: 1. sets the global variable 'evnewuser'.
;              2. starts the run timing clock after giving the startup
;                 questionnaire(s).
(defun evuser (filename)
 (prog (tempevalstuff)
       (msg "Please wait. Loading rest of program." N)
       (load 'evuser)
       (setq evalstuff nil)
       (load filename)
       (setq tempevalstuff nil)
       (inittes)
       (setplist 'evalstuff (plist 'tempevalstuff))
       (clearruntiming)
       (load (get 'evalstuff 'questfilename))
       (initquestionnaires)
       (msg "Please enter your 'evaluation identification code'." N
            "If you do not have such a code, then enter your name." N)
       (setq evuserid (stringread))
       (cond
        ((checkuser evuserid)
         (olduserblurb) (setq evnewuser nil)
         (givequaires 's 'repeat))
        (t
         (newuserblurb) (setq evnewuser t)
         (givequaires 's 'first_time)))))


; explain a term to a user, or, if not possible, record request for explanation
(defun explain (term)
 (prog (anexplanation)
       (setq anexplanation (assoc term explanations))
       (cond (anexplanation (msg "According to "
                                 (nthelement anexplanation 3) "," N
                                 (nthelement anexplanation 2) N))
             (t (msg "I cannot yet explain " term "." N)
                (msg "I've noted your request and I'll see what I can find out." N)
                (msg "Meanwhile, please continue as best you can." N)
                (setq needsexplaining (append1 needsexplaining (list term userid)))))))


; getquairetiming returns the questionnaire timing value on the property
; 'timingstuff' in the variable 'evalstuff' for a particular questionnaire.
(defun getquairetiming (quairenumber)
 (prog (answer)
       (mapc
        (function (lambda (element)
                                (cond ((equal (car element) quairenumber)
                                       (setq answer element)))))
           (get 'evalstuff 'timingstuff))
       (return answer)))
```

```
; implodesymbol takes a list of single-character fixnums, and implodes it
;  into either a symbol or an integer.
; NOTE: decimal 45 is a minus sign.
;       decimal 43 is a plus sign.
;       decimal 48 to 57 are ASCII 0 to 9 inclusive.
(defun implodesymbol (symlist)
  (prog (negative possibleint intvalue)
        (setq possibleint symlist)
        signloop
        (cond ((equal (car possibleint) 45) (setq negative (toggle negative))
              (setq possibleint (cdr possibleint))
              (cond (possibleint (go signloop))
                    (t (return (implode symlist)))))
             ((equal (car possibleint) 43) (setq possibleint (cdr possibleint)))
              (cond (possibleint (go signloop))
                    (t (return (implode symlist))))))
        (setq intvalue 0)
        intloop
        (cond (possibleint
              (cond ((or (lessp (car possibleint) 48)
                         (lessp 57 (car possibleint)))
                     (return (implode symlist))))
              (setq intvalue (plus (times 10 intvalue)
                                   (diff (car possibleint) 48)))
              (setq possibleint (cdr possibleint))
              (go intloop)))
        (cond (negative (return (minus intvalue)))
              (t (return intvalue)))))


; initeval is the main evaluation program.
; It can be used by a system designer on a stand-alone basis to set up an
; evaluation environment, or it can be called by an application program in
; order to interact with the evaluation environment, administer questionnaires,
; and the like.
;
; input: nil, if used as a stand-alone program to set up the environment.
;        otherwise, the name of an evaluation data file.
; output: various files are written - see documentation for details.
(defun initeval (filename)
  (msg "Initeval. © Copyright 1984, Fred J. Martin" N)
  (cond ((null filename) (evdesigner))
        (t (evuser filename)))
  (saveevalstuff)
  "Initeval completed.")


; intersect returns the intersection of two sets.
; source: Touretzky, 1984, speeded up by F. Martin.
; It is used to simulate the corresponding MTS function.
(defun intersect (set1 set2)
  (prog (result)
        (cond ((lessp (length set2) (length set1)) (return (intersect set2 set1))))
        loop
        (cond ((null set1) (return result)))
              ((member (car set1) set2)
               (setq set2 (remove (car set1) set2 1))
               (setq result (append1 result (car set1)))))
        (setq set1 (cdr set1))
        (go loop)))


; lastelement returns a copy of the last element of a list.
(defun lastelement (alist)
```

```
  (car (last alist)))

; msglist prints a list without the clutter of parentheses
(defun msglist (alist)
 (prog ()
       loop
       (cond (alist
            (cond ((listp alist)
                 (cond ((listp (car alist)) (msglist (car alist)))
                       (t (msg (car alist) " ")))
                 (setq alist (cdr alist))
                 (go loop))
                (t (msg alist))))
           (t nil))))

; nthelement returns the nth element of a list
; using a count starting at 1 (not at zero).
(def nthelement
 (lambda (alist position)
  (cond ((lessp position 1) nil) (t (car (nthmts alist position))))))

; nthmts simulates the mts nth function, which differs from that of Franzlisp
(def nthmts
 (lambda (l n)
  (cond ((lessp n 1) (last l))
        ((greaterp n (length l)) nil)
        (t (nthcdr (sub1 n) l)))))

; onoroff returns "on" if its input is non-nil, "off" otherwise.
(defun onoroff (input)
 (cond (input "on")
       (t "off")))

; pleaseanswer prompts user for a yes or no reply and accepts reply
(defun pleaseanswer ()
  msg "Please answer 'yes' or 'no' to the original question." N)
  (yesorno (cleanread)))

; quaireaud returns a copy of the audience information portion of a
;  questionnaire.
(defun quaireaud (aquaire)
  (cadr aquaire))

; quaireaudexp returns the audience experience value from a questionnaire.
(defun quaireaudexp (aquaire)
 (car (quaireaud aquaire)))

; quaireaudtype returns the audience_type value(s) from a questionnaire.
(defun quaireaudtype (aquaire)
 (cadr (quaireaud aquaire)))

; quairecats returns the categories portion of a questionnaire.
(defun quairecats (aquaire)
 (lastelement (lastelement aquaire)))
```

```
; quaireheader returns the header of a questionnaire
(defun quaireheader (aquaire)
  (caaddr aquaire))


; quairenumber returns the number of a questionnaire
(defun quairenumber (aquaire)
  (car aquaire))


; questansvals returns a copy of the answer_values portion of a question.
(defun questansvals (aquestion)
  (lastelement (questinfo aquestion)))


; questaux returns a copy of the aux_questions portion of a question.
(defun questaux (aquestion)
  (lastelement aquestion))


; questcomment returns a copy of the comment_flag portion of a question.
(defun questcomment (aquestion)
  (nthelement (questinfo aquestion) 3))


; questdetail returns a copy of the detail_lines portion of a question.
(defun questdetail (aquestion)
  (cadr (questinfo aquestion)))


; questheader returns a copy of the header of a question.
(defun questheader (aquestion)
  (car (questinfo aquestion)))


; questinfo returns a copy of the question_info portion of a question.
(defun questinfo (aquestion)
  (cadr aquestion))


; questnumber returns the number of a question.
(defun questnumber (aquestion)
  (car aquestion))


; readline brings in a line from the keyboard.
; and returns a list of the numeric equivalents of each ascii character read.
; NOTE: 10 decimal is ASCII linefeed.
(def readline
  (lambda nil
          (prog (input result)
                (drain piport)
                loop (setq input (tyi))
                (cond ((equal input 10) (return result))
                      (t (setq result (append1 result input)) (go loop))))))


; readnumber reads a response from the keyboard and ensures that it is numeric.
(defun readnumber ()
  (prog (reply)
        loop
        (setq reply (read))
        (cond ((numberp reply) (return reply)))
```

```
        (msg N reply " is not a number. Please reenter your answer." N)
        (go loop)))

; readsymbols brings in a line from the keyboard and converts it to a
; list of symbols
; NOTE: This function should be cleaned up some time - use mapcar.
;       32 decimal is ASCII space.
(def readsymbols
  (lambda nil
          (prog (workword workline result)
                (setq workline (append1 (readline) 32))
                loop (cond ((null workline) (return result))
                           ((equal (car workline) 32)
                            (setq result
                                  (append1 result (implodesymbol workword)))
                            (setq workline (cdr workline))
                            (setq workword nil))
                           (t (setq workword (append1 workword (car workline)))
                              (setq workline (cdr workline))))
                (go loop))))

; removenth removes the nth element of a list, returning a copy of the altered
; list.
(defun removenth (alist count)
  (remq 'evdeleted (replacenth alist count 'evdeleted) 1))

; replacenth replaces the nth element of a list with a new element
; and returns a copy of the modified list.
; side-effect: if there are fewer than n elements it the list, the list is
;   extended with null elements to the desired length.
(defun replacenth (alist count newelement)
  (cond ((lessp (length alist) count)
         (replacenth (append1 alist nil) count newelement))
        (t
         (mapcar (function (lambda (listelement)
                             (setq count (sub1 count))
                             (cond ((equal count 0) newelement)
                                   (t listelement)))) alist))))

; saveevalstuff creates a function 'inittes' which can be used in future runs
; to initialize the property list of 'tempevalstuff'. The function captures
; the property list of 'evalstuff'.
; When 'inittes' is created, it is saved in the file specified for holding
; run independent system evaluation data.
(defun saveevalstuff ()
  (eval (list 'defun 'inittes nil
              (list 'setplist (addaquote 'tempevalstuff) (addaquote (plist 'evalstuff)))))
  (updateok (get 'evalstuff 'evalfilename) 'inittes))

; seenenough allows viewer to pause as screen scrolls
; and cancel a display if it is not of interest
; input: none
; output: t if viewer has seen enough, nil otherwise
(defun seenenough ()
  (msg "Press s to stop the display, any other key to see more.")
  (cond ((equal 's (stringread)) t)
```

```
        (t nil)))


; setdifference returns the difference between two sets.
;  Specifically, the remainder of the first set (x), after the second
;  set (y) is subtracted from it.
; Source: Touretzky.
(defun setdifference (x y)
  (prog (result)
        loop
        (and (null x) (return (reverse result)))
        (or (member (car x) y)
            (setq result (cons (car x) result)))
        (setq x (cdr x))
        (go loop)))


; stringread reads data from the keyboard until a linefeed (decimal 10).
; returns: a copy of the input, as a symbol.
;        Thus, a user need not key double quotes (decimal 34).
; note: uses implode rather than maknam to build its string, thus the symbol
;       created is intern'd.
(defun stringread ()
  (implode (readline)))


; toggle sets a flag to t if it is nil, and to nil if it is t.
(defun toggle (flag)
  (cond (flag nil)
        (t t)))


; union returns the union of two sets.
; Source: Touretzky.
(defun union (x y)
  (append x (setdifference y x)))


; update is a variant of the MTS update function.
; it writes a single function to a file.
; The file may contain no other material.
; If the file already exists, the user must confirm that the update is ok.
(defun update (filename funcname)
  (cond ((probef filename)
         (msg "File " filename " exists. OK to overwrite? (y/n) ")
         (cond ((yesorno (cleanread)) (updateok filename funcname))
               (t (msg N "Enter new filename, or nil to abandon update. ")
                  (setq filename (cleanread))
                  (cond (filename (update filename funcname))))))
        (t (updateok filename funcname))))


; updatea appends an s-expression to a file.
;  If the s-expression if a function definition, it is pretty-printed,
;   otherwise princ is used.
(defun updatea (filename sexpr)
  (prog (outport)
        (setq outport (outfile filename 'append))
        (cond ((and (symbolp sexpr) (getd sexpr))
               (eval (list 'pp '|P| outport) sexpr))
              (t (eval (list 'princ (addaquote sexpr) outport))))
        (close outport)))
```

```
; updateok pretty prints a function to a file
;  or uses princ to print other s-expressions to a file.
; It is safest if called by update.
(defun updateok (filename funcname)
  (prog (outport)
        (setq outport (outfile filename))
        (cond ((and (symbolp funcname) (getd funcname))
               (eval (list 'pp '(P outport) funcname)))
              (t (eval (list 'princ (addaquote funcname) outport))))
        (close outport)))


; waitforyes waits until a user replies 'yes' (as filtered through 'yesorno').
(defun waitforyes ()
  (cond ((yesorno (cleanread)) t)
        (t (msg "Reply 'yes' when ready.")
           (waitforyes))))


; yesorno gets user replies to questions and allows user to exit program.
; inputs: the user response to some question
; outputs: returns t if a user response is affirmative,
;        returns nil if a user response is negative
; side effects/misc notes: If user response is break, invokes lisp break
; function, then asks for yes or no reply to original question
;          : if user response is exit, allows user to exit the program
;          : if user response is help, presents a menu to user and recurses
;          : if user response is explain x, tries to explain x to the user
;          : other user responses cause a warning message and a recurse
(def yesorno
  (lambda (reply)
          (cond ((member reply
                         '(y yes ok Y YES OK please PLEASE t T true TRUE)) t)
                ((member reply '(n no N NO nil NIL f F false FALSE)) nil)
                ((and (listp reply) (equal (car reply) 'explain))
                 (explain (cdr reply))
                 (pleaseanswer))
                ((equal reply 'break) (break) (pleaseanswer))
                ((equal reply 'exit)
                 (msg "Really? Leave the program and return to UNIX?")
                 (cond ((yesorno (cleanread))
                        (msg "Save current changes before leaving?")
                        (cond ((yesorno (cleanread)) (shutdown)) (t (exit))))
                       (t (pleaseanswer))))
                ((or (equal reply 'help) (equal reply '?))
                 (msg "To proceed, choose one of the following:" N)
                 (msg "Reply 'yes' or 'no' to the original question;" N)
                 (msg "Reply '(explain x)' if you want me to explain something;" N)
                 (msg "Reply 'exit' to leave lisp and go back to UNIX;" N)
                 (msg "Reply 'break' to activate the lisp break function." N)
                 (yesorno (cleanread)))
                (t (msg "Please reply yes or no") (yesorno (cleanread))))))
```

1.3. Evdesigner

```
; ------------------------------------------------------------
; This file contains the portion of the evaluation system which is
; used only in 'designer mode'. To follow the program logic, look at
; phase1, phase2, ... phase6 (all of which are called from the main
; portion of the program.. see file evmain).
```

```
;
; All functions are listed alphabetically by name.
;

; addacategory appends a new category to a list of categories.
; input: a list of categories built so far.
; output: a copy of the input, with a new category appended at the end.
; side effects / misc. notes: none.
(defun addacategory (categories)
 (cond ((null categories) (setq categorynumber 1))
       (t (setq categorynumber (add1 (caar (last categories))))))
 (append1 categories (list categorynumber (addcategoryinfo))))

; addallauds finds and saves the set of all audiences for all existing
;  questionnaires.
; side-effect: sets the property 'audiences' on the variable 'evalstuff'.
(defun addallauds (aquaire)
 (putprop 'evalstuff
       (union (get 'evalstuff 'audiences)
              (quaireauditype aquaire))
       'audiences))

; addananswerrange obtains an answer range from the user.
; input: none
; output: an answer range.
; side effects / misc. notes: prompts user for response.
(defun addananswerrange ()
 (prog (reply)
       (msg "(s)ingle value, (r)ange of values, or (y)es/no pair?")
       (setq reply (stringread))
       (cond ((equal reply 's) (msg "Please enter value." N)
              (setq reply (cleanread))
              (return (list reply reply)))
             ((equal reply 'r) (msg "Please enter low value. Use 'nil' if none.")
              (setq reply (cleanread))
              (msg "Please enter high value. Use 'nil' if none.")
              (return (list reply (cleanread))))
             ((equal reply 'y) (return (list 'yes 'no)))
             (t (return (addananswerrange))))))

; addanauxquestion builds an auxiliary.
; input: none.
; output: a list containing an auxiliary.
; side effects / misc. notes: none.
(defun addanauxquestion ()
 (list (car (addacategory nil)) (addanswertriggers nil)))

; addanswertriggers builds the 'answer triggers' portion of an auxiliary.
```

```
; input: a list of answer triggers built so far.
; output: a copy of the input, with the new trigger(s) appended at the end.
; side effects / misc. notes: recurses until user wants no more.
(defun addanswertriggers (answertriggers)
  (msg "Add an answer value to trigger this category? (y/n)")
  (cond ((yesorno (stringread))
         (setq answertriggers (append1 answertriggers (addananswerrange)))
         (cond ((equal (lastelement answertriggers) '(yes no))
                (setq answertriggers
                    (append1 (replacenth answertriggers
                                         (length answertriggers)
                                         '(yes yes))
                             '(no no)))))
         (addanswertriggers answertriggers))
        (t answertriggers)))


; addanswervalues builds the 'answer values' component of a question.
; input: a list of answer values built so far.
; output: a copy of the input, with the new value(s) appended at the end.
; side effects / misc. notes: recurses until user wants no more.
(defun addanswervalues (ansvals)
  (msg "Add an allowable answer for this question? (y/n)")
  (cond ((yesorno (stringread))
         (setq ansvals (append1 ansvals (addananswerrange)))
         (cond ((equal (lastelement ansvals) '(yes no))
                (setq ansvals (append1 (replacenth ansvals (length ansvals)
                                                   '(yes yes))
                                       '(no no)))))
         (addanswervalues ansvals))
        (t ansvals)))


; addaquestion appends a new question to a list of questions.
; input: a list of questions
; output: a copy of the input, with a new question appended at the end.
; side effects / misc. notes: a question may itself contain questions.
(defun addaquestion (questions)
  (prog (tempquestion questionnumber)
        (cond ((null questions) (setq questionnumber 1))
              (t (setq questionnumber (add1 (caar (last questions))))))
        (msg "Question number: " questionnumber N)
        (setq tempquestion (list questionnumber (addquestioninfo) nil))
        (return (append1 questions (addauxquestions tempquestion)))))


; addaquestionnaire adds a questionnaire to a list of questionnaires.
; input: a list of questionnaires built so far.
; output: a copy of the input, with a new questionnaire appended at the end.
; side effects / misc. notes: none.
(defun addaquestionnaire (questionnaires)
  (cond ((null questionnaires) (setq questionnairenumber 1))
        (t (setq questionnairenumber (add1 (caar (last questionnaires)))))))
```

```
  (msg "Questionnaire number " questionnairenumber N)
  (append1 questionnaires (list questionnairenumber
                                (addaudienceinfo) (addquestionnaireinfo))))

; addaudexp builds the audience experience portion of a questionnaire.
; input: a dummy variable to avoid the need of local variables.
; output: a list of audience experience elements.
(defun addaudexp (reply)
  (msg "Should questionnaire be given to (f)irst-time users, to" N
       "(r)epeat users, or to (b)oth? ")
  (setq reply (stringread))
  (cond ((equal reply 'f) '(first_time))
        ((equal reply 'r) '(repeat))
            ((equal reply 'b) '(first_time repeat))
            (t (addaudexp nil))))


; addaudienceinfo builds the audience info portion of a questionnaire.
(defun addaudienceinfo ()
  (list (addaudexp nil) (addaudtypes)))


; addaudsubgroup builds audience subgroup information.
; input: none.
; output: a list containing the audience type subgroups.
(defun addaudsubgroup ()
  (prog (reply)
        loop
        (msg "Please identify all the subgroups required." N
             "Enter only the first letter of the standard subgroups," N
             "but enter the entire name of any nonstandard subgroups." N
             "Separate your entries with spaces." N
             "Standard subgroups are: (d)esigner, (b)uyer, (p)rogrammer, (e)nd user," N
             "and (k)nowledge-source." N)
        (setq reply (cleanread))
        (cond ((atom reply) (setq reply (list reply))))
        (return (mapcar
                  (function (lambda (relem)
                                (cond ((equal relem 'd) 'designer)
                                      ((equal relem 'b) 'buyer)
                                      ((equal relem 'p) 'programmer)
                                      ((equal relem 'e) 'enduser)
                                      ((equal relem 'k) 'knowledgesource)
                                      (t relem))))) reply))))


; addaudtypes builds the audience type portion of a questionnaire
; input: none
; output: a list of audience type information
; side effects/misc notes: prompts user for information
(defun addaudtypes ()
```

```
(prog (reply)
        loop
        (msg "What is the intended target audience type for this questionnaire?_." N
            "(a)ll system users, (n)o audience type known yet, or (s)pecific subgroups.")
        (setq reply (stringread))
        (cond ((equal reply 'a) (return '(all)))
              ((equal reply 'n) (return nil))
              ((equal reply 's) (return (addaudsubgroup)))
              (t (go loop)))))


; addauxquestions builds the 'auxquestions' portion of a question.
; input: the list of 'auxquestions' built so far.
; output: a copy of the input, with the new auxiliaries appended at the end.
; side effects / misc. notes: recurses until the user wants no more.
(defun addauxquestions (aquestion)
 (msg "Current question is:" N " " (questheader aquestion) "_." N)
 (msg "Add an auxiliary question category to this question? (y/n)")
 (cond ((yesorno (stringread))
        (setq aquestion (list (car aquestion) (cadr aquestion)
                              (append1 (lastelement aquestion)
                                       (addanauxquestion))))
        (msg "You may add another auxiliary category now, if you want." N)
        (addauxquestions aquestion))
       (t
        (msg "Finished with auxiliary question category for question " N " "
            (questheader aquestion) "." N)
        (msg N "Reverting to higher level." N) aquestion)))


; addcategories builds the 'categories' portion of a questionnaire.
; input: a list of 'questionnaire info'.
; output: a copy of the input, with new categories appended at the end.
; side effects / misc. notes: recurses until user requires no more.
(defun addcategories (questionnaireinfo)
 (msg
  "Add a category to questionnaire " (car questionnaireinfo) "? (y/n)")
 (cond ((yesorno (stringread))
        (setq categories (addacategory (cadr questionnaireinfo)))
        (addcategories (list (car questionnaireinfo) categories)))
       (t questionnaireinfo)))


; addcategoryinfo builds the 'categoryinfo' component of a questionnaire.
; input: none
; output: a list of category information.
; side effects / misc. notes: prompts user for information.
(defun addcategoryinfo ()
 (msg "Please enter category description." N)
 (addquestions (list (stringread) nil)))


; addcomphrcat returns a new comment phrase category to be added to a list
```

```
; of existing ones. (It does not actually do the addition).
(defun addcomphrcat (phrases)
 (prog (catnumber)
        (cond ((null phrases) (setq catnumber 1))
              (t (setq catnumber (add1 (car (lastelement phrases))))))
        (msg "comment phrase category: " catnumber N)
        (return (list catnumber (getphrcathdr) (addphrases nil)))))


; adddetaillines obtains details of a question from the user.
; input: a list of the detail lines produced so far.
; output: a copy of the input, with the new line appended at the end.
; side effects / misc. notes: recurses until user indicates no more
;      lines are required.
(defun adddetaillines (detaillines)
 (msg "Enter next line. Reply 'nil' if no more lines." N)
 (cond ((equal 'nil (setq detailline (stringread)))
        detaillines)
       (t (adddetaillines (append1 detaillines detailline)))))


; addphrases adds phrases to a comment phrase category body.
(defun addphrases (phrcatbody)
 (prog (reply)
       loop
       (msg "Enter a new phrase or (n)o more phrases." N)
       (setq reply (stringread))
       (cond ((equal reply 'n) (return phrcatbody)))
       (setq phrcatbody (append1 phrcatbody reply))
       (go loop)))


; addquestioninfo builds the 'info' component of a question.
; input: none
; output: a list of questioninfo
; side effects / misc. notes: dialog with user to obtain information
(defun addquestioninfo ()
 (msg "Please enter the first line of the question." N)
 (setq questionheader (stringread))
 (setq detaillines (adddetaillines nil))
 (msg "Ask respondent to comment on answer? (y/n)")
 (list questionheader detaillines (yesorno (stringread))
       (addanswervalues nil)))


; addquestionnaireinfo builds the 'info' component of a questionnaire.
; input: none.
; output: a list of questionnaireinfo.
; side effects / misc. notes: prompts user for information.
(defun addquestionnaireinfo ()
 (msg "Please enter questionnaire description." N)
 (addcategories (list (stringread) nil)))
```

```
; addquestions builds the 'questions' component of a category.
; input: a list of the category info built so far.
; output: a copy of the input, with new questions appended at the end.
; side effects / misc. notes: recurses until user requires no more questions.
(defun addquestions (categoryinfo)
  (msg "Add a question to category " (car categoryinfo) "? (y/n)")
  (cond ((yesorno (stringread))
         (setq questions (addaquestion (cadr categoryinfo)))
         (addquestions (list (car categoryinfo) questions)))
        (t categoryinfo)))


; ansfileinit gets the name of a file to save user answers to questions.
; input: a dummy variable for holding user replies.
; returns: the name of the answer file.
(defun ansfileinit (reply)
  (msg "Save questionnaire answers on what file?" N
       "Enter filename or choose (s)tandard file default." N)
  (setq reply (cleanread))
  (cond ((equal reply 's) (setq reply 'evstdans)))
  (cond ((null (probef reply)) reply)
        (t (msg reply " already exists: (u)se it anyway or (n)ew filename? ")
           (cond ((equal (stringread) 'u) reply)
                 (t (ansfileinit reply))))))


; commentfate determines the fate of a user comment.
; returns: nil
; side effect: may update the property list of 'evalstuff'
(defun commentfate ()
  (prog (reply)
        loop
        (msg "Should user comments be (i)ntercepted by the evaluation system,"
             N "or (f)orwarded to your application program? ")
        (setq reply (stringread))
        (cond ((equal reply 'i) (putprop 'evalstuff 'intercept 'commentfate))
              ((equal reply 'f) (putprop 'evalstuff 'forward 'commentfate))
              (t (msg "Inappropriate reply." N)
                 (go loop)))
        (msg "Save user comments on a separate file? (y/n) ")
        (cond ((yesorno (stringread)) (msg "Please enter filename: ")
               (putprop 'evalstuff (getfilename) 'commentfilename))
              (t (remprop 'evalstuff 'commentfilename)))))


; copyaux allows an auxiliary question to be copied to provide the nucleus
;   of a new auxiliary.
; inputs: the auxiliary to be copied, and the list of auxiliaries to which
;         the new copy is to be appended.
; output: a copy of the auxiliaries with the new auxiliaries appended.
;         The new auxiliary is given a category number that is one greater
;         than that of the last existing auxiliary.
```

```
(defun copyaux (anaux auxes)
 (prog (newnumber)
        (setq newnumber (add1 (catnumber (car (lastelement auxes)))))
        (msg "Copying auxiliary number " (catnumber (car anaux))
          " to auxiliary number " newnumber "." N)
        (msg "Remember to modify one of the copies before you leave the program." N)
        (return (append1 auxes (cons (cons newnumber (cdr (car anaux)))
                                     (cdr anaux))))))


; copycat allows a category to be copied to provide the nucleus of a new
;  category.
; inputs: the category to be copied, and the list of categories to which the
;       new copy is to be appended.
; output: a copy of the categories with the new category appended at the end.
;           The new category is given a category number that is one greater than
;       the number of the last existing category.
(defun copycat (acat categories)
 (prog (newnumber)
        (setq newnumber (add1 (catnumber (lastelement categories))))
        (msg "Copying category number " (catnumber acat)
          " to category number " newnumber "." N)
        (msg "Remember to modify one of the copies before you leave the program." N)
        (return (append1 categories (cons newnumber (cdr acat))))))


; copyphrase allows a phrase to be copied to provide the nucleus of a new one.
; inputs: the phrase to be copied and the list of phrases to which the new
;       copy is to be appended.
; output: a copy of the phrases with the new phrase appended at the end. The
;       new phrase is given a phrase category number that is one greater
;       than the number of the last existing phrase.
(defun copyphrase (phrcat comphrases)
 (prog (newnumber)
        (setq newnumber (add1 (car (lastelement comphrases))))
        (msg "Copying comment phrase category number " (car phrcat)
          " to category number " newnumber "." N)
        (msg "Remember to modify one of the copies before you leave the program." N)
        (return (append1 comphrases (cons newnumber (cdr phrcat))))))


; copyquaire allows a questionnaire to be copied to provide the nucleus of a
; new questionnaire.
; inputs: the questionnaire to be copied, and the list of questionnaires to
;       which the new copy is to be appended.
; output: a copy of the questionnaires with the new questionnaire appended at
;       the end. The new questionnaire is given a question number that is
;       one greater than the number of the last existing questionnaire.
(defun copyquaire (aquaire quaires)
 (prog (newnumber)
        (setq newnumber (add1 (quairenumber (lastelement quaires))))
```

```
        (msg "Copying questionnaire number " (quairenumber aquaire)
            " to questionnaire number " newnumber "." N)
        (msg "Remember to modify one of the copies before you leave the program." N)
        (return (append1 quaires (cons newnumber (cdr aquaire))))))


; copyquest allows a question to be copied to provide the nucleus of a new
;   question.
; inputs: the question to be copied, and the list of questions to which the
;       new copy is to be appended.
; output: a copy of the questions with the new question appended at the end.
;           The new question is given a question number that is one greater than
;       the number of the last existing question.
(defun copyquest (aquest questions)
  (prog (newnumber)
        (setq newnumber (add1 (questnumber (lastelement questions))))
        (msg "Copying question number " (questnumber aquest)
            " to question number " newnumber "." N)
        (msg "Remember to modify one of the copies before you leave the program." N)
        (return (append1 questions (cons newnumber (cdr aquest))))))


; failpassword returns t if a user cannot provide a correct password within
; some number of attempts. Otherwise it returns nil.
(defun failpassword (enough)
  (cond ((zerop enough) t)
        (t (cond ((equal (stringread) (get 'tempevalstuff 'password)) nil)
                 (t (msg N "Invalid password. Please try again." N)
                    (failpassword (sub1 enough)))))))


; failsecurity returns t if a user 'fails a security test', nil otherwise.
; To pass the test, the user must be recognized as a valid system evaluator,
; the application name entered by the user must be correct, and the user
; must be able to supply a correct password within three attempts.
(defun failsecurity (userid)
  (prog (reply)
        (cond ((null (member userid (get 'tempevalstuff 'evaluators))) (return t))
              ((null (equal (get 'evalstuff 'application)
                            (get 'tempevalstuff 'application))) (return t))
              (t (msg "Please enter password. ")
                 (return (failpassword 3))))))


; focusansrange modifies or examines an answer range.
(defun focusansrange (atrig)
  (prog (reply)
        loop
        (msg "Answer range values presently read: " N
            "Low value")
        (tab 15)
```

```
        (msg "High value" N)
        (msg (car atrig))
        (tab 15)
        (msg (lastelement atrig) N)
        (msg "(r)eplace these values, (d)elete line, or (n)o change? ")
        (setq reply (stringread))
        (cond ((equal reply 'r) (setq atrig (addananswerrange))
              (go loop))
              ((equal reply 'd) (msg "Deleting trigger " atrig N)
              (return nil))
              ((equal reply 'n) (return atrig))
              (t (msg "Inappropriate reply." N)
                 (go loop)))))


; focusanstrigs*builds, modifies, or examines the answer triggers portion
;   of an auxiliary.
; input: an auxiliary.
; returns: a copy of the answer triggers, with any changes incorporated.
(defun focusanstrigs (anaux)
 (prog (reply worktrig anstrigs linecount)
        (setq anstrigs (anstriggers anaux))
        loop
        (msg "The following answer ranges are present for" N
             "auxiliary category " (catnumber (car anaux)) N)
        (msg "Single-value answer ranges are shown as ranges with identical high and low values." N N)
        (msg "Number Low value")
        (tab 25)
        (msg "High value" N)
        (setq linecount 1)
        (mapc (function (lambda (anstrig)
                                (msg linecount) (tab 8)
                                (msg (car anstrig))
                                (tab 25)
                                (msg (lastelement anstrig) N)
                                (setq linecount (add1 linecount))))
              anstrigs)
        (msg "Enter number of answer range to focus on:" N)
        (msg "Use 0 if none, use 999 to add a new answer trigger." )
        (setq reply (readnumber))
        (cond ((equal reply 0) (return
                                 anstrigs))
              ((equal reply 999) (setq worktrig nil))
              (t (setq worktrig (nthelement anstrigs reply))))
        (setq worktrig (focusansrange worktrig))
        (cond ((equal reply 999) (setq anstrigs
                                       (append1 anstrigs worktrig)))
              (t (cond
                   (worktrig
                    (setq anstrigs (replacenth
                                     anstrigs reply worktrig)))
                   (t (setq anstrigs (removenth anstrigs reply))))))
        (go loop)))
```

```
; focusaux allows a user to 'focus' on an auxiliary to examine or change it.
; input: usually, a list of auxiliaries, and the specific auxiliary to be
; focused on. If the latter is nil, focusaux assumes that the
;       user wants to add a new auxiliary.
; output: usually, a copy of the input auxiliary or the auxiliary
;         just added. If nil, tells the calling program that user wants to
;         delete the auxiliary. If t, tells the calling program that
;       the user wants it to make a copy of the auxiliary as the
;       nucleus of a new one.
(defun focusaux (auxes anaux)
 (prog (reply)
        loop
        (cond ((null anaux)
               (return (addanauxquestion))))
        (msg "Focus on which aspect of auxiliary category "
           (catnumber (car anaux)) "?..." N)
        (msg "(c)ategory contents, (a)nswer triggers," N
           "(d)elete entire auxiliary, (r)eplicate this auxiliary" N
           "to make the nucleus of a new one, or (n)one. ")
        (setq reply (stringread))
        (cond ((equal reply 'c) (setq anaux (list (focuscat nil (car anaux))
                                                  (anstriggers anaux))))
              ((equal reply 'a) (setq anaux (list (car anaux)
                                                  (focusanstrigs anaux))))
              ((equal reply 'd)
               (msg "Deleting auxiliary category "
                    (catnumber (car anaux)) "." N)
               (return nil))
              ((equal reply 'r) (return t))
              ((equal reply 'n) (return anaux))
              (t (go loop)))
        (go loop)))


; focuscat allows a user to 'focus' on a category to examine or change it.
; input: usually, a list of categories, and the specific category to be
; focused on. If the latter is nil, focuscat assumes that the
;       user wants to add a new category.
; output: usually, a copy of the input category or the category
;         just added. If nil, tells the calling program that user wants to
;         delete the category. If t, tells the calling program that
;       the user wants it to make a copy of the category as the
;       nucleus of a new one.
(defun focuscat (categories acategory)
 (prog (reply)
        loop
        (cond ((null acategory)
               (return (lastelement (addacategory categories)))))
        (msg "Focus on which aspect of category "
           (catnumber acategory) "?..." N)
        (msg "(h)eader, (q)uestions, (d)elete entire category." N
           "(r)eplicate this category to make the nucleus of a" N
           "new one, or (n)one. ")
```

```
        (setq reply (stringread))
        (cond ((equal reply 'h) (setq acategory (focuscathdr acategory)))
              ((equal reply 'q) (setq acategory (focuscatquestions acategory)))
              ((equal reply 'd)
               (msg "Deleting category "
                    (catnumber acategory) N)
               (return nil))
              ((equal reply 'r) (return t))
              ((equal reply 'n) (return acategory))
              (t (go loop)))
        (go loop)))


; focuscathdr allows a user to examine and/or modify a category header.
(defun focuscathdr (acategory)
 (prog (reply)
        loop
        (msg "Header presently reads:" N (catheader acategory) N)
        (msg "Enter new value or (n)o change." N)
        (setq reply (stringread))
        (cond ((equal reply 'n) (return acategory)))
        (setq acategory (list (catnumber acategory)
                              (list reply (catquests acategory))))
        (go loop)))


; focuscatquestions builds, modifies, or examines the questions portion
;  of a category.
; input: a category.
; returns: a copy of the input, with any changes incorporated.
(defun focuscatquestions (acategory)
 (prog (reply workquest copyflag questions)
        (setq questions (catquests acategory))
        loop
        (msg "The following questions are present in category:" N
           (catheader acategory) N)
        (tab 63)
        (msg "User")
        (tab 72)
        (msg "Answer" N)
        (msg "Number Header")
        (tab 50)
        (msg "Auxiliaries? Comment?")
        (tab 72)
        (msg "Values" N)
        (mapc 'showquests questions)
        (msg "Enter number of question to focus on:" N)
        (msg "Use 0 if none, use 999 to add a new question." )
        (setq reply (readnumber))
        (cond ((equal reply 0) (return
                                 (list (catnumber acategory)
                                       (list (catheader acategory)
                                             questions))))
```

```
        ((equal reply 999)(setq workquest nil))
        (t (setq workquest (nthelement questions reply))))
    (setq copyflag (focusquest questions workquest))
    (cond ((null (equal copyflag 't)) (setq workquest copyflag)))
    (cond ((equal reply 999) (setq questions
                                (append1 questions workquest)))
        (t (cond
            ((equal copyflag 't)
             (setq copyflag nil)
             (setq questions
                    (copyquest workquest questions)))
            (workquest
             (setq questions (replacenth
                                questions reply workquest)))
            (t (setq questions (removenth questions reply))))))
    (go loop)))


; focuscommentphrases builds, modifies, or examines comment_phrases.
; input: a copy of the phrases presently existing.
; returns: a copy of the input, with any changes incorporated.
(defun focuscommentphrases (phrases)
 (prog (reply workphrcat copyflag)
        loop
        (msg "The following phrase categories presently exist:" N)
        (msg "Number Header Body" N)
        (mapc
         (function (lambda (phrasecat)
                        (msg (car phrasecat))
                        (tab 8)
                        (msg (cadr phrasecat))
                        (tab 15)
                        (msg (lastelement phrasecat) N))) phrases)
        (msg "Enter number of phrase category to focus on:" N)
        (msg "Use 0 if none, use 999 to add a new phrase category. ")
        (setq reply (readnumber))
        (cond ((equal reply 0) (return phrases))
            ((equal reply 999) (setq workphrcat nil))
            (t (setq workphrcat (nthelement phrases reply))))
        (setq copyflag (focuscomphrcat phrases workphrcat))
        (cond ((null (equal copyflag 't)) (setq workphrcat copyflag)))
        (cond ((equal reply 999) (setq phrases
                                    (append1 phrases workphrcat)))
            (t (cond
                ((equal copyflag 't)
                 (setq copyflag nil)
                 (setq phrases
                        (copyphrase workphrcat phrases)))
                (workphrcat
                 (setq phrases (replacenth
                                    phrases reply workphrcat)))
                (t (setq phrases (removenth phrases reply))))))
        (go loop)))
```

```
; focuscomphrcat allows a user to 'focus' on a comment phrase category to
; examine or change it.
; input: a list of phrases (used by addcomphrcat) and (usually)
;       a phrase category. If the latter is nil, focuscomphrcat assumes that
;       the user wants to add a new phrase category.
; output: usually, a copy of the input or the phrase category
;             just added. If nil, tells the calling program that user wants to
;             delete the phrase category. If t, tells the calling program that
;       the user wants it to make a copy of the phrase category as the
;       nucleus of a new one.
(defun focuscomphrcat (phrases aphrcat)
 (prog (reply)
         loop
         (cond ((null aphrcat) (return (addcomphrcat phrases))))
         (msg "Focus on which aspect of comment phrase category "
             (car aphrcat)
             "?_." N)
         (msg "(h)eader, (b)ody, (d)elete entire comment phrase category," N
             "(r)eplicate this category to make the nucleus of a new one," N
             "or (n)one. ")
         (setq reply (stringread))
         (cond ((equal reply 'h) (setq aphrcat (focusphrcathdr aphrcat)))
              ((equal reply 'b) (setq aphrcat (focusphrcatbody aphrcat)))
              ((equal reply 'd)
               (msg "Deleting category "
                       (car aphrcat) N)
               (return nil))
              ((equal reply 'r) (return t))
              ((equal reply 'n) (return aphrcat))
              (t (go loop)))
         (go loop)))


; focusdetline modifies or examines a detail line of a question.
(defun focusdetline (adetline)
 (prog (reply)
         loop
         (msg "Line presently reads: " N adetline N)
         (msg "Enter new value, (d)elete line, or (n)o change." N)
         (setq reply (stringread))
         (cond ((equal reply 'd) (msg "Deleting line " adetline N)
               (return nil))
              ((equal reply 'n) (return adetline))
              (t (setq adetline reply)
                    (go loop)))))


; focusphrase allows a user to 'focus' on a phrase to examine or change it.
; input: a phrase.
; output: usually, a copy of the input category.
;             If nil, tells the calling program that user wants to
;             delete the phrase.
(defun focusphrase (aphrase)
```

```
 (prog (reply)
        loop
        (msg "Phrase currently is " aphrase "." N
             "Enter new value, (n)o change, or (d)elete the phrase." N)
        (setq reply (stringread))
        (cond ((equal reply 'd) (return nil))
              ((equal reply 'n) (return aphrase))
              (t (setq aphrase reply)
                 (go loop)))))


; focusphrcatbody builds, modifies, or examines the body of a comment
;  phrase category.
; input: a phrase category.
; returns: a copy of the input, with any changes incorporated.
(defun focusphrcatbody (aphrcat)
 (prog (count reply workphrase phrcatbody)
        (setq phrcatbody (lastelement aphrcat))
        loop
        (setq count 0)
        (msg "Present values are:" N)
        (msg "Number")
        (tab 8)
        (msg "Phrase" N)
        (mapc (function (lambda (phrase)
                                (setq count (add1 count))
                                (msg count)
                                (tab 8)
                                (msg phrase N))) phrcatbody)
        (msg "Enter number of phrase to focus on:" N)
        (msg "Use 0 if none, use 999 to add a new phrase." )
        (setq reply (readnumber))
        (cond ((equal reply 0) (return
                                (list (car aphrcat)
                                      (cadr aphrcat)
                                      phrcatbody)))
              ((equal reply 999)
               (msg "Please enter new phrase ")
               (setq phrcatbody (append1 phrcatbody (stringread)))
               (go loop)))
        (setq workphrase (focusphrase (nthelement phrcatbody reply)))
        (cond (workphrase (setq phrcatbody (replacenth phrcatbody reply workphrase)))
              (t (setq phrcatbody (removenth phrcatbody reply))))
        (go loop)))


; focusphrcathdr allows one to examine and/or modify a phrase category header.
(defun focusphrcathdr (aphrcat)
 (prog (reply)
        loop
        (msg "Header presently reads:" N (cadr aphrcat) N)
        (msg "Enter new value or (n)o change." N)
        (setq reply (stringread))
        (cond ((equal reply 'n) (return aphrcat)))
```

```
            (setq aphrcat (list (car aphrcat) reply
                                (lastelement aphrcat)))
            (go loop)))


; focusquaire allows a user to 'focus' on a questionnaire to examine or
; change it.
; input: usually, a questionnaire. If nil, focusquaire assumes that the
;       user wants to add a new questionnaire.
; output: usually, a copy of the input questionnaire or the questionnaire
;             just added. If nil, tells the calling program that user wants to
;             delete the questionnaire. If t, tells the calling program that
;         the user wants it to make a copy of the questionnaire as the
;         nucleus of a new one.
(defun focusquaire (aquaire)
  (prog (reply)
            loop
            (cond ((null aquaire) (return (newquestionnaire))))
            (msg "Focus on which aspect of questionnaire "
                 (quairenumber aquaire) "?.." N)
            (msg "(h)eader, (c)ategories, user (e)xperience, user (t)ype," N
                 "(d)elete entire questionnaire, (r)eplicate  this questionnaire" N
                 "to make the nucleus of a new one, or (n)one. ")
            (setq reply (stringread))
            (cond ((equal reply 'h) (setq aquaire (focusquairehdr aquaire)))
                  ((equal reply 'c) (setq aquaire (focusquairecats aquaire)))
                  ((equal reply 'e) (setq aquaire (focusquaireaudexp aquaire)))
                  ((equal reply 't) (setq aquaire (focusquaireaudtype aquaire)))
                  ((equal reply 'd)
                   (msg "Deleting questionnaire "
                        (quairenumber aquaire) N)
                   (return nil))
                  ((equal reply 'r) (return t))
                  ((equal reply 'n) (return aquaire))
                  (t (go loop)))
            (go loop)))


; focusquaireaudexp allows one to examine and/or modify the audience experience
; portion of a questionnaire.
(defun focusquaireaudexp (aquaire)
  (prog (reply audexp)
            loop
            (msg "Audience presently receiving questionnaire "
                 (quaireheader aquaire)
                 " is:" N (quaireaudexp aquaire) N)
            (msg "Set to (f)irst-timers, (r)epeat users, or (b)oth? ")
            (setq reply (stringread))
            (cond ((equal reply 'f) (setq audexp '(first_time)))
                  ((equal reply 'r) (setq audexp '(repeat)))
                  ((equal reply 'b) (setq audexp '(first_time repeat)))
                  (t (go loop)))
            (return (list (car aquaire) (list audexp (car (last (cadr aquaire)))))
```

```
                    (car (last aquaire))))))

; focusquaireaudtype allows one to examine and/or modify the audience type
; portion of a questionnaire.
(defun focusquaireaudtype (aquaire)
 (prog (reply audtype)
        (setq audtype (quaireaudtype aquaire))
        loop
        (msg "Users presently receiving questionnaire " (quaireheader aquaire)
          " are:" N audtype N)
        (msg "Any changes? (y/n) ")
        (cond ((yesorno (stringread)) (setq audtype (addaudtypes)) (go loop))
              (t (return (list (car aquaire) (list (caadr aquaire) audtype)
                               (car (last aquaire))))))))


; focusquairecats builds, modifies, or examines the categories portion
;   of a questionnaire.
; input: a questionnaire.
; returns: a copy of the input, with any changes incorporated.
(defun focusquairecats (aquaire)
 (prog (reply workcat copyflag categories)
        (setq categories (quairecats aquaire))
        loop
        (msg "The following categories are present in questionnaire:" N
          (quaireheader aquaire) N)
        (msg "Number")
        (tab 8)
        (msg "Header" N)
        (mapc 'showcats categories)
        (msg "Enter number of category to focus on:" N)
        (msg "Use 0 if none, use 999 to add a new category." )
        (setq reply (readnumber))
        (cond ((equal reply 0) (return
                                 (list (quairenumber aquaire)
                                       (quaireaud aquaire)
                                       (list (quaireheader aquaire)
                                             categories))))
              ((equal reply 999) (setq workcat nil))
              (t (setq workcat (nthelement categories reply))))
        (setq copyflag (focuscat categories workcat))
        (cond ((null (equal copyflag 't)) (setq workcat copyflag)))
        (cond ((equal reply 999) (setq categories
                                       (append1 categories workcat)))
              (t (cond
                   ((equal copyflag 't)
                    (setq copyflag nil)
                    (setq categories
                          (copycat workcat categories)))
                   (workcat
                    (setq categories (replacenth
```

```
                                        categories reply workcat)))
                        (t (setq categories (removenth categories reply))))))
            (go loop)))

; focusquairehdr allows a user to examine and/or modify a questionnaire header.
(defun focusquairehdr (aquaire)
 (prog (reply)
        loop
        (msg "Header presently reads:" N (quaireheader aquaire) N)
        (msg "Enter new value or (n)o change." N)
        (setq reply (stringread))
        (cond ((equal reply 'n) (return aquaire)))
        (setq aquaire (list (quairenumber aquaire) (quaireaud aquaire)
                            (list reply (quairecats aquaire))))
        (go loop)))


; focusquaires builds, modifies, or examines questionnaires.
; input: a copy of the questionnaires presently existing.
; returns: a copy of the input, with any changes incorporated.
(defun focusquaires (questionnaires)
 (prog (reply workquaire copyflag)
        loop
        (msg "The following questionnaires presently exist:" N)
        (tab 50)
        (msg "User Characteristics" N)
        (msg "Number")
        (tab 8)
        (msg "Header")
        (tab 50)
        (msg "Experience Type" N)
        (mapc 'showquaires questionnaires)
        (msg "Enter number of questionnaire to focus on:" N)
        (msg "Use 0 if none, use 999 to add a new questionnaire. ")
        (setq reply (readnumber))
        (cond ((equal reply 0) (return questionnaires))
            ((equal reply 999) (setq workquaire nil))
            (t (setq workquaire (nthelement questionnaires reply))))
        (setq copyflag (focusquaire workquaire))
        (cond ((null (equal copyflag 't)) (setq workquaire copyflag)))
        (cond ((equal reply 999) (setq questionnaires
                                       (append1 questionnaires workquaire)))
            (t (cond
                 ((equal copyflag 't)
                  (setq copyflag nil)
                  (setq questionnaires
                        (copyquaire workquaire questionnaires)))
                 (workquaire
                  (setq questionnaires (replacenth
                                         questionnaires reply workquaire)))
                 (t (setq questionnaires (removenth questionnaires reply))))))
        (go loop)))
```

```
; focusquest allows a user to 'focus' on a question to examine or change it.
; input: usually, a list of questions, and the specific question to be
; focused on. If the latter is nil, focusquest assumes that the
;       user wants to add a new question.
; output: usually, a copy of the input question or the question
;         just added. If nil, tells the calling program that user wants to
;         delete the question. If t, tells the calling program that
;       the user wants it to make a copy of the question as the
;       nucleus of a new one.
(defun focusquest (questions aquestion
 (prog (reply)
       loop
       (cond ((null aquestion)
              (return (lastelement (addaquestion questions)))))
       (msg "Focus on which aspect of question "
            (questnumber aquestion) "?_" N)
       (msg "(h)eader, (b)ody, (c)omment flag, (a)uxiliaries," N
            "(p)ermissible answer values," N
            "(d)elete entire question, (r)eplicate this question" N
            "to make the nucleus of a new one, or (n)one. ")
       (setq reply (stringread))
       (cond ((equal reply 'h) (setq aquestion (focusquesthdr aquestion)))
             ((equal reply 'b) (setq aquestion (focusquestbody aquestion)))
             ((equal reply 'c) (setq aquestion (focusquestcomment aquestion)))
             ((equal reply 'p) (setq aquestion (focusquestansvals aquestion)))
             ((equal reply 'a) (setq aquestion (focusquestauxes aquestion)))
             ((equal reply 'd)
              (msg "Deleting question "
                   (questnumber aquestion) N)
              (return nil))
             ((equal reply 'r) (return t))
             ((equal reply 'n) (return aquestion))
             (t (go loop)))
       (go loop)))


; focusquestansvals builds, modifies, or examines the answer values portion
;   of a question.
; input: a question.
; returns: a copy of the input, with any changes incorporated.
(defun focusquestansvals (aquestion)
  (prog (reply workansval ansvals linecount)
        (setq ansvals (questansvals aquestion))
        loop
        (msg "The following answer values are present for" N
             "question " (questheader aquestion) N N)
        (msg "Single-value answer values are shown as ranges with identical high and low values." N N)
        (msg "Number Low value"
        (tab 25)
        (msg "High value" N)
        (setq linecount 1)
        (mapc (function (lambda (ansval)
                                  (msg linecount) (tab 8)
```

```
                                (msg (car ansval)) (tab 25)
                                (msg (lastelement ansval) N)
                                (setq linecount (add1 linecount))))
        ansvals)
    (msg "Enter number of answer range to focus on:" N)
    (msg "Use 0 if none, use 999 to add a new answer range." )
    (setq reply (readnumber))
    (cond ((equal reply 0) (return
                            (list (questnumber aquestion)
                                  (list (questheader aquestion)
                                        (questdetail aquestion)
                                        (questcomment aquestion)
                                        ansvals)
                                  (questaux aquestion))))
        ((equal reply 999) (setq workansval nil))
        (t (setq workansval (nthelement ansvals reply))))
    (setq workansval (focusansrange workansval))
    (cond ((equal reply 999) (setq ansvals
                                   (append1 ansvals workansval)))
        (t (cond
              (workansval
               (setq ansvals (replacenth
                              ansvals reply workansval)))
              (t (setq ansvals (removenth ansvals reply))))))
    (go loop)))


; focusquestauxes builds, modifies, or examines the auxiliaries portion
;  of a question.
; input: a question.
; returns: a copy of the input, with any changes incorporated.
(defun focusquestauxes (aquestion)
 (prog (reply workaux copyflag auxes)
       (setq auxes (questaux aquestion))
       loop
       (msg "These auxiliary questions are present for question "
          (questnumber aquestion) "." N)
       (msg "A U X I L I A R I E S" N)
       (msg "Category Category" N)
       (msg "Number   Header" N)
       (mapc 'showauxes auxes)
       (msg "Enter number of auxiliary category to focus on:" N)
       (msg "Use 0 if none, use 999 to add a new auxiliary." )
       (setq reply (readnumber))
       (cond ((equal reply 0) (return
                               (list (questnumber aquestion)
                                     (questinfo aquestion)
                                     auxes)))
          ((equal reply 999) (setq workaux nil))
          (t (setq workaux (nthelement auxes reply))))
       (setq copyflag (focusaux auxes workaux))
       (cond ((null (equal copyflag 't)) (setq workaux copyflag)))
       (cond ((equal reply 999) (setq auxes
```

```
                                        (append1 auxes workaux)))
            (t (cond
                    ((equal copyflag 't)
                     (setq copyflag nil)
                     (setq auxes
                             (copyaux workaux auxes)))
                    (workaux
                     (setq auxes (replacenth
                                     auxes reply workaux)))
                    (t (setq auxes (removenth auxes reply))))))
        (go loop)))


; focusquestbody builds, modifies, or examines the detail lines portion
;  of a question.
; input: a question.
; returns: a copy of the input, with any changes incorporated.
(defun focusquestbody (aquestion)
 (prog (reply workdetline detlines linecount)
        (setq detlines (questdetail aquestion))
        loop
        (msg "The following detail lines are present for question " N
            (questheader aquestion) N)
        (msg "Number Detail_line" N)
        (setq linecount 1)
        (mapc (function (lambda (detline)
                                (msg linecount) (tab 8) (msg detline N)
                                (setq linecount (add1 linecount))))
              detlines)
        (msg "Enter number of detail line to focus on:" N)
        (msg "Use 0 if none, use 999 to add a new detail line." )
        (setq reply (readnumber))
        (cond ((equal reply 0) (return
                                (list (questnumber aquestion)
                                    (list (questheader aquestion)
                                            detlines
                                            (questcomment aquestion)
                                            (questansvals aquestion))
                                    (questaux aquestion))))
            ((equal reply 999) (setq workdetline nil))
            (t (setq workdetline (nthelement detlines reply))))
        (setq workdetline (focusdetline workdetline))
        (cond ((equal reply 999) (setq detlines
                                        (append1 detlines workdetline)))
            (t (cond
                    (workquest
                     (setq detlines (replacenth
                                     detlines reply workdetline)))
                    (t (setq detlines (removenth detlines reply))))))
        (go loop)))
```

```
; focusquestcomment allows a user to examine and/or modify the comment_flag
; of a question.
(defun focusquestcomment (aquestion)
 (prog (commentflag)
        (setq commentflag (questcomment aquestion))
        loop
        (msg "Presently, user will ")
        (cond ((null commentflag) (msg "not ")))
        (msg "be asked to comment after giving an answer." N)
        (msg "Change this? (y/n) ")
        (cond ((yesorno (stringread)) (setq commentflag (toggle commentflag))
              (go loop))
             (t (return (list (questnumber aquestion)
                              (list (questheader aquestion)
                                    (questdetail aquestion)
                                    commentflag
                                    (questansvals aquestion))
                              (questaux aquestion)))))))


; focusquesthdr allows a user to examine and/or modify a question header.
(defun focusquesthdr (aquestion)
 (prog (reply)
        loop
        (msg "Header presently reads:" N (questheader aquestion) N)
        (msg "Enter new value or (n)o change." N)
        (setq reply (stringread))
        (cond ((equal reply 'n) (return aquestion)))
        (setq aquestion (list (questnumber aquestion)
                              (list reply (questdetail aquestion)
                                    (questcomment aquestion)
                                    (questansvals aquestion))
                              (questaux aquestion)))

        (go loop)))


; focususercomments displays and/or modifies the words which are recognized
; as triggers for user comments.
; returns nil
; side effect: sets the usercomment property of variable 'evalstuff'.
(defun focususercomments ()
 (prog (usercomments reply)
        loop
        (msg "Present trigger keywords for user comments are:" N)
        (msg (setq usercomments (get 'evalstuff 'usercomments)) N)
        (msg "Enter revised keywords, or the word 'nil' to disallow spontaneous"
             N "user comments, or (n)o change to the present list." N)
        (setq reply (cleanread))
        (cond ((equal reply 'n) (return nil))
              ((null reply) (remprop 'evalstuff 'usercomments))
              (t (putprop 'evalstuff reply 'usercomments)))
```

```
        (go loop)))

; geterrorid obtains from a designer-user the numeric value of a one-byte ascii
; character which identifies error messages from the application program to the
; end-user
(defun geterrorid (value)
 (msg "Enter numeric value of ascii character which will precede" N
      "error messages from your system to the user (0...255) ")
 (setq value (read))
 (cond ((and (signp ge value) (lessp value 256)) value)
       (t (msg "Inappropriate reply." N)
          (geterrorid value))))


; geterrortype asks a designer-user whether to tally error messages by type
(defun geterrortype ()
 (msg "Tally error messages by a type code following the error identifier? (y/n) ")
 (yesorno (stringread)))


; getfilename obtains a file name from a designer-user, warning of possible
; problems and allowing a second choice if the file already exists
(defun getfilename ()
 (prog (reply)
       loop
       (setq reply (cleanread))
       (cond ((null (probef reply)) (return reply)))
       (msg reply " already exists: (u)se it anyway or (n)ew filename? ")
       (cond ((equal 'u (stringread)) (return reply)))
       (msg "Please enter new filename." N)
       (go loop)))


; getothers builds a list of names of additional people who are authorized
; to change evaluation data.
(defun getothers (others)
 (msg "Will you allow someone else to change evaluation rules? (y/n) ")
 (cond ((yesorno (stringread))
        (msg "Please provide the name of another evaluator." N)
        (getothers (append1 others (stringread))))
       (t others)))


; getpassword obtains a password from a designer-user
(defun getpassword (password)
 (msg "Please choose a password." N)
 (setq password (stringread))
 (msg "Again please, to be sure." N)
 (cond ((equal password (stringread)) password)
       (t (msg "Sorry. That does not match the first password." N)
```

```
        (msg "Please try again." N)
        (getpassword password))))

; getphrcathdr gets a phrase category header from the user and returns it.
(defun getphrcathdr ()
  (msg "Please enter a header for this comment phrase category." N)
  (stringread))

; initquestionnaires gives the variable 'questionnaires' a value.
; input: none.
; output: does a 'setq questionnaires....'
; side effects / misc. notes: this function is hand-coded, but is replaced
; by an equivalent function made by function 'makequestionnaires'.
(defun initquestionnaires ()
  (setq questionnaires nil))

; makequestionnaires creates a function 'initquestionnaires', then runs it.
; input: none.
; output: a function, called 'initquestionnaires'.
; side effects / misc. notes: initquestionnaires, when evaluated, initializes
;     the variable 'questionnaires'.
(defun makequestionnaires ()
  (eval (list 'defun
              'initquestionnaires
              nil
              (list 'setq 'questionnaires (addaquote
                                            (focusquaires questionnaires)))))
  (initquestionnaires))

; newapplication checks whether a file exists for evaluation data for the
; present application. If so, it reads the file and loads the data in it
; onto the property list of the variable tempevalstuff.
; If no evaluation data file is present for this evaluation, it concludes
; that it is dealing with a new application and returns t.
(defun newapplication ()
  (cond ((diskin (get 'evalstuff 'evalfilename)) (inittes) nil)
        (t t)))

; newquestionnaire builds a new questionnaire.
; side effects / misc. notes: chooses a questionnaire number based on the
; number of the last existing questionnaire.
(defun newquestionnaire ()
  (cond ((null questionnaires) (setq questionnairenumber 1))
        (t (setq questionnairenumber (add1 (caar (last questionnaires))))))
  (msg "Questionnaire number " questionnairenumber N)
  (list questionnairenumber
        (addaudienceinfo) (addquestionnaireinfo)))
```

```
; phase1 allows a designer user to do several general initialization functions.
; its main effect is to fill in the property list of the variable 'evalstuff'.
(defun phase1 ()
 (prog (userid tempevalstuff)
        (msg "Phase 1: General Initialization." N)
        (msg "What is the name of the application system or program to be evaluated?" N)
        (putprop 'evalstuff (stringread) 'application)
        (msg "What file can be used to save run independent evaluation data?" N)
        (putprop 'evalstuff (getfilename) 'evalfilename)
        (msg "Please enter your name." N)
        (setq userid (stringread))
        (cond ((newapplication) (msg
                                          "This program will recognize you in future as a valid system eval
                                          "for " (get 'evalstuff 'application) "." N)
              (putprop 'evalstuff (getpassword nil) 'password)
              (putprop 'evalstuff (list userid) 'evaluators))
              (t (cond
                    ((failsecurity userid)
                     (msg "Your identification or application name was not recognized." N
                          "Check the wording and/or ask an authorized user to validate your name." N)
                     (exit))
                    (t (setplist 'evalstuff (plist 'tempevalstuff))))))
        (putprop 'evalstuff (getothers (get 'evalstuff 'evaluators)) 'evaluators)
        (msg "What file can be used to log user runs?" N)
        (putprop 'evalstuff (getfilename) 'logfilename)
        (msg "Tally error messages from your system to the user? (y/n) ")
        (putprop 'evalstuff (yesorno (stringread)) 'tallyerrors)
        (cond ((get 'evalstuff 'tallyerrors)
              (putprop 'evalstuff (geterrorid nil) 'errorid)
              (putprop 'evalstuff (geterrortype) 'tallyerrortype)))
        (msg "Phase 1 complete." N)
        (msg "Reenter Phase 1 to change anything? (y/n) ")
        (cond ((yesorno (stringread)) (phase1))
              (t nil))))


; phase2 determines the contents of questionnaires.
; side-effects: the variable 'questionnaires' may be altered.
;               the property list of 'evalstuff' is updated.
(defun phase2 ()
 (msg "Phase 2: Questionnaire Contents." N)
 (putprop 'evalstuff (questfileinit nil) 'questfilename)
 (putprop 'evalstuff (ansfileinit nil) 'ansfilename)
 (makequestionnaires)
 (questfilesave nil)
 (mapc 'addallauds questionnaires)
 (msg "Phase 2 complete." N)
 (msg "Reenter phase 2 to change anything? (y/n) ")
 (cond ((yesorno (stringread)) (phase2))
       (t nil)))
```

```
; phase3 reviews and updates timing information.
; side-effects: the property list of 'evalstuff' may be updated, specifically
;          the properties 'timingstuff' and 'dspecs.
(defun phase3 (questionnaires)
 (prog (reply)
       (msg "Phase 3: Questionnaire timing." N)
       (msg "The following questionnaires presently exist:" N)
       (msg "Number Header")
       (tab 50)
       (msg "When Given" N)
       (mapc 'showquaires2 questionnaires)
       loop
       (msg "Enter number of questionnaire to focus on:" N)
       (msg "Use 0 if none. ")
       (setq reply (readnumber))
       (cond ((null (equal reply 0))
            (settimingstuff reply)
            (go loop)))
       (putprop 'evalstuff nil 'dspecs)
       (mapc (function
            (lambda (quairetiming)
                 (putprop 'evalstuff
                          (union (get 'evalstuff 'dspecs)
                               (lastelement quairetiming))
                          'dspecs))) (get 'evalstuff
                                          'timingstuff))
       (putprop 'evalstuff (setdifference
                          (get 'evalstuff 'dspecs)
                          '(s e)) 'dspecs)
       (msg "Phase 3 complete." N)
       (msg "Reenter phase 3 to change anything? (y/n) ")
       (cond ((yesorno (stringread)) (return (phase3 questionnaires)))
            (t (return nil)))))


; phase4 determines how to recognize the end of the application program's run.
; returns nil
; side effects: sets the 'endofrun' property of variable 'evalstuff'.
(defun phase4 ()
 (msg "Phase 4: End of Run Specifications." N)
 (msg "What character will you send to signal end of run and trigger" N
      "any end-of-run questionnaires?..." N
      "Enter numeric value of ascii code (0...255) ")
 (putprop 'evalstuff (readnumber) 'endofrun)
 (msg "Phase 4 complete." N))


; phase5 determines whether and how to handle spontaneous user comments.
; returns: nil
; side-effect: may change the property list of the variable 'evalstuff'.
(defun phase5 ()
```

```
 (msg "Phase 5: Spontaneous User Comments." N)
 (focususercomments)
 (msg "Scan user comments for words or phrases of interest? (y/n) ")
 (cond ((yesorno (stringread))
        (putprop
         'evalstuff
         (focuscommentphrases (get 'evalstuff 'commentphrases))
         'commentphrases))
       (t (remprop 'evalstuff 'commentphrases)))
 (commentfate)
 (msg "Phase 5 complete." N)
 (msg "Reenter phase 5 to change anything? (y/n) ")
 (cond ((yesorno (stringread)) (phase5))
       (t nil)))


; phase6 sets automatic data gathering parameters.
; returns: nil
; side-effect: may alter the property list of the variable 'evalstuff'.
(defun phase6 ()
 (msg "Phase 6: Automatic Data Gathering Parameters" N)
 (msg "The evaluation system can time delays between user inputs and program outputs." N)
 (msg "This feature is presently set "
    (onoroff (get 'evalstuff 'timesystempauses)) "." N)
 (msg "Toggle this feature? (y/n) ")
 (cond ((yesorno (stringread))
        (putprop 'evalstuff
                 (toggle
                  (get 'evalstuff 'timesystempauses)) 'timesystempauses)
        (msg "OK" N)))
 (msg "The evaluation system can time delays between system output and user inputs." N)
 (msg "This feature is presently set "
    (onoroff (get 'evalstuff 'timeuserpauses)) "." N)
 (msg "Toggle this feature? (y/n) ")
 (cond ((yesorno (stringread))
        (putprop 'evalstuff
                 (toggle
                  (get 'evalstuff 'timeuserpauses)) 'timeuserpauses)
        (msg "OK" N)))
 (msg "The evaluation system can time the overall length of user sessions." N)
 (msg "This feature is presently set "
    (onoroff (get 'evalstuff 'timesessionlength)) "." N)
 (msg "Toggle this feature? (y/n) ")
 (cond ((yesorno (stringread))
        (putprop 'evalstuff
                 (toggle
                  (get 'evalstuff 'timesessionlength)) 'timesessionlength)
        (msg "OK" N)))
 (msg "Save timing information on what file?" N)
 (putprop 'evalstuff (getfilename) 'timingfilename)
 (msg N "Phase 6 complete. " N)
 (msg "Reenter phase 6 to change anything? (y/n) ")
 (cond ((yesorno (stringread)) (phase6))
```

```
        (t nil)))

; printinstructions tells a designer how to use the evaluation program.
; input: none
; output: t
; side effects: prints instructions on the screen.
(defun printinstructions ()
(prog ()
 (msg "Thank you for using the evaluation package." N N)
 (msg "The program you have called can be used in two ways." N)
 (msg "It can set up an evaluation environment, or conduct an evaluation." N N)
 (msg "When called with a null parameter list, (as you have just done)," N)
 (msg "it assumes that you are a system designer who will provide rules" N)
 (msg "for having your application program(s) evaluated." N N)
 (msg "This program will ask you various questions, and will use your" N)
 (msg "answers to define the evaluation criteria. Among other questions," N)
 (msg "it will ask you to provide a filename where it can store the" N)
 (msg "information you supply. After you finish the present initialization" N)
 (msg "run, you will be able to have any program evaluated by inserting" N)
 (msg "the statement '(initeval filename)' as the first instruction in" N)
 (msg "your program. The filename will be the name you choose in this run," N)
 (msg "namely the file where initeval stores the information you are about" N)
 (msg "to provide." N N)
 (cond ((seenenough) (return 't)))
 (msg N "This run consists of six distinct phases." N)
 (msg "Phase 1 does general initialization, asking you for various filenames." N)
 (msg "Phase 2 lets you browse the user questionnaires, altering them at will." N)
 (msg "Phase 3 gathers information from you about questionnaire timing." N)
 (msg "Phase 4 asks how to identify 'end of run' for your application." N)
 (msg "Phase 5 asks you about spontaneous user comments." N)
 (msg "Phase 6 allows you to specify automatic data gathering parameters." N)
 (msg N "In all cases, just reply to the prompt messages." N)
 (msg "They will lead you through all six phases." N)
 (msg N N)
 (return t)))


; quairetrig returns a list of timinginfo, with specific numeric values
; for designer-specified questionnaire triggers.
(defun quairetrig (timinginfo)
 (mapcar
  (function
   (lambda
    (timingcode)
    (cond ((equal timingcode 'd)
           (msg "What value will you send to trigger this questionnaire?..." N
                "Enter numeric value of ascii code (0..255) ")
           (readnumber))
          (t timingcode)))) timinginfo))

; questfileinit obtains the name of a file containing questionnaires, loads
```

```
; the file if it exists, and initializes the variable 'questionnaires'.
; input: a dummy variable for holding user replies.
; returns: the name of the questionnaire file.
(defun questfileinit (reply)
 (msg "What file holds the questionnaires?" N
     "Enter filename or choose (s)tandard file default." N)
 (setq reply (cleanread))
 (cond ((equal reply 's) (diskin 'evstdquest)
          (initquestionnaires) 'evstdquest)
         (t (cond ((diskin reply) (initquestionnaires) reply)
                  (t (msg "File " reply " not found." N)
                    (questfileinit reply))))))


; questfilesave obtains the name of a file where questions can be saved,
; and saves the questions on the file within the function initquestionnaires.
(defun questfilesave (reply)
 (msg "Write questionnaires to what file?" N
     "Enter filename or choose (s)tandard file default." N)
 (setq reply (cleanread))
 (cond ((equal reply 's) (setq reply 'evstdquest)))
 (eval (list 'update (addaquote reply) (addaquote 'initquestionnaires))))


; settimingstuff displays and/or modifies the 'timingstuff' for a questionnaire.
; input: the number of the questionnaire of interest.
; side-effect: may update the 'timingstuff' property of 'evalstuff'.
(defun settimingstuff (quaireno)
 (prog (reply timingstuff)
         (setq timingstuff (get 'evalstuff 'timingstuff))
         loop
         (msg "Questionnaire " quaireno " is presently given at:" N)
         (cond
          ((mapcar 'whengiven (lastelement (getquairetiming quaireno))))
          (t (whengiven nil)))

         (msg N "Give questionnaire at (s)tartup, under (d)esigner control," N
             "or at (e)nd of run?_" N
             "Enter any combination of s, d, and e, separated by spaces, or" N
             "(n)o change. ")
         (setq reply (cleanread))
         (cond ((equal reply 'n) (return))
             (t
              (cond ((atom reply) (setq reply (list reply))))
              (setq timingstuff
                      (replacenth timingstuff quaireno
                                  (list quaireno (quairetrig reply))))
              (putprop 'evalstuff timingstuff 'timingstuff)))
         (go loop)))


; showauxes displays the category number and header of an auxiliary question.
(defun showauxes (anaux)
```

```
  (msg (catnumber (car anaux)))
  (tab 10)
  (msg (catheader (car anaux)) N))


; showcats displays the number and header of a category.
(defun showcats (acategory)
  (msg (catnumber acategory))
  (tab 8)
  (msg (catheader acategory) N))


; showquaires displays the number, header, and audience (user) characteristics
; of a questionnaire.
(defun showquaires (aquaire)
  (msg (quairenumber aquaire))
  (tab 8)
  (msg (quaireheader aquaire))
  (tab 50)
  (msg (quaireaudexp aquaire))
  (tab 62)
  (msg (quaireaudtype aquaire) N))


; showquaires2 displays the number, header, and timing information
; of a questionnaire.
; returns: nil
(defun showquaires2 (aquaire)
  (prog (quairetiming)
        (msg (quairenumber aquaire))
        (tab 8)
        (msg (quaireheader aquaire))
        (tab 50)
        (cond ((null (setq quairetiming (getquairetiming (quairenumber aquaire))))
               (msg "not specified." N))
              (t (mapc 'whengiven (lastelement quairetiming))
                 (msg N)))))


; showquests displays the number, header etc of a question.
(defun showquests (aquestion)
  (msg (questnumber aquestion))
  (tab 8)
  (msg (questheader aquestion))
  (tab 50)
  (cond ((questaux aquestion) (msg "yes"))
        (t (msg "no")))
  (tab 63)
  (cond ((questcomment aquestion) (msg "yes"))
        (t (msg "no")))
  (tab 72)
  (cond ((questansvals aquestion) (msg "yes"))
```

```
        (t (msg "no")))
  (msg N))


; whengiven displays the natural language equivalent of a timing code.
; returns: t
(defun whengiven (timingcode)
  (cond ((equal timingcode 's) (msg "startup"))
        ((equal timingcode 'e) (msg "end of run"))
        ((numberp timingcode) (msg "designer triggers with " timingcode))
        ((null timingcode) (msg "time not specified"))
        (t (msg "illegal value")))
  (msg "." ") t)
```

1.4. Evuser

```
; ---------------------------------------------------------------
; This file contains the portion of the evaluation system which is used only
; in user/application mode, ie while the application program is running.
;
; All functions are listed alphabetically by name.
;
; ---------------------------------------------------------------


; checkcomphrs checks user's spontaneous comments for phrases of interest to
;  a designer.
; Note: It is presently left as a hook - unimplemented - since there is no
;       immediately obvious useful pattern of comment analysis.
(defun checkcomphrs (acomment)
  nil)

; checkuser returns the user identification information from the property
; 'knownusers' of the variable 'evalstuff'.
; If the user is not known to the system, checkuser returns nil.
(defun checkuser (userid)
  (assoc userid (get 'evalstuff 'knownusers)))

; commentblurb tells a user how to enter spontaneous comments.
(defun commentblurb ()
  (msg "From time to time as the program is running, you may want to make a" N
       "comment about it. You may enter a comment at any time that the" N
       "program is waiting for a reply from you. Any comments you make will" N
       "be passed on to the system designers. If you start any reply with a" N
       "'keyword' from the following list, your reply will be interpreted as" N
       "a comment:" N)
  (msg (get 'evalstuff 'usercomments) N)
  (msg "After you enter your comment(s), the program will still be waiting" N
       "for your 'real' reply to whatever question it asked you." N N))
```

```
; errtally tallies error messages from the application system to a user.
; Messages are tallied by type, if requested by a system designer.
; input: the errortype byte of the first word of a message to be printed.
; output: a list of 'errorstuff', updated if the present input is an error msg.
(defun errtally (errortype)
 (prog (result typeknown)
       (setq
        result
        (mapcar
         (function
          (lambda (error)
                  (cond ((get 'evalstuff 'tallyerrortype)
                         (cond ((equal (car error) errortype)
                                (setq typeknown t)
                                (list errortype (add1 (lastelement error))))
                               (t error)))
                        (t (cond ((null (car error))
                                  (setq typeknown t)
                                  (list nil (add1 (lastelement error))))))))))
         (get 'evalstuff 'errorstuff)))
       (cond (typeknown (return result)))
       (cond ((get 'evalstuff 'tallyerrortype)
              (return (append1 (get 'evalstuff 'errorstuff)
                               (list errortype 1))))
             (t (return (list nil 1))))))


; evcleanread reads from the keyboard. It checks the user input to see
;  whether it is a comment. Comments are either intercepted or passed on to the
; calling program, according to the designer's setting of 'commentfate'.
; The function uses 'cleanread' to free users from the need to use parentheses
;  and quote characters in their inputs.
; side-effect: inputs of 'y' are converted to 'yes' and inputs of 'n' are
;  converted to 'no'.
(defun evcleanread ()
 (prog (reply)
       loop
       (noteuserpause)
       (cond ((equal (setq reply (cleanread)) 'y) (return 'yes))
             ((equal reply 'n) (return 'no)))
       (cond ((null (listp reply)) (return reply)))
       (cond ((null (member (car reply) (get 'evalstuff 'usercomments)))
              (return reply)))
       (cond ((get 'evalstuff 'commentphrases) (checkcomphrs reply)))
       (cond ((get 'evalstuff 'commentfilename)
              (updatea (get 'evalstuff 'commentfilename) reply)))
       (cond ((equal (get 'evalstuff 'commentfate) 'intercept)
              (notesystempause)
              (msg "Comment noted. Please reply to the original question." N)
              (go loop))
             (t (return reply)))))
```

```
; evmsg is the evaluation system replacement for msg.
; It should be used instead of msg by an application program.
; It monitors data to be msg'd to see whether error messages are present.
; If so, the error messages are handled according to the designer's wishes.
(def evmsg
 (nlambda
  (anarg
  (prog (workcar)
        (notesystempause)
        (cond ((null (get 'evalstuff 'tallyerrors))
               (return (eval (cons 'msg anarg)))))
        (cond ((listp (setq workcar (car anarg)))
               (return (eval (cons 'msg anarg)))))
        (setq workcar (aexploden workcar))
        (cond ((equal (get 'evalstuff 'errorid) (car workcar))
               (putprop 'evalstuff (errtally (nthelement workcar 2))
                        'errorstuff)
               (return (eval (cons 'msg (cdr anarg))))))
        (return (eval (cons 'msg anarg)))))))


; evpatom is equivalent to evprinc, in the same way that patom is equivalent to
; princ.
(defun evpatom (achar)
 (evprinc achar))


; evprinc does a normal princ, unless passed a single character with special
; meaning to the evaluation system. Such a character may be a trigger for a
; mid-run or end-of-run questionnaire. The questionnaire(s) are given, and
; the character trigger is not princ'd.
(defun evprinc (achar)
 (prog (numchar)
       (notesystempause)
       (cond ((listp achar) (return (princ achar))))
       (cond ((greaterp (length (setq numchar (aexploden achar))) 1)
              (return (princ achar))))
       (setq numchar (car numchar))
       (cond ((or (and (equal numchar (get 'evalstuff 'endofrun))
                       (setq numchar 'e))
                  (member numchar (get 'evalstuff 'dispecs)))
              (cond (evnewuser (return (givequaires numchar 'first_time)))
                    (t (return (givequaires numchar 'repeat)))))
             (t (return (princ achar)))))))


; evprint prints an argument, after checking whether it is an error message
; which the designer wants tallied by the evaluation system. If so, the error
; is tallied and the errorid (and type, if present) is stripped before the
; argument is printed.
; Note: evprint does not support the optional port parameter of print.
(def evprint
 (nlambda
```

```
(anarg)
(notesystempause)
(prog (workcar)
        (cond ((null (get 'evalstuff 'tallyerrors))
                (return (eval (cons 'print anarg)))))
        (cond ((listp (setq workcar (car anarg)))
                (return (eval (cons 'print anarg)))))
        (setq workcar (aexploden workcar))
        (cond ((equal (get 'evalstuff 'errorid) (car workcar))
                (putprop 'evalstuff (errtally (nthelement workcar 2))
                        'errorstuff)
                (return (eval (cons 'print (cdr anarg))))))
        (return (eval (cons 'print anarg)))))


; evread does a normal lisp keyboard read. It checks the user input to see
; whether it is a comment. Comments are either intercepted or passed on to the
; calling program, according to the designer's setting of 'commentfate'.
; Note 1: 'evread' does not support end of file handling, nor an explicit file
;       port parameter.
; Note 2: Since 'read' requires valid s-expressions as inputs, its utility is
;       limited. Try 'cleanread' or 'stringread' instead.
(defun evread ()
 (prog (reply)
        loop
        (noteuserpause)
        (setq reply (read))
        (cond ((null (listp reply)) (return reply)))
        (cond ((null (member (car reply) (get 'evalstuff 'usercomments)))
                (return reply)))
        (cond ((get 'evalstuff 'commentphrases) (checkcomphrs reply)))
        (cond ((get 'evalstuff 'commentfilename)
                (updatea (get 'evalstuff 'commentfilename) reply)))
        (cond ((equal (get 'evalstuff 'commentfate) 'intercept)
                (notesystempause)
                (msg "Comment noted. Please reply to the original question." N)
                (go loop))
              (t (return reply))))


; getuserrelationships returns a list containing the relationship(s) of
; this user to the application system.
; The function gets its information from the property 'knownusers' of
; the variable 'evalstuff'.
(defun getuserrelationships ()
 (lastelement (assoc evuserid (get 'evalstuff 'knownusers))))


; giveacategory gives a user a category of questions and returns the answers.
; input a category of questions from a questionnaire.
(defun giveacategory (acat)
 (list (catnumber acat) (list (exploden (catheader acat))
                              (mapcar 'giveaquest (catquests acat)))))
```

```
; giveaquest gives a question to a user and returns the answer.
; input: a question to be given.
(defun giveaquest (aquestion)
 (prog (answer)
        (msg "Question " (questnumber aquestion) "." N)
        (return (list (questnumber aquestion)
                      (givequestinfo aquestion)
                      (giveauxes (questaux aquestion) answer)))))


; giveauxes gives auxiliary questions to a user (if they exist and if they are
;  warranted by the user's answer to the parent question).
; returns: the corresponding auxiliary portion of questionnaire_answers.
(defun giveauxes (auxes answer)
 (remove nil
         (mapcar
          (function
           (lambda
            (anaux)
            (cond ((matchanswer answer (lastelement anaux))
                   (list (giveacategory (car anaux))
                         nil))
                  (t nil))))
          auxes)))


; givequaire gives a questionnaire to a user and appends the answers to an
;  answer file.
(defun givequaire (aquaire)
 (msg "Questionnaire " (quairenumber aquaire) "." N)
 (updatea (get 'evalstuff 'ansfilename)
          (list (exploden evuserid)
                (list (quairenumber aquaire)
                      (list
                      (exploden (quaireaudexp aquaire))
                      (exploden (quaireaudtype aquaire)))
                      (list (exploden (quaireheader aquaire))
                            (mapcar 'giveacategory (quairecats aquaire))))))
 (msg "End of Questionnaire " (quairenumber aquaire) "." N))


; givequaires gives appropriate questionnaires to users at the appropriate time.
; inputs: a timing_code indicating whether it is now start or end of run, or
;         time for designer-activated questionnaires.
;         The experience of the user with respect to this application.
; returns: t.
; side-effect: After giving end-of-run questionnaire(s), it will save timing
;              information if any has been requested by the designer.
(defun givequaires (timing_code experience)
 (prog (quairetiming)
       (mapc
        (function
         (lambda (aquaire)
```

```
                (cond ((and (member experience (quaireaudexp aquaire))
                            (setq quairetiming (getquairetiming
                                                (quairenumber aquaire)))
                            (member timing_code (lastelement quairetiming))
                            (or (intersect (getuserrelationships)
                                           (quaireaudtype aquaire))
                                (member 'all (quaireaudtype aquaire))))
                       (givequaire aquaire))))) questionnaires)
        (cond ((and (equal timing_code 'e) (timesomething))
               (saveruntiming)))
        t))


; givequestinfo gives the question_info portion of a question to a user and
;  returns the corresponding answer_info portion of questionnaire_answers.
; side-effect: sets the value of the variable 'answer'. This function is
;              intended to be called by 'giveaquest'.
(defun givequestinfo (aquestion)
 (prog (comment)
       (msg (questheader aquestion) N)
       (mapc (function (lambda (detline)
                               (msg detline N))) (questdetail aquestion))
       (setq answer (validanswer aquestion))
       (cond ((questcomment aquestion)
              (msg "Please expand your answer - just a line to clarify it." N)
              (setq comment (evcleanread))))
       (return (list
                (explodeh (questheader aquestion))
                nil
                (exploden comment)
                (exploden answer)))))


; matchanswer checks whether an answer is found within a set of answer ranges.
; returns t if the answer is found, nil otherwise.
(defun matchanswer (answer ansranges)
 (prog (test1 test2)
       (mapc
        (function
         (lambda
          (ansrange)
          (cond ((and (equal (type answer) (type (car ansrange)))
                      (equal (type answer) (type (lastelement ansrange))))
                 (cond ((numberp answer)
                        (setq test1 (list 'lessp
                                          'answer
                                          (car ansrange)))
                        (setq test2 (list 'lessp
                                          (lastelement ansrange)
                                          'answer)))
                       (t (setq test1 (list 'alphalessp
                                            'answer
                                            (car ansrange)))
```

```
                    (setq test2 (list 'alphalessp
                                      '(lastelement ansrange)
                                      'answer))))
              (cond ((and
                       (null (eval test1))
                       (null (eval test2)))
                     (return t))))))
        ansranges)))


; newuserblurb tells a first time user about the evaluation system.
; side-effect: sets the property 'knownusers' on the variable 'evalstuff'.
(defun newuserblurb ()
 (prog (count relationships reply)
       (msg "According to the system files, you have not used "
          (get 'evalstuff 'application) N
          "with the evaluation feature before." N)
       (msg "The evaluation feature is a program which is attached to "
          (get 'evalstuff 'application) "." N
          "It gathers information from users like yourself on a variety of topics." N
          "This information is passed on to the system designers so that" N
          "they can further improve the system." N N)
       (msg "Please help improve " (get 'evalstuff 'application)
         " by accurately answering the questionnaires" N
          "which will be presented to you. Thank you for your help." N)
       (msg "Ready to proceed? ")
       (waitforyes)
       (commentblurb)
       (msg N "Thank you again for your cooperation." N)
       (msg "Ready to proceed? ")
       (waitforyes)
       (msg "Which of these terms describe(s) your relationship to "
          (get 'evalstuff 'application) "?_" N)
       (msg
        (setdifference
         (union
          '(designer knowledgesource buyer programmer enduser)
          (get 'evalstuff 'audiences))
         '(all))
        N)
       (msg "If you are not sure which one applies to you, you are probably" N
          "an enduser." N)
       loop
       (msg "Enter all the names that apply, separated by spaces." N)
       (setq relationships (evcleanread))
       (msg "Are you satisfied that your entry is complete and correct? ")
       (cond ((null (yesorno (stringread))) (go loop)))
       (cond ((null (listp relationships)) (setq relationships
                                                 (list relationships))))
       (putprop 'evalstuff (append1 (get 'evalstuff 'knownusers)
                                    (list evuserid relationships))
                'knownusers)))
```

```
; notesystempause updates the 'systempauses' property of 'evalstuff'.
; ie. it records that a system pause took place, and its duration.
; A system pause is the interval from an i/o operation to a write.
; side-effect: updates the 'lastwrite' property of 'evalstuff' to note when
;              the last write took place.
(defun notesystempause ()
 (prog
  (number min max total pauselength)
         (putprop 'evalstuff (systime) 'lastwrite)
         (cond
          ((get 'evalstuff 'timesystempauses)
              (cond ((null (get 'evalstuff 'lastread))
                     (putprop 'evalstuff (get 'evalstuff 'lastwrite)
                              'lastactivity)
                     (return)))
              (setq number (add1 (car (get 'evalstuff 'systempauses))))
              (setq pauselength (abs (diff (get 'evalstuff 'lastwrite)
                                           (get 'evalstuff 'lastactivity))))
              (setq min (cadr (get 'evalstuff 'systempauses)))
              (cond ((lessp pauselength min) (setq min pauselength)))
              (setq max (caddr (get 'evalstuff 'systempauses)))
              (cond ((greaterp pauselength max) (setq max pauselength)))
              (setq total (plus (lastelement (get 'evalstuff 'systempauses))
                                pauselength))
              (putprop 'evalstuff (list number min max total)
                       'systempauses)
              (putprop 'evalstuff (get 'evalstuff 'lastwrite)
                       'lastactivity)))))

; noteuserpause updates the 'userpauses' property of 'evalstuff'.
; ie. it records that a user pause took place, and its duration.
; A user pause is the interval from an i/o operation to a read.
; side-effect: updates the 'lastread' property of 'evalstuff' to note when
;              the last read took place.
(defun noteuserpause ()
 (prog (number min max total pauselength)
         (putprop 'evalstuff (systime) 'lastread)
         (cond ((get 'evalstuff 'timeuserpauses)
              (cond ((null (get 'evalstuff 'lastwrite))
                     (putprop 'evalstuff (get 'evalstuff 'lastread)
                              'lastactivity)
                     (return)))
              (setq number (add1 (car (get 'evalstuff 'userpauses))))
              (setq pauselength (abs (diff (get 'evalstuff 'lastread)
                                           (get 'evalstuff 'lastactivity))))
              (setq min (cadr (get 'evalstuff 'userpauses)))
              (cond ((lessp pauselength min) (setq min pauselength)))
              (setq max (caddr (get 'evalstuff 'userpauses)))
              (cond ((greaterp pauselength max) (setq max pauselength)))
              (setq total (plus (lastelement (get 'evalstuff 'userpauses))
                                pauselength))
              (putprop 'evalstuff (list number min max total)
```

```
                    'userpauses)
                (putprop 'evalstuff (get 'evalstuff 'lastread)
                    'lastactivity)))))

; olduserblurb welcomes back a repeat-user of the system.
(defun olduserblurb ()
  (msg "Welcome back to " (get 'evalstuff 'application) "." N
       "You may be asked to reply to questionnaires from time to time." N
       "Thank you for your cooperation." N
       "Do you remember how to enter comments? ")
  (cond ((null (yesorno (stringread))) (commentblurb))))

; saveruntiming updates the run timing file with timing information for a
; particular run.
(defun saveruntiming ()
  (updatea (get 'evalstuff 'timingfilename)
           (list evuserid (get 'evalstuff 'systempauses)
                 (get 'evalstuff 'userpauses)
                 (abs (diff (get 'evalstuff 'sessionstart)
                            (systime))))))

; timesomething checks which aspects of a run are to be timed, and clears the
; values of items which are not to be timed.
; returns: t if at least one item is to be timed, nil otherwise.
(defun timesomething ()
  (prog (result)
        (cond ((get 'evalstuff 'timesystempauses) (setq result t))
              (t (putprop 'evalstuff nil 'systempauses)))
        (cond ((get 'evalstuff 'timeuserpauses) (setq result t))
              (t (putprop 'evalstuff nil 'userpauses)))
        (cond ((get 'evalstuff 'timesessionlength) (setq result t))
              (t (putprop 'evalstuff nil 'sessionlength)))
        (return result)))

; validanswer gets a user's answer to a question and ensures that it meets any
; restrictions specified in the answer_value portion of a questionnaire.
(defun validanswer (aquestion)
  (prog (answer ansranges)
        loop
        (setq answer (evcleanread))
        (cond ((null (setq ansranges (questansvals aquestion)))
               (return answer)))
        (cond ((matchanswer answer ansranges) (return answer)))
        (msg "Inappropriate reply. Please enter new answer." N)
        (go loop)))
```

1.5. Printansfile

```
;-----------------------------------------------------------------
; This file contains a stand-alone utility program which prints the contents
; of a questionnaire answer file for the evaluation system.
; It uses many of the general-purpose routines in the evaluation logic.
; To follow the program logic, begin reading at 'printansfile'.
;
; All functions are listed alphabetically by name.
;
;-----------------------------------------------------------------


; printansfile prints (on the terminal) the contents of the questionnaire
;  answer file for an application.
; input: the name of the file where run-independent evaluation data was saved.
(defun printansfile (filename)
  (load 'evmain)
  (load 'evdesigner)
  (setq evalstuff nil)
  (load filename)
  (setq tempevalstuff nil)
  (initles)
  (setplist 'evalstuff (plist 'tempevalstuff))
  (printanswers (get 'evalstuff 'ansfilename)))


; printanswers lists the contents of a questionnaire answer file.
; input: the name of the file containing the answers.
(defun printanswers (filename)
  (prog (quaireans inport)
        (cond ((null (probef filename)) (msg "No answer file." N)
               (return)))
        (setq inport (infile filename))
        loop
        (cond ((null (setq quaireans (read inport)))
               (msg "End of answer file." N)
               (close inport)
               (return)))
        (printquaireans quaireans)
        (go loop)))

; printquaireans lists the contents of a questionnaire answer record.
; input: an answer record.
(defun printquaireans (quaireans)
  (msg "User " (implode (car quaireans)) ", Questionnaire:" N)
  (msg "Number")
  (tab 8)
  (msg "Header")
  (tab 50)
  (msg "Experience")
  (tab 62)
  (msg "Type" N)
```

```
  (showquairesi (lastelement quaireans))
  (mapc 'showcatsi (quairecats (lastelement quaireans))))


; showcatsi displays the contents of a category on the screen. It assumes that
;  most items are in the form of lists of small fixnums, which it implodes.
(defun showcatsi (acat)
  (msg "Category:" N)
  (msg "Number Header" N)
  (msg (catnumber acat))
  (tab 8)
  (msg (implode (catheader acat)) N)
  (mapc 'showquestsi (catquests acat)))

; showquairesi displays the number, header, and audience (user) characteristics
; of a questionnaire.
; It assumes that all but the number are in the form of lists of small fixnums.
;  which it implodes.
(defun showquairesi (aquaire)
  (msg (quairenumber aquaire))
  (tab 8)
  (msg (implode (quaireheader aquaire)))
  (tab 50)
  (msg (implode (quaireaudexp aquaire)))
  (tab 62)
  (msg (implode (quaireaudtype aquaire)) N))

; showquestsi displays answer information for a question on the screen. It
;  assumes that most items are in the form of lists of small fixnums, which
;  it implodes.
(defun showquestsi (aquest)
  (prog (auxcats)
        (msg "Question:" N)
        (msg "Number Header")
        (tab 60)
        (msg "Answer" N)
        (msg (questnumber aquest))
        (tab 8)
        (msg (implode (questheader aquest)))
        (tab 60)
        (msg (implode (questansvals aquest)) N)
        (cond ((and
                 (questcomment aquest)
                 (implode (questcomment aquest)))
               (msg "Solicited comment: " (implode (questcomment aquest)) N)))
        (cond ((questaux aquest) (setq auxcats (mapcar 'car (questaux aquest)))
               (msg "Auxiliaries:" N)
               (mapc 'showcatsi auxcats)))
        (msg N)))
```

APPENDIX B

SELECTMICRO DESCRIPTION

Selectmicro is a small-scale expert system which allows users to choose which microcomputer(s) might fit their needs. Its knowledge-base is contributed by users themseleves, rather than being 'hard-coded' by a programmer. Knowledge is stored in the form of 'candidates', which are names of microcomputer models, and 'rules', which are criteria for selecting or rejecting particular machines. At the time of the tests, the program contained representations of 101 candidates and thirty-four rules.

In a typical run, a user is asked what model(s) of machine he had been considering. If he responds with a model name unfamiliar to the program, he would be asked to provide as much information as he could about the particular model. This information would then be made available to future users, if needed. The program asks questions of the users in such a way that the user's time commitment is minimized. In other words, the program 'reflects' on what question (or rule) to ask in a particular user's unique circumstances in order to arrive at a 'short list' of candidates as quickly as possible. The user can ask the program to explain unfamiliar terms at will. As with rules and candidates, the explanations have been provided by other users over time, rather than having been hard-coded into the program. Before leaving the program a user is invited to contribute new knowledge, in the form of new rules, candidates, or explanations.

APPENDIX C

DETAILED CORRELATION RESULTS

The table below shows complete correlation results for probabilities no greater than p=0.05 using Pearson's r test. A perfect positive correlation value would be 1.000. A perfect negative value would be -1.000. A zero value indicates no correlation between the variables. A probability figure p=0.05 means there is a five per cent probability that the observed correlations were due to chance. As outlined in the text, these results are intended only as an illustration of the kind of use to which the evaluation system could be put. The small sample sizes and large numbers of variables arbitrarily examined below suggest that many of the apparently significant correlations shown are entirely due to chance.

Item	Correlated With (n=12)			
Perceived Accurate	ease of use 0.5222 p=0.041	speed adequate 0.7746 p=0.002	would use often 0.5556 p=0.030	
Age	education 0.5476 p=0.033	excessive jargon -0.5926 p=0.021	should ask 0.5370 p=0.036	
Education	age 0.5476 p=0.033	lighting ok -0.7023 p=0.005		
Ease of Use	accuracy 0.5222 p=0.041	interfered -0.6742 p=0.008	speed ok 0.6742 p=0.008	would use often 0.5222 p=0.041
Goal Met	should ask -0.5976 p=0.020	speed ok 0.6325 p=0.014		
Interference	lighting adequate -0.6742 p=0.008			
Excess Jargon	age -0.5926 p=0.021	should ask -0.5292 p=0.038		
Lighting Adequate	accuracy 0.5222 p=0.041	interference -0.6742 p=0.008		
Mood Changed	noise ok -1.000 p=0.0			
Sex	Used comment feature 0.6831 p=0.007	info complete -0.5222 p=0.041	physical change 0.5222 p=0.041	
Should Ask More Questions	age 0.5370 p=0.036	goal met -0.5976 p=0.020	jargon -0.5292 p=0.038	
Speed Adequate	accuracy 0.7746 p=0.002	easy 0.6742 p=0.008	goal met 0.6325 p=0.014	would use often 0.7746 p=0.002
Would Use Often	accuracy 0.5556 p=0.030	easy 0.5222 p=0.041	speed ok 0.7746 p=0.002	

APPENDIX D

DETAILS OF STANDARD QUESTIONNAIRES

```
; This is the standard questionnaire file, evstdquest.

(def initquestionnaires (lambda nil
 (setq questionnaires
  '((1 ((first_time) (all))
   ((Describe users)
    ((1
     (Get user profile)
     ((1 (Please enter your age at your nearest birthday.)
         nil nil ((3 123))) nil)
      (2 (What is your sex? Please reply m or f.)
         nil nil ((m m) (f f))) nil)
      (3 (Please indicate your highest completed level of formal education.)
        (Answer with the appropriate letter.)
          a. Grade school.)
          b. High school.)
          c. College or technical diploma.)
          d. University degree.)
          e. Graduate degree(s).))
        t ((a e))) nil)
      (4 (Please indicate the amount of computer experience you have had.)
         (a. None until using this program.)
          b. Very little.)
          c. Some, but not extensive.)
          d. A great deal.))
        t ((a d)))
        (((1
         (Nature of computing experience)
          ((1 (Has your experience been)
              (a. Occasional?) b. Regular?))
           nil ((a b))) nil)
           (2 (Has your computing experience been)
               (a. Recent?)
                b. Some time ago?)
                c. Both recent and some time ago?))
            nil ((a c))) nil)))) ((b d)))))
      (5 (Generally, how would you rate your level of computing skills?)
         (a. Beginner.)
          b. Intermediate.)
          c. Expert.))
        t ((a c))) nil)
      (6 (What is your overall goal in using this system?)
         (Just summarize your goal in a single line.))
        nil nil) nil)
      (7 (How thoroughly were you briefed before you used the system)
         (a. No briefing at all.)
          b. Casual, informal explanation.)
          c. Detailed explanation, possibly with thorough written
              documentation.)
          d. Formal course of instruction.)
          e. Other.))
        nil ((a e)))
        (((1
         (Explain 'other' briefing.)
```

```
        ((1 (|Please explain your answer - just a line to clarify it.|
             nil nil nil) nil)))) ((e e)))))
    (8 (|Generally, do you see computers as|
       (|a. A threat?|
        |b. A fact of modern life which you must become used to.|
        |c. A useful tool.|
        |d. A great boon to mankind.|)
       t ((a e))) nil)
    (9 (|Please enter your job title, or major field of study if you are
        a student.|
       nil nil nil) nil)
    (10 (|How long have you been in this role?|
        nil nil nil) nil)
    (11 (|How long have you been in the same general type of work
        or studies?|
nil nil nil) nil)))))))

  (2 ((first_time repeat) (all))
   (|User state on startup|
    ((1
     (|Find user state on startup|
       ((1 (|What is your present mood or frame of mind?|
           (|Choose the letter that comes closest to describing how you
             feel now.|
            |a. Relaxed, contented, or happy.|
            |b. nervous, apprehensive, or anxious.|
            |c. Angry, upset, or frustrated.|)
           nil ((a c)))
          (((1 (|Cause of bad mood|
           ((1 (|Please indicate the main cause of your present mood.|
               (|a. Personal problems (eg family, health, marital).|
                |b. Problems related to work or school (eg workload,
                    argument with boss).|
                |c. Problems related to your surroundings (eg weather,
                    noise).|
                |d. Problems related to this program or system.|)
               t ((a d))) nil))))) ((b c)))))
         (2 (|Please describe your physical state at the moment.|
             (|Just key a line indicating your 'degree of comfort'.
               Mention any headaches,|
              |eyestrain etc, if you don't mind.|)
             t nil) nil)))))))

  (3 ((first_time) (designer programmer))
   (|Describe the technical user.|
    ((1 (|Profile of technical user.|
        ((1 (|Please describe your overall level of computing experience.|
            (|Just summarize the main points in a single line.|)
            nil nil) nil)
         (2 (|Please describe your experience with this type of application.|
            (|Again, just summarize the main points in a single line.|)
           nil nil) nil)
         (3 (|What computer languages do you know? Just list the names,
             separated by blanks.|
           nil t nil) nil)
         (4 (|How would you rate your morale at the present moment?|
            (|a. Excellent.| |b. Good.| |c. Fair.| |d. Poor.|)
           t ((a d))) nil)))))))

  (4 ((first_time) (enduser buyer))
   (|Naive user expectations.|
    ((1 (|User expectations.|
        ((1 (|Where did you first hear of this system?|
         nil t nil) nil)
```

```
(2 (|What kinds of things have you heard about it?|
   (|Just summarize the main points in a single line.|)
   nil nil) nil)))))))

(5 ((repeat) (all))
 (|Reasons for system re-use.|
  ((1 (|Why use again?|
      ((1 (|Why are you using this system again?|
          (|a. Necessary in job.|
           |b. Mainly to correct error(s).|
           |c. Other reason(s).|)
          nil ((a c)))
         (((1 (|more use expected?|
          ((1 (|Do you expect to use the system yet again
                (after this run)?|
              nil t ((yes yes) (no no))) nil)))) ((b c)))))
       (2 (|Has your opinion of the system changed significantly since you
            last used it?|
          nil nil ((yes yes) (no no)))
           (((1 (|reason for changed opinion of system.|
            ((1 (|Please expand on your answer - just enter a line to
                 clarify it.|
                nil nil nil) nil))))) ((yes yes)))))))))))

(6 ((repeat) (enduser))
 (|Remarks re ease of learning.|
  ((1 (|Easy to learn?|
      ((1 (|How often would you say you have used this application system?|
          (|a. Two to five times.|
           |b. Six to twenty times.|
           |c. Over twenty times.|)
          nil ((a c)))
          (((1 (|Ease of learning.|
           ((1 (|How easy or difficult was this system to learn?|
               (|a. Extremely easy.|
                |b. Fairly easy.|
                |c. Fairly difficult.|
                |d. Extremely difficult.|)
               t ((a d))) nil)))) ((b b)))))))))))

(7 ((first_time repeat) (enduser))
 (|User remarks re errors.|
  ((1 (|Error rate.|
      ((1 (|How many errors did you feel you made in this session?|
          (|a. None.| |b. One to five.| |c. Over five.|)
          nil ((a c)))
         (((1 (|How to improve error rate.|
          ((1 (|What might help reduce the number of errors?|
              nil nil nil) nil)))) ((b c)))))))))

(8 ((repeat) (designer programmer))
 (|Technical user status report.|
  ((1 (|Time and costs.|
      ((1 (|How many hours have you worked on the system since you last
            signed on to it?|
          nil nil ((0 1000))) nil)
       (2 (|How much money have you spent on this system since you last
            signed on to it?|
          nil nil ((0 50000))) nil)
       (3 (|Since you last used the system, has most of your system time
            been spent on:|
          (|a. Design.|
           |b. Consultation.|
           |c. Programming.|
```

```
                |d. Hardware issues.|
                |e. User training.|
                |f. Documentation.|
                |g. Other.|)
               t ((a g))) nil))))
        (2 (|System Status.|
          ((1 (|Is the system presently:|
              (|a. Being developed?|
               |b. In production?|
               |c. Undergoing maintenance?|
               |d. Nearing the end of its life cycle?|)
              nil ((a d)))
              (((1 (|Development questions.|
               ((1 (|What is the state of the main problem definition for the
                     system?|
                   nil nil nil) nil)
               (2 (|What task are you working on now?|
                   nil nil nil) nil)
               (3 (|Have any new constraints been placed on the problem
                     solution?|
                   nil nil ((yes yes) (no no)))
                   (((1 (|Nature of constraints.|
                    ((1 (|What is the nature of these constraints?|
                        (|a. Budgetary.| |b. Deadline.| |c. Other.|)
                        t ((a c))) nil))))) ((yes yes)))))
               (4 (|Are you using any new computer languages that you have
                     not mentioned before?|
                   nil nil ((yes yes) (no no)))
                   (((1 (|New languages|
                    ((1 (|Please enter the language name(s).|
                        nil t nil) nil)))) ((yes yes)))))
               (5 (|Are you using any new development tools that you have
                     not mentioned before?|
                   nil nil ((yes yes) (no no)))
                   (((1 (|New development tools.|
                    ((1 (|Please enter the tool name(s).|
                        nil t nil) nil)))) ((yes yes)))))))) ((a a)))))
          (2 (|How well do you think this implementation meets the design
                goals?|
              nil nil nil) nil)
          (3 (|Please rate the system with respect to efficiency:|
             (|a. Excellent.| |b. Good.| |c. Fair.| |d. Poor.|)
             nil ((a d))) nil)
          (4 (|Please rate the system with respect to speed:|
             (|a. Excellent.| |b. Good.| |c. Fair.| |d. Poor.|)
             nil ((a d))) nil)
          (5 (|Please rate the system with respect to resource utilization:|
             (|a. Excellent.| |b. Good.| |c. Fair.| |d. Poor.|)
              nil ((a d))) nil)
          (6 (|Please rate the maintainability of the source code:|
             (|a. Excellent.| |b. Good.| |c. Fair.| |d. Poor.|)
              nil ((a d))) nil)
          (7 (|Please rate the portability of the source code:|
             (|a. Excellent.| |b. Good.| |c. Fair.| |d. Poor.|)
              nil ((a d))) nil)
          (8 (|Please rate the adequacy of the system's hardware:|
             (|a. Excellent.| |b. Good.| |c. Fair.| |d. Poor.|)
              nil ((a d))) nil)))))))

    (9 ((first_time repeat) (buyer))
     (|Buyers' opinions.|
      ((1 (|Buyer resistance.|
          ((1 (|In your view, what is the strongest competitor for this
                system?|
```

```
      nil nil nil) nil)
    (2 (|How does this system compare to the competition?|
      (|a. This system is better.|
       |b. This system and its competition are about equal.|
       |c. This system is worse.|)
      nil ((a c)))
      (((1 (|Differences from competition.|
       ((1 (|Please indicate the most important difference between
            the systems.|
          nil nil nil) nil)))) ((a a) (c c)))))
    (3 (|If the price were right, would you use this system?|
      nil nil ((yes yes) (no no)))
      (((1 (|Reasons for not buying.|
       ((1 (|Why wouldn't you use the system?|
          nil nil nil) nil)))) ((no no)))))))
  (2 (|Marketing tips.|
   ((1 (|In your opinion, where should the system be advertised?|
      (|a. Computer journals.|
       |b. General audience magazines.|
       |c. Newspapers.|
       |d. Radio.|
       |e. Television.|
       |f. Other.|)
      t ((a f))) nil)))))))

(10 ((first_time repeat) (knowledgesource))
 (|Opinions of knowledge sources.|
  ((1 (|Expert criticism.|
   ((1 (|Has the system captured your knowledge effectively?|
      nil nil ((yes yes) (no no)))
      (((1 (|Why expert knowledge is not captured effectively.|
       ((1 (|Please explain why you answered 'no'. Just summarize your
            reasons in a line.|
          nil nil nil) nil))))) ((no no)))))
    (2 (|Where could the system use improvement?|
      nil nil nil) nil)))))))

(11 ((first_time repeat) (all))
 (|Summary of user impressions|
  ((1 (|Opinions re implementation.|
   ((1 (|What should the system do that it does not presently do?|
      nil nil nil) nil)
    (2 (|What does the system presently do that it should not do?|
      nil nil nil) nil)
    (3 (|Would you use the system often if it were available?|
      nil nil ((yes yes) (no no)))
      (((1 (|Why not use?|
       ((1 (|Why not? Just key a line summarizing your reason(s).|
          nil nil nil) nil)))) ((no no)))))
    (4 (|Would you recommend the system to a friend?|
      nil nil ((yes yes) (no no))) nil)
    (5 (|Generally, were the system's response times fast enough for your
       needs?|
      nil nil ((yes yes) (no no))) nil)
    (6 (|Was the system easy to use?|
      nil nil ((yes yes) (no no)))
      (((1 (|Why not easy to use?|
       ((1 (|Why not? Just key a line to summarize the main problem(s).|
          nil nil nil) nil)))) ((no no)))))
    (7 (|Does the system use jargon or technical terms unnecessarily?|
      nil nil ((yes yes) (no no)))
      (((1 (|Example of jargon.|
       ((1 (|Please enter an example of such jargon - just key a few
            words.|
```

```
        nil nil nil) nil))))) ((yes yes)))))
    (8 (|Please rate the quality of documentation for the system:|
        (|a. Excellent.| |b. Good.| |c. Fair.| |d. Poor.| |e. Don't know.|)
        nil ((a e))) nil)
    (9 (|Please rate the quantity of documentation for the system:|
        (|a. Too much.| |b. About right.| |c. Too little.|)
        nil ((a c))) nil)
    (10 (|As far as you know, does the system provide accurate information?|
        nil nil ((yes yes) (no no))) nil)
    (11 (|As far as you know, does the system provide reasonably complete
          information?|
        nil nil ((yes yes) (no no))) nil)
    (12 (|Did you achieve your goal in this session?|
        nil nil ((yes yes) (no no)))
        (((1 (|Why goal not achieved.|
          ((1 (|Why not? Just summarize the main reason(s) in a line.|
            nil nil nil) nil)))) ((no no)))))
    (13 (|In your opinion, what is the system's best feature?|
        nil nil nil) nil)
    (14 (|In your opinion, what is the system's worst feature?|
        nil nil nil) nil))))
(2 (|Opinions re application.|
    ((1 (|Do you see a need for this kind of system?|
        nil nil ((yes yes) (no no))) nil)
     (2 (|In your view, what is the probable future life of this kind of
          system:|
        (|a. Under a year.|
         |b. One to three years.|
         |c. More than three years.|)
        nil ((a c))) nil)
     (3 (|Do you see the problem that this system addresses as being:|
        (|a. Complex.| |b. Fairly routine.| |c. Easy or trivial.|)
        nil ((a c))) nil)
     (4 (|In your view, would a regular user of this system be a person of:|
        (|a. High status.|
         |b. Neutral or unpredictable status.|
         |c. Low status.|)
        nil ((a c))) nil))))
(3 (|Physical environment.|
    ((1 (|What room and building are you in? Just describe your location
          in one line.|
        nil nil nil) nil)
     (2 (|What kind of terminal are you using?|
        nil nil nil) nil)
     (3 (|The noise level here is:|
        (|a. Much too quiet.|
         |b. Somewhat more quiet than I would like.|
         |c. Just about ideal for my purposes.|
         |d. Somewhat more noisy than I would like.|
         |e. Much too noisy.|)
        nil ((a e)))
        (((1 (|Noise source.|
          ((1 (|What is the main source of the noise:|
            (|a. Computer equipment.|
             |b. Other equipment.|
             |c. Voices.|
             |d. Traffic or other 'outside' noise.|
             |e. Other sources.|)
            nil ((a e)))
            (((1 (|Source of equipment noise.|
              ((1 (|Which piece of equipment is mainly responsible for
                    the noise?|
                (|a. Printer(s).|
                 |b. Keyboard(s).|
```

```
        |c. Terminal hum or whine.|
        |d. Other sources.|)
       nil ((a d))) nil)))) ((a a)))))))) ((d e)))))
(4 (|The temperature here is:|
   (|a. Much too cold.|
    |b. Somewhat too cold for comfort.|
    |c. Just about ideal for this kind of work.|
    |d. Somewhat too warm for comfort.|
    |e. Much too warm.|)
   nil ((a e))) nil)
(5 (|Is the lighting adequate where you are now?|
   nil nil ((yes yes) (no no))
   (((1 (|Problems with lighting.|
    ((1 (|What is the main problem with the lighting?|
        nil nil nil) nil)
     (2 (|How long has this problem existed?|
        nil nil nil) nil)
     (3 (|Please suggest a way to improve the problem.|
        nil nil nil) nil)))) ((no no)))))
(6 (|Are the seating arrangements satisfactory?|
   nil nil ((yes yes) (no no)))
   (((1 (|Problems with seating.|
    ((1 (|What is the main problem? Just summarize it in a single
          line.|
        nil nil nil) nil)))) ((no no)))))
(7 (|Is there enough work space at your terminal?|
   nil nil ((yes yes) (no no))) nil)
(8 (|Please summarize your impressions of your present site - just a
     single line.|
   nil nil nil) nil))))
(4 (|Changes in mood or condition.|
((1 (|Has your mood changed significantly since you started this run?|
   nil nil ((yes yes) (no no)))
   (((1 (|Explain mood changes.|
    ((1 (|Please explain your changed mood.|
        nil nil nil) nil)))) ((yes yes)))))
     (2 (|Has your physical condition changed since you started this
          run?|
        nil nil ((yes yes) (no no)))
        (((1 (|Explain changed physical condition.|
         ((1 (|Please explain your changed physical condition.|
            nil nil nil) nil)))) ((yes yes))))))))
(5 (|Effect of questionnaires on user.|
((1 (|Did answering the questionnaires interfere with your use of the
      system?|
   nil nil ((yes yes) (no no))) nil)
 (2 (|Did you find any of the questions offensive or annoying?|
   nil nil ((yes yes) (no no)))
   (((1 (|Annoying questions.|
    ((1 (|Please give an example of a question that bothered you.|
       nil nil nil) nil)))) ((yes yes)))))
 (3 (|Are there questions you feel should have been asked which were
      not asked?|
   nil t ((yes yes) (no no))) nil)
 (4 (|Did you use the 'comment' feature?|
   nil nil ((yes yes) (no no))) nil)
 (5 (|What do you think of the comment feature?|
   nil nil nil) nil))))))))))))
```

BIBLIOGRAPHY

Aiken, Lewis. *Psychological Testing and Assessment*, Allyn and Bacon, Boston, 1982.

Alty, J. L. and M. J. Coombs. "Communicating with University Computer Users: A Case Study", in Coombs, M. J. and J. L. Alty. *Computing Skills and the User Interface*, Academic Press, London, 1981.

Bair, James. "Avoiding Working Non-Solutions To Office Systems Design", in Landau, Bair and Siegman, eds, *Emerging Office Systems*, Ablex, Norwood, New Jersey, 1982, pp. 125-135.

Basili, Victor. "Data Collection, Validation And Analysis", in Perlis, Sayward and Shaw, eds. *Software Metrics: An Analysis and Evaluation*, MIT Press, Cambridge, Mass., 1981, pp. 143-159.

Borovits, Israel and Seev Neumann. *Computer Systems Performance Evaluation*, Lexington Books, Lexington, Mass., 1979.

Browne, J.C. and Mary Shaw. "Toward a Scientific Basis for Software Evaluation", in Perlis, Sayward and Shaw, eds. *Software Metrics: An Analysis and Evaluation*, MIT Press, Cambridge, Mass., 1981, pp. 19-40.

Carroll, John and Mary Beth Rosson. "Beyond MIPS: Performance is not Quality", *Byte*, February, 1984, pp 168-172.

Cercone, Nick, Robert Hadley, and Tomek Strzalkowski. "The Automated Academic advisor: an Introduction and Initial Experiences", *TR83-11*, Laboratory for Computer and Communication Research, Department of Computing Science, Simon Fraser University, Burnaby, B.C., Canada, 1983.

Chrysler, Earl. "Some Basic Determinants of Computer Programming Productivity", *Comm. of the ACM* 21 (6), June, 1978, pp. 472-483.

Coombs, M. J. and J. L. Alty. *Computing Skills and the User Interface*, Academic Press, London, 1981.

Damodaran, Leela, Alison Simpson, and Paul Wilson. *Designing Systems for People*, NCC

Publications, Manchester, 1980.

De Millo, Richard and Richard Lipton. "Software Project Forecasting", in Perlis, Sayward and Shaw, eds. *Software Metrics: An Analysis and Evaluation*, MIT Press, Cambridge, Mass., 1981, pp. 77-89.

Denicoff, Marvin and Robert Grafton. "Software Metrics: a Research Initiative", in Perlis, Sayward and Shaw, eds. *Software Metrics: An Analysis and Evaluation*, MIT Press, Cambridge, Mass., 1981, pp. 13-18.

Deutsch, Michael. *Software Verification and Validation*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

Ferrari, Domenico. *Computer Systems Performance Evaluation*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

Ferrari, Domenico and Massimo Spadoni, eds. *Experimental Computer Performance Evaluation*, North-Holland, Amsterdam, 1981.

Gilb, Tom. *Software Metrics*, Studentlitteratur, Sweden, 1977.

Gorsuch, R. L. *Factor Analysis*, Saunders, Philadelphia, 1974.

Gough, H. G. "The Adjective Checklist as a Personality Assessment Research Technique," *Psychological Reports*, 6, 1960, pp 107-122.

Hadley, Robert. "A Natural Language Query System for a PROLOG Database", MSc Thesis, Simon Fraser University, Burnaby, B.C., Canada, 1983.

Haynes, Stephen, and Chrisman Wilson. *Behavioral Assessment: Recent Advances in Methods, Concepts, and Applications*, Jossey-Bass, San Francisco, 1979.

Kaplan, Robert, and Dennis Saccuzzo. *Psychological Testing*, Wadsworth, Belmont, California, 1982.

King, Judy. *Evaluating Data Base Management Systems*, Van Nostrand Reinhold, New York, 1981.

Landau, Robert, James Bair, Jean Siegman, eds. *Emerging Office Systems*, Ablex, Norwood, New Jersey, 1982.

Ledgard, Henry, Andrew Singer, John Whiteside. *Directions in Human Factors for Interactive Systems*, Springer-Verlag, Berlin, 1981.

Likert, R. A Technique for the Measurement of Attitudes, *Archives of Psychology*, 40, 1932.

Martin, Fred J. "SELECTMICRO: an Expert Systems Approach to Microcomputer Selection", unpublished typescript (presently under commercial development), North Vancouver, B.C., Canada, 1983.

Mehlmann, Marilyn. *When People Use Computers*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

Michaelsen, Robert and Donald Michie. "Expert Systems in Business", *Datamation*, November, 1983, pp240-246.

Milgram, Stanley. "Behavioral Study of Obedience", *Journal of Abnormal and Social Psychology*, 67, 1963, pp-371-378.

Miller, G. A. *The Psychology of Communication*, Pelican Books, London, 1970.

Moos, R. H. "Conceptualizations of Human Environment," *American Psychologist*, 28, 1973, pp 652-665.

Morris, Michael, and Paul Roth. *Computer Performance Evaluation*, Van Nostrand Reinhold, New York, 1982.

Parducci, A. "The Relativism of Absolute Judgements," *Scientific American*, 219 (6), 1968, pp 84-90.

Perlis, Alan, Frederick Sayward and Mary Shaw, eds. *Software Metrics: An Analysis and Evaluation*, MIT Press, Cambridge, Mass., 1981.

Richards, William, and Emily Larimer. "Experimental and Non-Experimental Methods for Evaluating Office Automation Systems", Paper given at the International Communications Association, San Francisco, May, 1984.

Roethlisberger, F. J. and W. J. Dickson. *Management and the Worker*, Harvard University Press, Cambridge, Mass., 1939.

Rogers, Everett. "Once a System is Invented, How Do We Get It Into The Users' Hands?", in Landau, Bair and Siegman, eds, *Emerging Office Systems*, Ablex, Norwood, New Jersey, 1982, pp. 279-285.

Schneiderman, B., R. Mayer, D. McKay, and P. Heller. "Experimental Investigation of the Utility of Detailed Flowcharts in Programming", *Comm. of the ACM* 20 (6), June, 1977, pp. 373-384.

Sundberg, Norman. *Assessment of Persons*, Prentice-Hall, Englewood Cliffs, New Jersey, 1977.

Strzalkowski, Tomek. "ENGRA, Yet Another Parser for English", *TR83-10*, Laboratory for Computer and Communication Research, Department of Computing Science, Simon Fraser University, Burnaby, B.C., Canada, 1983.

Weinberg, Gerald. *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York, 1971.

Weizenbaum, Joseph. *Computer Power and Human Reason*, Freeman, San Francisco, 1976.