Division of Research
Graduate School of Business Administration

September 1986

TECHNIQUES FOR INCORPORATING HUMAN FACTORS IN THE SOFTWARE LIFECYCLE*

Working Paper #475

Marilyn Mantei
University of Michigan

*Proceedings Structured Techniques Association Third Annual Conference, June 24-26, 1986, Chicago, IL, pp 177-203.

Abstract

Although much literature is devoted to the development of principles on how to design "user friendly" systems, very little exists on how to incorporate these principles in the process of software development and maintenance, commonly known as the software lifecycle. The author of this paper presents a "generic" software lifecycle plan and discusses where and how the application of human factors principles and the evaluation of the user interface take place in the software development process. The costs of this approach and the cost/benefit tradeoffs that can be expected using this style of user interface development are examined and used to recommend when the overall system usage and projected software life justify applying the human factors techniques being described.

I. Introduction

Traditional software development breaks the software project into individual components each of which performs a specific function in the process of putting together a software product (Aron, 1983; Jackson, 1975; Metzger, 1981; Mills, 1970; Rubin, 1970; Vick & Ramamoorthy, 1984). With the advent of new program development tools and the growing importance of human factors issues in the design and building of software, this accepted way of managing software development needs to give way to new ways of thinking about the process. In particular, because human factors issues arise at several of the stages and because prototypes of the proposed system can be tested long before the final system is in place, many levels of design iteration are now introduced into the development process.

Because these added methods enhance the software product and help ensure its acceptance in addition to reducing the product's maintenance (Card, Moran & Newell, 1983; Norman and Draper, 1986; Rubinstein & Hersh, 1984), their inclusion is a justified and natural evolution of the software development process. This paper does not address the issue of whether these ideas should be included in the software lifecycle. This is assumed. Instead, it addresses the issue of how to include these techniques by presenting an overview of recommended places within the traditional software development plan where human factors issues need to be treated. At each stage it then discusses what type of human factors work applies.

The next section of the paper presents the traditional software lifecycle. It then updates the lifecycle to represent current practice in the use of prototyping and automated design techniques. This modified prototyping lifecycle is expanded to illustrate how human factors methods can be added to the design.

A third section of the paper discusses the costs of adding the human factors studies to a project and relates these costs to the overall benefit they provide for the software design. It also presents a list of difficulties that can be encountered in testing the software such as the time investment in developing appropriate user tests and the limitations that prototypes and mockups have in assessing user difficulties. A fourth section gives a set of projections on when such an approach will result in real cost savings. It also raises the control issue flag, pointing out how the recommended changes to the lifecycle can bring about a loss of project control unless specific control procedures are implemented at the human factors stages as well. The final section of the paper summarizes and presents the author's conclusions.

The reader should note that many important items related to software development are not discussed in the paper. For example, the paper does not deal with the problems of staffing the proposed lifecycle or of getting the different types of professions working together as a team. Nor does it handle the real issues of product deadlines and organizational structures. It presents only a set of changes in structure and process for software development with the hope that their adoption will debug and modify them into useful working practices.

II. Recommended Lifecycle Changes

In order to establish a frame of reference for the changes to the software lifecycle which are being recommended, the paper briefly describes the traditional software lifecycle concepts that had their beginnings in the years 1970-75. It then illustrates the changes to the traditional lifecycle that have occurred with the advent of prototyping. This modified lifecycle represents current thoughts in software development and will be the format used to indicate the modifications being recommended, modifications which incorporate human factors engineering into the final product.

A. The Traditional Software Lifecycle

The traditional software lifecycle includes four to six stages depending on the level of detail in which it is presented. The six stage lifecycle plan has been selected for our discussion. This is illustrated in Figure 1. As a brief overview for the reader, a description of each of the stages follows.

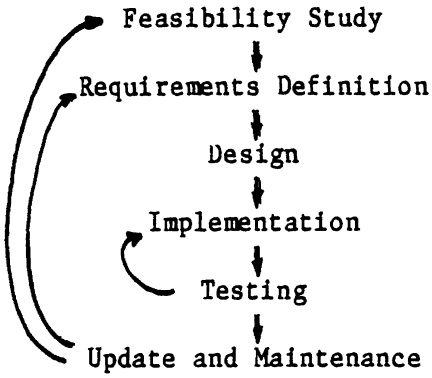

Figure 1. The Development Steps in a Traditional Software Lifecycle.

The first stage is the **Feasibility Study.** In this stage, the company planning a potential software project does a series of analyses to decide whether the project should or should not be carried out. The second stage is the **Requirements Definition.** If the software plan passes the first stage, then data is gathered to determine exactly what the software should accomplish. **Systems Design** follows the Requirements Definition. In this third stage, the design of the system is laid out, including inputs, outputs, program module specifications and procedures.

Following design, the fourth stage of **System Implementation** builds the system according to the design. A fifth stage, **Testing,** executes trial runs on the system with the goal of removing the errors in the system. An **Alpha Test** puts all the components of the system together and, after a debugging period, has them working as a system. A **Beta Test** involves a group of users who agree to work with the new system as the software developers continue to refine it and make it work as planned. Once the Beta Test is complete, the product is released.

The product then goes through its final phase, that of **Update and Maintenance.** The Update and Maintenance stage is usually the longest portion of the software lifecycle. As the software product is used, it evolves to meet the new needs of its users. It also breaks when new uses find problems that were not anticipated and tested for. Update and Maintenance often involves miniature steps of the entire lifecycle as changes are planned and incorporated into the system.

Breaking the product development into these components helps to plan the timing, cost and staffing of a software project and to maintain control over its progress. In particular, the linear sequence of steps makes it possible to achieve closure on a step in the development process, allowing the project managers to catch delays or problems early in the software lifecycle.

The introduction of prototyping techniques has changed the process, creating iterations between the Design and Requirements stages and, in some cases, including the Implementation stage as part of the Design stage. The use of prototyping significantly lowers the costs of user testing. Because of this, the prototyping lifecycle is used as the reference frame for building in the human factors aspects of software development.

B. The Prototyping Lifecycle

With the advent of prototyping, that is, the ability to quickly build early, albeit incomplete, versions of the proposed software system, the traditional software lifecycle has undergone major upheaval (Boar, 1984; Budde, Kuhlenkamp, Mathiassen & Zullighoven, 1984; Wasserman, 1982a; Wasserman, 1982b). In the traditional lifecycle future users of the software were not able to grasp the functions and support the software would provide until they viewed a finished or nearly finished product. Flowcharts, HIPO Techniques, Dataflow Diagrams, etc. (De Marco, 1978; Martin & McClure, 1985) do not adequately convey the actual workings of the system. Therefore, users cannot tell whether the designers have missed their mark and built a functionally useless system until late in the development process. This means expensive redesigns.

A prototype replaces the diagram designs of the design stage. With software prototypes, a user can get a "sense" of using the software tool before it has proceeded too far down the implementation trail. Thus, much more iteration and design changes take place early in the lifecycle. These changes to the traditional software lifecycle are shown in Figure 2. The modified lifecycle is briefly described in the paragraphs which follow.

The Feasibility Study and Requirements Definition stages remain the same as in the Traditional Software Lifecycle. The Systems Design stage is split into two stages, that of **Global Design** and **Prototype Construction.** The Global Design process lays out the system components in the same way as it is done in the Traditional Lifecycle, but the Prototype Construction replaces or precedes the Detailed Design. A prototype is built which represents the user interface to the system being constructed. This

prototype may contain all or a selected subset of the functions of the proposed system.

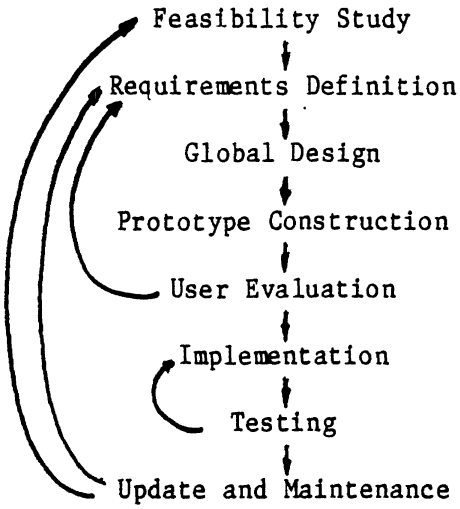

Figure 2. The Typical Development Steps in a Prototyping Lifecycle.

The prototype is then presented to future users of the system in the **User Evaluation** stage. These users examine the interface and try to use it. They then give feedback to the designers on whether the system meets their software support needs. Typically, the act of using the prototype brings about a new understanding of what the system will provide for the user. With this understanding comes a change in the requirements specifications of the system. The software development returns to the Requirements Definition stage and a new prototype is designed and tested. This takes place a number of times until the prototype reaches an acceptable stage of usefulness. The prototype then serves as the detailed design to be used for the Implementation stage. In some system designs, the prototype is incomplete and additional detailed specification is necessary. In other systems, the actual prototype is compiled into the final product and the Implementation stage is skipped or shortened.

The Testing stage is the same as the Testing stage in the Traditional Lifecycle, and the Update and Maintenance stage is similar with the exception that prototypes might be built or modified to reflect large system modifications. Although the Prototyping Lifecycle looks very much like the Traditional Lifecycle, time and energy is added to the development work at an early stage in order to obtain feedback on the acceptability of the final design. Adding human factors efforts to the lifecycle has the same impact of lengthening the software development time, and of increasing the acceptability of the final released code. These processes, in turn, reduce costs in the Update and Maintenance stage.

C. The Human Factors Software Lifecycle

The changes proposed for the Human Factors Software Lifecycle are illustrated in Figure 3. First, a stage preceding the Feasibility Study called **Market Analysis** is added. This stage determines what product needs to be developed for the market, usually by running focus groups on the intended users. A Market Analysis assesses the market's potential need for a proposed product. This is different from the traditional Feasibility Study in that it examines people's perceptions and feelings about the information processing tasks that they currently perform. The goal of this approach is to find out if a software system which aided them in accomplishing these tasks would be adopted.

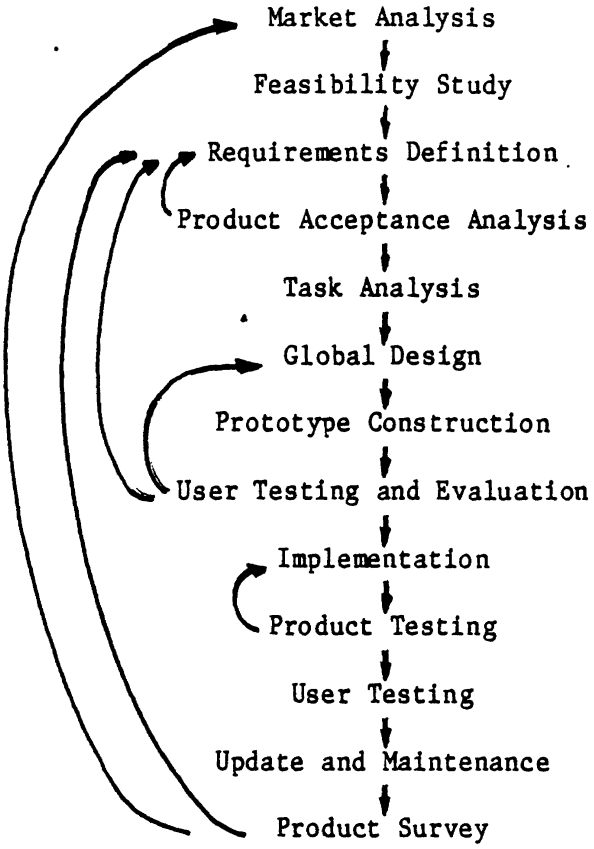

Figure 3. The Development Steps in a Human Factors Software Lifecycle.

Following the Requirements Analysis, a second human factor's stage is added, that of **Product Acceptance Analysis.** Now that the particular software product has been proposed, a mockup of what the product does can be presented to a sampling of its intended market. The future user's reactions to the mockup will generate information on whether the software system, as currently envisioned, will be acceptable. This information is usually acquired via focus group studies and user surveys. Projected users of the system can also recommend changes at this point. This phase is

particularly useful for testing whether the appropriate requirements have been specified.

Following the Product Acceptance Analysis, a **Task Analysis** is performed by the human factors specialist on the project. This task analysis looks at how the user "thinks" about the task and the "mental" data manipulation procedures he or she currently uses. The results of this analysis are applied in the Systems Design stage where the interface is tuned to match the thinking processes of the user. The Systems Design stage changes dramatically because many of the interface design decisions in the human factors structure are now negotiated with the psychology trained personnel on the project.

Once the design of the system is finished, a prototype is built in the **Prototype Construction** stage. Part of the design stage may incorporate prototype construction, especially if the prototyping software in use separates the interface design from the rest of the software system. The finished prototype is used in the **User Testing and Evaluation** stage. In the testing portion of this stage projected users perform a cross section of tasks for which the system is intended. They are studied while using the prototype. Problems they have with learning the system or with extended use of the system are incorporated into design change recommendations. These changes are built into new prototypes, and testing continues until the learning and extended use patterns of the system are at acceptable levels of effort and ease-of-use, respectively.

The Evaluation portion follows the Testing portion. After users have gained some experience with the system, they can then assess whether the system performs the desired functions and also whether the effort needed to learn the system is in concordance with the features the system provides. If the system does not do what the user requires of the system, the Requirements Definition stage is reopened and new requirements are drawn up to meet the user's needs.

Following the User Testing and Evaluation stage, the actual system is implemented. Once the system passes the Unit and System texts in the Product Testing stage, it is put through a second round of **User Testing.** These tests are performed on a working system. The results may differ from the prototype tests because of the differences in response time and complexity in the final system. Results from these tests can also serve as benchmarks for expected human performance levels and learning rates. These values can be used to schedule training, estimate work output and as a selling factor for the software system.

When the product is released, user feedback on the product is obtained during the **Product Survey** stage. This is intertwined with Update and Maintenance. The data from the Product Survey stage is used to drive updates. These updates are put into the system beginning with the Requirements Definition or the Market Analysis. The stage entered depends on the amount of change introduced by the proposed update. If the update dramatically changes the purpose and interface of the software, the entire Software Lifecycle is followed in its re-implementation. If it is a

smaller change, the change can be incorporated beginning with a new requirements definition.

The reader should note that the above changes can be described within the Traditional Lifecycle, i.e., a Feasibility Analysis consisting of a Market Analysis and the Cost/Benefit studies, a Requirements Definition that includes the Product Acceptance Analysis and the Task Analysis, a Design stage that includes Prototyping and User Testing, a Test stage which includes the User Testing and an Update and Maintenance stage which includes the Product Survey. Because results from the human factors portions of these stages can have a direct effect on whether the project proceeds to the next stage or reverts back for more effort to an earlier stage, they are explicitly separated out in this paper. The next section describes the added costs of these projected changes and compares these to the benefits accrued from the inclusion of human factors efforts.

III. Cost/Benefit Analysis of the Human Factors Addition

In the paragraphs which follow, the projected costs for carrying out the psychological and marketing studies that form the human factors portion of the software lifecycle are given. The financial benefits that arise from conducting a Human Factors Software Lifecycle are listed after these costs. Some costs and benefits can be quantified but others are ambiguous organizational and sociological gains and losses. These are discussed in four separate sections, two for the measurable results and two for those which cannot be measured.

A. A Breakdown of Tangible Costs

In performing the cost analysis for this section of the paper, a $15.00 per hour wage rate is used for hourly employees and a $30.00 per hour wage rate is used for non-hourly employees. A prototypical software project is used in the calculations. The project is assumed to be built for 250 non-hourly in-house employees and to fall in the class of medium software projects, i.e., 32,000 lines of delivered source instructions (Boehm, 1981). Using these assumptions, a cost breakdown can be generated for the following human factors expenses.

Eight distinct costs are added to a project by the human factors stages. These are:

1. the cost of running focus groups,

2. the cost of building product mockups,

3. the expense of building a prototype,

4. the expense of making a prototyping design change,

5. the expense of purchasing the prototyping software,

6. the cost of running the user studies,

7. the cost of creating a user study environment and

8. the cost of conducting the user survey.

Two of these, the prototyping system purchase and the user lab construction are fixed costs whose amount per software project can be reduced in proportion to the volume of software projects that undergo a human factors inclusion in their design. Table 1 breaks out the various human factors techniques and/or tools used in each of the new stages added to the software lifecycle. The costs of these tools and techniques is discussed further in the paragraphs which follow Table 1.

Lifecycle Stage	Cost Item	Cost/Unit	Number	Total Cost
Market Analysis	Focus Group	$2,185.00	3	$6,555.00
Product Acceptance Analysis	Focus Group	$2,185.00	3	$6,555.00
	Product Mockup	$2,960.00	1	$2,960.00
	User Survey	$5,600.00	1	$5,600.00
Task Analysis	User Study	$5,520.00	1	$5,520.00
	Lab Construct'n	$16,000.00	1	$16,000.00
Prototype Construction	Initial Design	$4,800.00	1	$4,800.00
	Design Change	$240.00	20	$4,800.00
	UIMS System	$15,000.00	1	$15,000.00
User Testing & Evaluation	User Study	$5,520.00	4	$22,080.00
	User Survey	$5,600.00	1	$5,600.00
User Testing	User Study	$5,520.00	1	$5,520.00
Product Survey	User Survey	$5,600.00	1	$5,600.00
TOTAL				**$106,590.00**

Table 1. A Breakdown of the Costs Required to Add Human Factors Elements to the Development of Software.

1. Focus Group Cost Breakdown

The cost of running a focus group is the time cost of the individuals involved in the focus group plus a small equipment cost. This includes the participants, the moderator, the videotaping personnel and any additional staff watching the focus group behind a one-way mirror. Focus groups take two to three hours to run and two days to set up and break down. A minimum of two days of analysis is needed to assess the results from the focus groups. On average, ten future users form the focus group population and a support staff of three employees plus the moderator are needed. A complete focus group analysis, using three consecutive groups, takes two weeks time.

The projected costs for running typical focus groups are shown in Table 2. The costs listed for focus group type A are for an in-house focus group using participants and facilities from inside the company. Those costs listed for group type B are for a completely contracted service from a marketing research company. Although the costs are much higher for a type B focus group, the marketing firm provides the appropriate participants and facilities and carries out a thorough analysis of the participants' reactions.

Type of Expense	Amount
Ten participants at $30.00/hour	$900.00
Group Moderator at $1000.00/group	$1,000.00
Three Support Staff at $15.00/hour	$225.00
Videotape	$60.00
Total	**$2,185.00**

A. Cost of Operating an Internal Focus Group

Type of Expense	Amount
Fee Charged by Agency for Complete Study (3 Focus Groups and Analysis)	**$10,000.00**

B. Cost of Contracting an External Focus Group

Table 2. Estimated Costs for Conducting Focus Groups.

For each information gathering session, an average of three focus groups is run. This is done because the focus group involves a small segment of the user population and may be a statistical anomaly. Running three groups does not avoid this problem but it does lessen the probability of the problem happening. An external unbiased marketing research firm is essential for running the internal focus groups. It is very unlikely that employees of a company will express their honest opinions unless they are guaranteed complete confidentiality through the external operation.

An external research firm is also recommended for a focus group made up of individuals external to the company. Marketing firms have resources for obtaining participants and expertise in setting up the facilities for a good session that may not be available within the company. The cost of focus groups are fixed costs and do not change much with the size of the software project. They are dependent on the variability and size of the expected user group not on the code under development. More focus groups are necessary to capture data on a large and diverse user population. Table 1 uses the costs estimated for running an in-house focus group. Both the Market Analysis and Product Acceptance Analysis stages use focus groups.

2. Estimation of Product Mockup Costs

The task of building product mockups is one of constructing a false user interface scenario in software and generating a videotape of the scenario. The script for the voice overlay needs to be written and the actor/actress trained in executing the scenario. The videotape need not be of market quality, but it must not be too amateurish or it will negatively influence future users of the software. Preparation and Videotaping usually take two weeks. Large companies have an AV department with staff available for preparing such a videotape. Smaller firms can usually rent the appropriate equipment and hire a person to prepare the videotape at minimal cost. Table 3 illustrates the basic costs of this task. These can be lowered or raised depending on the desired quality of the videotape.

Type of Expense	Amount
Preparation of Mockup Scenario (40 hours)	$1,200.00
Videotaping Sessions (20 hours)	$600.00
Splicing/Integration of Scenarios (20 hours)	$600.00
Equipment Rental for Splicing, etc.	$500.00
Videotape	$60.00
Total	**$2,960.00**

Table 3. Costs Incurred in Building a Mockup of the Proposed Software System.

The videotape mockup is used in the Product Acceptance Test in two ways. A mockup is shown to focus group participants who are encouraged to vocalize their reactions to the software system. The mockup is also shown to audiences of projected users who are asked to fill out a questionnaire designed to elicit their assessment of whether they will learn and use the software being demonstrated. The survey also probes into the perceived needs of the user population and asks whether the software demonstrated in the videotape mockup will respond to these needs.

In addition to the human factors uses, the mockup can also be used for marketing. It can be shown to senior managers who wish to know what the software development group is working on and to potential investors or customers who wish to know about a proposed product.

3. Expense Layout for Conducting the User Survey

A user survey is the distribution of questionnaires to the future or current users. The questionnaire is designed to collect information from these users on their reactions to the software system. User surveys are employed in the Product Acceptance stage to assess future users' responses to the video showing of the product mockup and to capture their suggestions for changes to the product design. They are used once the product is released to find out difficulties users have with the working system, to

find out what tasks the system is being used for and to gather suggestions for changes to the existing system.

Questionnaire design is an art. Survey researchers who are skilled in questionnaire design know where to put particularly difficult questions, when a person will not be able to respond truthfully to a question, how to test for the truthfulness of a response and, in general, how to elicit the maximum amount of information from a respondent with the minimum amount of questions. Without a good design, the responses on a survey form are worth little. With this understanding in mind, a considerable amount of the expense for running a user survey is in the development of the questions for the survey. The cost breakdown for conducting a single user survey is listed in Table 4 below.

Type of Expense	Amount
Development of Questionnaire (40 hr)	$1,200.00
Pilot Testing of Questionnaire (40 hr)	$1,200.00
Distributing and Collecting Survey (20 hr)	$300.00
Coding and Entering Data (20 hr)	$300.00
Analyzing the Results of the Survey (40 hr)	$1,200.00
Cost of Time Lost in Filling Out Survey	$1,200.00
Computer Time	$100.00
Supplies and Duplicating Costs	$100.00
Total	**$5,600.00**

Table 4. Cost Breakdown for Running a Product Survey for the Software Product Being Tested.

For the user population of 250 employees in our prototypical example, at least half of the employees would receive a user survey. A typical survey is four pages in length and requires approximately half an hour for an individual to fill out. The cost for conducting a user survey is extremely stable. Any increase in cost depends primarily on the number of users who receive the survey. The number of users surveyed need not rise if a good sample is taken of this group of individuals but the task of taking an accurate sample will increase the overall costs of the user survey by $1,200.00 or forty additional man-hours. The entire survey requires approximately four weeks to run.

4. Initial Prototype Building Costs

Although considerable time is spent in building a prototype, the cost breakdown for prototype construction excludes much of this time as design time and presents only that time required to build the actual prototype. The estimates shown in Table 5 are based on a study which implemented a pre-designed system in three separate prototyping systems (Mantei and Culver-Lozo, 1986). The time to build the entire prototype is four weeks.

Type of Expense	Amount
Specifications of the Screen Transitions	$2,400.00
Design of the Individual Screen Layouts	$2,400.00
Total	**$4,800.00**

Table 5. Costs Incurred in Building a Prototype of the Proposed Software System. The Prior Development of a Global Design is Assumed.

Most prototyping systems require a two-stage design specification. In the first stage, the connections between the screen displays must be specified. The second stage then involves the design of each individual screen layout. Advanced prototyping systems do not group design units into screen displays but into states between user interactions. The second stage design process is then the design of the individual states and the alterations that take place because of the user's action. The design work required for each stage is approximately equal.

As the interface grows more complex, the time required to build the prototype increases. If the complexity of the interface is characterized by the number of states required and the average number of new details to be specified in each state, the cost of building a prototype can be written as:

$$C = S(a + bD)$$

Where C = **Cost**
S = **Number of states**
D = **Average number of new details per state**
a = **Constant reflecting the cost of building a single state**
b = **Constant reflecting the cost of adding a single detail**

The above model assumes little interconnection between states and the ability to copy detailed state descriptions from one state to another. These assumptions reflect a large variety of user interfaces and prototyping systems. The time projections for building the prototype also assume a powerful and flexible rapid prototyping package. Limited prototyping packages decrease the prototype building time because they are only able to produce simple interfaces, e.g., numerical menu selections. Using such systems is a dangerous practice. Their limitations may, in turn, limit the designer's conceptualization of creative interface designs.

5. Cost Breakdown for Design Changes to the Original Prototype

Once the prototype is built, the user tests will uncover difficulties that the user has in learning and using the system. The cause of these difficulties will be used to suggest design changes. Once the suggested changes are incorporated in the prototype, the prototype will be tested

again. The iteration of testing and updating the prototype will occur until the number and type of difficulties a user has with the system reach an acceptable level.

The initial user studies will uncover many problem areas. These may lead to redesigns for parts of the system. Later changes are minimal. Because a task analysis and user surveys were used to build the basic design of the prototype, the prototype is close to the final design. Therefore, the design changes are not expected to force a complete redesign of the prototype but only updates of various parts of the prototype. Because of this time saving, the average time estimated for a design change is one day. The cost of each change is shown in Table 6.

Type of Expense	**Amount**
Modification of the Screen Transitions	$120.00
Redesign of the Individual Screen Layouts	$120.00
Total	**$240.00**

Table 6. Costs Incurred in Incorporating a Design Change to the Original Prototype of the Proposed Software System. These Costs Assume that the Design Change Does not Require a Complete Redo of the Screen Transitions.

The number of changes expected and the amount of time a change will take is dependent upon the complexity of the interface being constructed. The relationship is the same as the relationship shown for building the prototype.

6. The Prototyping Software Purchase

Commercially available prototyping systems cost as little as $2,500.00 and over $15,000.00. Most systems that are powerful enough to provide graphics capabilities, design tools and system management tools cost close to $10,000.00.

The various prototyping systems differ widely in the features and support they provide. The choice of a prototyping system depends on the match of features to the type of interfaces typically constructed by the software staff and the requirements for project management. The review and selection process for a prototyping package is therefore expected to take at least a month. The time spent on making an intelligent purchase and the actual purchase cost constitute the complete price of the package. This is shown in Table 7.

Type of Expense	Amount
Time Spent Reviewing Potential Packages	$5,000.00
Purchase Cost of Package	$10,000.00
Total	**$15,000.00**

Table 7. Costs Incurred in Examining the Market and Purchasing a Suitable Prototyping System.

The actual cost of purchasing a software package for a single project is quite low if the cost is distributed over several software projects. This distribution is not assumed in our cost analysis. If it were, the cost of the prototyping software would be negligible.

7. Cost Breakdown for Running User Studies

A typical user study presents an individual with a set of tasks to perform. As the user performs these tasks, measures are taken on the performance. These user studies are akin to psychological research since they take place in a laboratory and collect data on individuals. They are different from psychological research because they have no hypotheses to prove and no experimental treatments to administer.

The user studies gather data, data on how people use the planned software system. The primary mechanism for gathering this data is videotaping. As a user tries various tasks with the software system or the prototype, a camera records the user's difficulties and successes on videotape. In most cases the individual in the study vocalizes the difficulties encountered as they occur. This person also describes how the system is believed to operate and various strategies for getting the system to perform a desired task. Secondary mechanisms such as keystroke records and recorded comments by an unseen observer can also be used to obtain data in these studies.

The information captured in a user study is analyzed for a variety of potential problems with the system under development. Among these are consistent errors that occur in a particular place in the interface, stated misconceptions the user has about using the software, extremely high task performance times and numerous requests for help made by users at key points in the task. The human factors and the design staff discuss each of the observed difficulties and agree on design changes that might alleviate the problems.

The major difficulty in conducting a user study is the preparation of material for the individuals being studied. Since the software system is new, a manual must be written. The manual need not be complete, but it must contain a complete description of those parts of the system which the the individual will use in the study. In addition to the manual, a set of directions and a set of tasks are needed for the study.

As with most written material, the first version of the manual, the directions and the tasks will be incomplete, obtuse and too difficult or too easy for the user study. To correct these problems, a pilot of the user study is run on a small number of individuals. The feedback from the pilot is used to rewrite the instructions. Following the testing of the study material, the user study is run and analyzed. The entire process requires a month. Table 8 presents a cost breakdown of the worktime used in the process and the videotape required.

Type of Expense	Amount
Development of Subject Directions (40 hr)	$1,200.00
Pilot Testing of Directions (20 hr)	$600.00
Redesigning Subject Directions (20 hr)	$600.00
Running Experiment (40 hr)	$1,200.00
Analyzing Results of Lab Study (40 hr)	$1,200.00
Videotape	$120.00
Cost of Subjects in Experiment (20 hr)	$600.00
Total	**$5,520.00**

Table 8. Costs Incurred in Conducting a Single Laboratory Test on Five Subjects.

Three types of user studies are run in the Software Lifecycle. The first of these is used in the Task Analysis. Instead of testing a software system on projected users, these users are asked to perform the types of tasks the software system will help them accomplish, but without the software system. Paper and pencil, calculators, file cabinets or an existing computer system may be used to replace the not-yet-built system. The videotapes of these sessions are used to build a model of how the users think about the tasks. This model guides the interface design.

The second and third user studies are more conventional with the second study conducted on prototypes of the system under construction and the final study conducted on the implemented system. A final study is always run because the actual system is different enough from the prototype system, that potentially serious user problems could be embedded in its design.

The cost of a single user study is typically not related to the complexity of the user interface being developed. Instead, if a user interface has many complex parts, the number of user studies conducted increases. This occurs naturally by the need to divide the interface into distinct user studies to avoid tiring the participants.

8. Costing the Construction of a User Laboratory

A laboratory in which to conduct user studies can be an office that is temporarily appropriated for the purpose or a permanent facility. Permanent space is used when user studies are planned every month for many separate software efforts. A user study laboratory is a mockup of the natural environment where the software system will be used, e.g., an office. The individual being studied works in this environment. Ceiling mounted cameras and a one-way mirror allow human factors staff to record and observe the study session. The observation room is built next to the user environment and contains the recording equipment and the monitoring computers.

Table 9 presents a cost breakdown for a small permanent facility, one which may be used four months each year for user studies. In constrast to the $16,000.00 price tag shown in the table, IBM has built a permanent user study environment for one million dollars. IBM rents these facilities for $50,000.00 for two weeks. On the other side of the coin, a laboratory was established in a home for $100.00 in video equipment rental costs. It was used to test the interface of a home computer product.

Type of Expense	Amount
Time Spent Laying out Laboratory Design and Selecting Lab Equipment (80 hours)	$4,800.00
Cost of Carpenter, Electrician	$1,200.00
Cost of Cameras, VCRs, One-way Mirror	$10,000.00
Total	**$16,000.00**

Table 9. Costs of Establishing a Permanent Human Factors Testing Laboratory. These are Mid-Level Costs. Much Fancier and Much Less Well-appointed Laboratories can be Built.

The cost of the laboratory is independent of the software system. If it is used to test many software projects, the cost per development effort becomes negligible. The user study area can also be reconfigured for running the focus groups.

The type of laboratory environment generates an image of the software product being tested. Therefore, it is appropriate to appoint the laboratory with the furniture and equipment that conveys the desired image. For example, a machinist trying out numerical control software would not receive the software tests comfortably in a plush office. Nor would a secretary enjoy a user test conducted in a laboratory full of wires and measuring equipment.

B. The Most Common Tangible Benefits Derived from Human Factors Design.

Although empirically gathered data is not presented here, the direct benefits from adding the human factors aspect to the project can be calculated by making several valid assumptions about the improvements to the interface. These improvements are:

1. a reduction in user learning times,
2. a reduction in user errors and
3. a reduction in the cost of maintaining the system.

The same size system used in calculating the added costs of human factors stages is used for calculating these estimates, i.e., a 32,000 delivered source instructions system to be used interactively by 250 non-hourly employees. A summary of the first-year savings from the human factors addition to the software development is shown in Table 10.

Type of Savings Incurred	Amount
Training Costs	$30,000.00
Error Reduction Costs	$85,000.00
Avoidance of Late Design Changes	$18,000.00
Total	**$133,000.00**

Table 10. Sample Estimates of First Year Savings Incurred Through the Introduction of Human Factors Elements in the Software Design Process.

1. An Estimate of Potential Training Cost Savings

It is estimated that the learning time for the new system will be cut by one-fourth by the development of a human-factored system. If the turnover rate is fifty employees per year and the learning time is typically two weeks of classes, the business has saved $30,000 a year in education costs.

(Savings/Year) = (Turnover) * (Training Time Saved) * (Wage)
Savings/Year = 50 * 20 * $30.00
Savings/Year = $30,000.00

2. Calculating the Cost of Errors

It is also estimated that at least one "user trap" occurs regularly in each database retrieval scenario. A user trap is defined as a standard sequence of user responses where the user consistently makes an error. The errors are usually negligible and easily corrected, but extremely annoying to the user. These traps are a result of the interface design violating the learned behavior of the user. (The experiences a driver has with a car

with an automatic shift after driving a standard shift car are analogous to these user traps.) The User Testing stage catches these problems

Suppose a user of the system had ten scenarios which they used regularly and a probability of falling into a user trap of 0.025. For a company with 250 employees using the system at least 3 hours a day and performing approximately 20 scenarios per hour, the company would encounter 750 traps per day. If it took 2 minutes to recover from each such trap, a total of 125 hours or $1,875.00 would be lost per week because of these unremoved difficulties. This is a lower bound estimate. A number of these traps take as much as 10 minutes to recover from. To estimate the savings per year involved in removing a user trap from the design, the number of man days per month are taken as 19 following the practice of Boehm (1981). Using the calculations below, an estimate of $85,500.00 is saved per year if a user trap with a 2.5 percent chance of happening is caught and removed from the system in the User Testing stages. For a firm with 25 employees, the savings is still considerable, $8,500.00 per year.

Errors/Year = (No Emp) * (P[Error]) * (Scenarios/Hr) * (Hrs/Yr)
Err/Yr = 250 * 0.025 * 20 * (19 dy/mo) * (3 hr/dy) * (12 mo/yr)
Errors/Year = 85,500

Cost/Year = (Errors/Yr) * (Hrs/Error Corr) * (Wage/Hr)
Cost/Year = 85,500 * (2 min/err) * (1 hr/60 min) * $30.00
Cost/Year = $85,500.00

3. Potential Savings Achieved by Early Change Detection

Harder to estimate is the amount of system maintenance time that was saved by engineering the system to match the thinking behavior and limitations of the users. The design changes that were incorporated in the prototype and the final system can be estimated to cost one-fourth of what they will cost to make in a released system. Let us assume that twenty-five necessary design changes occurred as a direct result of user tests on the prototypes. If these changes took, on average, a day to implement in the prototype, then $18,000 was saved through early design changes. It is true that some of these user problems may never have been found and changed, but then, a new cost is incurred via the user's difficulties with the particular system problem.

Early Cost = (Hr/Change) * (No. Changes) * (Wage/Hr)
Early Cost = 8 * 25 * $30.00
Early Cost = $6,000

Design Change Savings = (Late Cost) - (Early Cost)
Design Change Savings = 4 * (Early Cost) - (Early Cost)
Design Change Savings = 4 * $6,000 - $6,000
Design Change Savings = $18,000.00

A final very tangible benefit is that of system adoption. If careful planning is made for the system to meet the psychological and functional needs of the user, the system has a higher probability of acceptance and

use. In this case, the benefit is the entire cost of system development which is not lost.

C. Intangible Costs to Control For in Human Factors Software Lifecycles

Intangible costs can occur in a software design project when human factors elements are included in the design process. These costs arise mainly from the four situations explained in the following paragraphs.

1. The Selection of Non-Critical Design Decisions for User Testing

User interface design is a nightmare of detailed decisions. "Should the selected text have a shadow border when displayed in reverse video or will the existing border be discriminable?" "If we permit a very large field width on a form, how should the form be displayed when the user types in more information than will fit on the computer screen?" These questions are not always answerable by the direct application of user interface design theory. Therefore, a designer will often opt to build a variety of solutions in a prototype and evaluate these solutions through user studies. Although there are a great number of design decisions to be made in building an interface, a great many of them make no difference in the quality of the user interface (Norman, 1983).

Human factor researchers have considerable intuition on what design issues are important to test, but they can still be wrong. For example, the careful selection and testing of icon names to use in the Xerox Star interface did not reveal any differences in performance between the design choices (Bewley, Roberts, Schroit and Verplank, 1983). It is possible to both spend time on testing what appear to be crucial design decisions that make no difference in the final software and to miss testing design decisions that are essential to the effective operation of the interface. Running the wrong user tests costs time and money. Missing the necessary ones makes their change more expensive when discovered at a later stage of software development.

One approach to this problem is to make the testing process open-ended enough so that information is always acquired from a user test. To uncover missed design problems, followup user tests are run on the final product, which do not, unfortunately, avoid the additional expense incurred from the earlier omission.

2. The Establishment of too High a Level of Usability

The second cost problem with applying human factors elements to software development occurs when user interface standards are set too stringently for the software system. It is easy to set standards for learnability and usability on paper, but hard to meet these standards in the design of software. Often, the task of learning a software system is hard because the task the system is designed for is hard. In this case a performance requirement for the system to be learned in one week of training might be impossible because the actual task cannot be learned in one week.

Few benchmarks exist for describing the level of performance expected of computer users. For example, only two published studies describe benchmarks for text editors. Digital Equipment has collected data on user performance with their "vi" editor (Good, Spine, Whiteside and George, 1986) and Roberts and Moran (1984) have established ballpark performance levels for nine text editors. Since few standard performance levels are available, it is extremely difficult to set performance standards for system use.

Information collected in the Task Analysis stage can be used to set user performance levels for the system tests, but this data is not infallible. Therefore, it is important to establish incremental levels of improvement from current practices and to recognize that improvements to the user interface are iterative and will come with increased experience with new designs and new ideas for eliminating user problems in subsequent software systems. If this is not done, considerable time can be spent trying to implement a design to meet unreasonable requirements.

3. Falling into the Trap of Overdesign because of the Powerful Prototyping Tools Available

Because a prototyping system makes it easy to design changes to a user interface, a designer can fall onto the trap of building more and more bells and whistles into the user interface. The additions are not necessarily added functionality, but such detail as borders which uniquely identify different screen groupings or the installation of a running clock in the corner of the screen. The extra intangible cost is both that of the designer's time and of the time needed to implement the final design.

Avoidance of overdesign requires strong management control and a regular review of the design process. As with program design, errors can be made both on the side of underdesign and overdesign where necessary features are left out and unnecessary features put in. User tests will not indicate overdesign problems but will pinpoint systems that do not have enough careful user design considerations. Most projects will incur some wasted costs in this area. These amounts will drop as human factors personnel obtain more experience in working with different levels of design complexity.

4. Communication Problems between Psychologists and Software Designers

A large problem with current human factors efforts in software development is that of the knowledge gap between the psychologist carrying out the human factors aspects of the software building task and the computer scientists who are designing and building the system. The psychologist has been trained to recognize and interpret a wide variety of human behavior which the computer scientist will miss when observing a computer system user. Although the psychologist can recognize problems with system usage, the recognition of these problems has no translation into a software design specification.

For example, consider the following user problem that has been observed to occur with the Macintosh pulldown menu. Users find the task of holding the button of the mouse in depressed mode while positioning the mouse to be difficult. Many menu selection mistakes occur with this selection mechanism. The obvious solution is to allow the menu to remain open when selected. However, allowing the selected menu to remain open until a selection is made introduces a large number of other design decisions--decisions such as whether a "close" selection option needs to be included in each menu and whether the menu should close automatically if the cursor moves away from the menu window. The psychologist working with the software team is often not aware of the ramifications a corrected problem has on the rest of the interface.

Individuals with human factor's training do not have the bag of interface design tricks that software designers have developed through performing their profession. They cannot easily use a rapid prototyping system to design a user interface on their own. They need to work with individuals who have these design skills. This leads to a large communication cost. The psychologist needs to convey the entire nature of the user problem that is being solved and the software scientist needs to convey the possible solutions and their ramifications on other parts of the system. Until more expertise in the opposing areas is gained by both types of personnel, a communications overhead will exist on any project that incorporates the human factors methodologies. How much this overhead will cost has never been measured although Gould and Lewis (1983) and Grimes, Ehrlich and Vaske (1986) have captured a qualitative assessment of the problem.

D. Intangible Benefits Associated with Human Factors Software Development

A variety of difficulties can occur with software that has not been "tuned" to its user, difficulties which cannot readily be measured. Intangible benefits accrue when these difficulties are removed from the software. Three such common difficulties are listed and explained in the paragraphs which follow.

1. Reduced Adoption of Features Which Save Time

A reduction in feature adoption occurs when the complexity of the system causes its user to eschew learning advanced features. Davis (1985) has shown that users adopt the less complex software package if they can achieve the same functionality. Thus, if a task can still be carried out, albeit less efficiently, it is unlikely that more powerful features built into the system will be used. The Product Acceptance Analysis and the User Tests avoid the development of the unnecessary features and reduce their complexity, respectively.

2. Employee Disaffection Leading to System Sabotage

Many organizational situations can lead to employee sabotage. Being requested to use a system which is inappropriate, difficult or inadequate for a task that an employee needs to get done can cause intense

frustration. In the right individual, this frustration can be followed by typing in inaccurate data or reporting false system failures (Dowling, 1979). The focus groups used in the Market Analysis and Product Acceptance stages are designed to capture information on the receptivity of projected users for the software innovation. The Task Analysis is done to make the final product fit the user's conception of the task as closely as possible.

3. Reduced Ability to Solve Conceptual Problems Using the Software System.

If a software system requires an intense amount of concentration on detail in order to carry out a task using the software system, this concentration takes away from an individual's available mental capacity for solving the problem. Although the problem is solved, it may not be solved as creatively. This loss in creativity that occurs with the use of a difficult system is impossible to measure, but the loss may be very large. The User Testing stage is designed to remove system complexities that can hamper problem solving.

Although the aforementioned costs and benefits can and do occur, because they are not measurable, they are often discounted in decisions to include or exclude stages in software development. Unfortunately, the dollar figures associated with the intangibles are usually much larger than those associated with the tangibles.

IV. Recommendations for Human Factor's Inclusion

A large amount of project management data collected over a large variety of projects is needed to prepare a model for determining when human factors stages are to be included in the software lifecycle. The human factors savings as well as the costs of including human factors in the software are sensitive to a variety of phenomena. These include such items as the type of system user, the number of users of the system, the complexity of the user interface being built and the amount and type of human factors stages that are included in the software development lifecycle. Since a model cannot be built and a tradeoff analysis performed from the small amount of data available, the paragraphs which follow will discuss the qualitative aspects of the software project and make recommendations as to the viability of including various human factors stages in developing the software from this perspective.

The four factors, discussed previously in this paper, which make the inclusion of human factors cost effective are listed in Table 11. They are matched with the human factors stages of the software development lifecycle that are most relevant to reducing the costs associated with these factors.

If the use of the software or the software features being developed is discretionary, those stages which measure whether the software meets the needs of the user, both functionally and emotionally are important to add to the lifecycle. Performing a Market Analysis is a crucial step if the software is being developed for an external market, especially if the market is a mass market.

Cost Reduction Item	Related Lifecycle Stage
Increased System Adoption	Market Analysis Product Acceptance Analysis User Evaluation
Reduced Training Costs	Task Analysis User Testing
Reduced User Errors	Task Analysis User Testing
Transfer of Design Changes to an Earlier Stage in the Lifecycle	Prototype Construction User Testing on Prototype Product Survey (next redesign)

Table 11. The Human Factors Lifecycle Stages and the Type of Software Cost Reductions that They are Most Likely to Affect.

If the software will be used by a large number of employees, reducing both the training costs and the time lost to user errors will make the Task Analysis and the User Testing stages cost effective. The cost of running user studies rises with the complexity of the interface. Since an interface built for many users is somewhat more complex than one built for a few users, the cost of running the user studies increases slightly with the number of users.

The savings incurred from running the user studies rises dramatically as the size of the user population goes up. These two functions intersect, as is shown in Figure 4. For user populations larger than the intersection point, it is appropriate to include the human factors efforts in the software design. The reader should note that the size of the user population is calculated over the life of the software system. If a system has ten users who will use the system for the next ten years, the size of the user population is measured as one hundred.

With or without the User Testing, the Prototype Construction stage will cause design changes to occur at an earlier less expensive stage in the lifecycle. A prototype should always be used on complex projects where a later stage update or design change would be very expensive. A quick rule of thumb is to use a prototype when the cost of the prototype is less than one-fourth of the project cost. This assumes that the design changes that will come about later in the project will cost at least that amount of additional change effort.

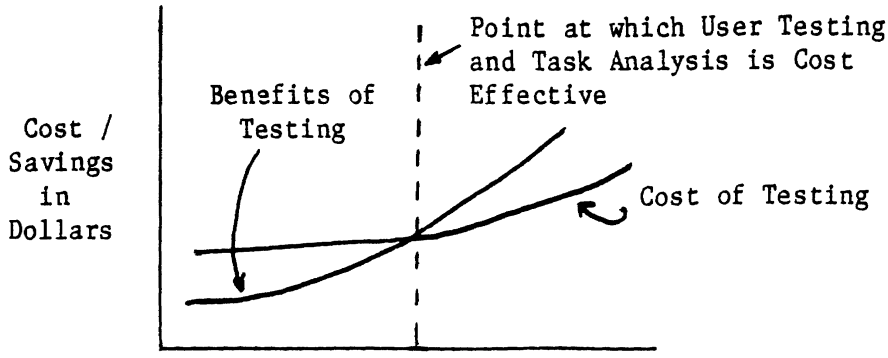

Figure 4. Cost of Running User Tests and Benefits Achieved from Running User Tests Graphed Against the Size of the User Population. When the Cost Curve Drops Below the Benefit Curve, these Steps can Achieve a Savings for the Software System.

In general, the human factors techniques are not recommended for small simple projects but for larger more complex interfaces, especially those used by a large number of people.

V. Conclusion

The goal of this paper has been to give the system analyst and project manager a description of human factors methods available for building a software product that meets the desires, needs and capabilities of its proposed user. These methods are incorporated into the existing management structure of software projects. An explanation of what they provide to the software development at each stage is given and a cost/benefit analysis is presented to provide a quantitative basis for deciding how to budget these methods.

Space is too limited to give a detailed description of the personnel needed to carry out the human factors tasks, to suggest organizational structures or to lay out the variations in the estimates that occur with differing project sizes. In short, this paper is intended to fill the current gap that exists between the human-computer interaction research papers and the pragmatic needs of the software developer.

VI. Biography and Address of Author

Dr. Mantei is currently an Assistant Professor of Computer and Information Systems at the University of Michigan. Her research focuses on the development of User Interface Management Systems and Methodologies for Incorporating Human Factors in the Software Design Process. She recently chaired the ACM CHI'86 Conference on Human Factors in Computing Systems and is currently Chair of the ACM SIGCHI Advisory Board.

Prior to joining the faculty at Michigan, Dr. Mantei worked at Carnegie-Mellon University and in human-computer interaction research at Xerox PARC. Before entering graduate school she developed very large database systems and modeling packages for Lawrence Berkeley Laboratory.

Marilyn Mantei
School of Business Administration
University of Michigan
Ann Arbor, MI 48109-1234
(313) 763-5936

VII. Bibliography

Aron, J. D. 1983. The Program Development Process Part II - The Programming Team. Addison-Wesley, Reading, MA.

Bewley, W. L., Roberts, T. L., Schroit, D. and Verplank, W. L. 1983. "Human factors testing in the design of Xerox's 8010 'Star' office workstation." Proceedings CHI'83 Human Factors in Computing Systems, pp 72-77. ACM, New York, NY.

Boar, B. H. 1984. Application Prototyping: A Requirements Definition Strategy for the 80's. John Wiley & Sons, Inc., New York, NY.

Boehm, B. W. 1981. Software Engineering Economics. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Budde, R., Kuhlenkamp, K. Mathiassen, L. and Zullighoven, H. eds. 1984. Approaches to Prototyping. Springer-Verlag, New York, NY.

Card, S. K., Moran, T. P., and Newell, A. 1983. The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

Davis, F. D. 1985. A technology acceptance model for empirically testing new end-user information systems: Theory and results. Ph. D. dissertaion, Sloan School of Management, Massachusetts Institute of Technology.

De Marco, T. 1978. Structured Analysis and System Specification. Yourdon Press, Inc., New York, NY.

Dowling, A. F., Jr. 1979. "Hospital staff interference with medical computer systems implementation: An exploratory analysis." Sloan School of Management Working Paper No. 1073-79, Massachusetts Institute of Technology.

Good, M., Spine, T. M., Whiteside, J. and George, P. 1986 "User-derived impact analysis as a tool for usability engineering." Proceedings CHI'86 Human Factors in Computing Systems, pp 241-246. ACM, New York, NY.

Gould, J. D. and Lewis, C. 1983. "Designing for usability--key principles and what designers think." Proceedings CHI'83 Human Factors in Computing Systems, pp 50-53. ACM, New York, NY.

Grimes, J., Ehrlich, K. and Vaske, J. J. 1986. "User interface design: Are human factors principles used?" SIGCHI Bulletin, Vol. 17, No. 3, pp 22-26.

Jackson, M. A. 1975. Principles of Program Design. Academic Press, New York, NY.

Mantei, M. and Culver-Lozo, K., 1986. "A proposed benchmark for testing user interface management systems." Working Paper, School of Business Administration, University of Michigan, Ann Arbor, MI.

Martin, J. P. and McClure, C. 1985. Diagramming Techniques for Analysts and Programmers. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Metzger, P. W. 1981. Managing a Programming Project. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Mills, H. D. 1976. "Software development." IEEE Transactions on Software Engineering, SE-2, No. 4.

Norman, D. A. 1983. "Design principles for human-computer interfaces." Proceedings CHI'83 Human Factors in Computing Systems, pp 1-10. ACM, New York, NY.

Norman, D. A. and Draper, S. W. eds. 1986. User Centered System Design: New Perspectives on Human-Computer Interaction. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

Roberts, T. and Moran, T. P. 1983. "The evaluation of text editors: Methodology and empirical results." Communications of the ACM, Vol. 26, No. 4, pp 265-283.

Rubinstein, R. and Hersh, H. 1984. The Human Factor: Designing Computer Systems for People. Digital Press, Bedford, MA.

Rubin, M. L. 1970. Introduction to the System Life Cycle. Handbook of Data Processing Management Vol. 1. Brandon/Systems Press, Princeton, NJ.

Vick, C. R. and Ramamoorthy, C. V. eds. 1984. Handbook of Software Engineering. Van Nostrand Reinhold, New York, NY.

Wasserman, A. I. 1982a. "Rapid prototyping of interactive information systems." Software Engineering Notes, Vol. 7, No. 5, pp 171-180.

Wasserman, A. I. 1982b. "The user software engineering methodology: An overview." In Information System Design Methodologies, ed. Verrijn-Stuart, A. A., pp 591-628. North Holland Press, Amsterdam.