NASA TM-1998-208470

Field Guide for Designing Human Interaction With Intelligent Systems

Carroll G. Thronesbery
Metrica

Jane T. Malin
Lyndon B. Johnson Space Center

July 1998

Available from:

NASA Center for AeroSpace Information
7121 Standard
Hanover, MD 21076-1320
Price Code: A17

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Price Code: A10

CONTENTS

Page

CONTENTS
(continued)

1. Introduction to Field Guide

The purpose of this document is to describe the methods we are using to design human interactions with intelligent systems which support Space Shuttle flight controllers in the Mission Control Center at NASA/Johnson Space Center. Although these software systems usually have some intelligent features, the design challenges arise primarily from the innovation needed in the software design.

While these methods are tailored to our specific context, they should be extensible, and helpful to designers of human interaction with other types of automated systems. We review the unique features of this context so that you can determine how to apply these methods to your project.

The most important feature of our software is that it is innovative, with a number of implications:

- No one can start the project by generating an adequate requirements description. Generating such requirements documents at the beginning would be a misuse of time—detailed requirements must be derived from initial design efforts and interactions with users.
- At the beginning, users cannot give an accurate, detailed description of what they would like the software to do because they have no previous experience using similar software.
- Developers may be unsure whether they can accomplish some of the technical objectives of the intelligent automation. The development is not routine.

Additional characteristics are defined by the Space Shuttle context. The highly trained flight controllers are engineers with specialized expertise in Shuttle subsystems. New software is integrated into a system of legacy software and processes. When software is certified for flight support, flight controllers become adept at using that software, but most do not have a programmer's depth of knowledge about the software.

Throughout this Field Guide, goals of the design methods are discussed. This should help designers understand how a specific method might need to be adapted to the project at hand.

This guide differs from other documents written by members of this group because it is oriented to human-computer interaction (HCI) design issues and is intended for use during HCI design. This guide is intended to have an applied orientation, in a style that provides a new team member with a written description of how the team works. It is intended to be like a field guide: open while designing and short enough to find an answer before the bird flies away. When examples are available, we provide links so that the application of a principle can be more concrete—in much the same spirit of providing pictures of birds in a field guide for bird identification to clarify their descriptions.

Our approach cannot be used as a cookbook, guaranteeing perfect "brownies" each time the recipe is followed. Instead, it is to be used as a quasi-checklist, helping to ensure that you haven't forgotten something important. It provides just enough reminders to assist a thoughtful team in building useful, usable applications of advanced technology while avoiding common distractions and traps.

The following steps are recommended for using this Field Guide efficiently:

1. **Preview the entire document,** to understand what the document has to offer and how it is organized.
2. **Read the "Rationale for Development Approach" (Section 2),** a short discussion of key characteristics of the methods in the Field Guide.
3. **Consult the "Development Activities and Products" section (Section 3),** to gain a firm understanding of the specific design products you are developing. Once you have become familiar with this design methodology, you can use this section to refresh your memory before focusing on a specific design product.
4. **Read the appropriate activity section (Sections 4 through 7),** where four of the key development products are discussed: task descriptions, scenarios, prototypes, and evaluations. These sections are intended to give in-depth descriptions about how to develop these products.

2. Rationale for Development Approach

The characteristics of this approach address the problems of designing innovative software to support user tasks. The requirements for novel software are difficult to specify a priori, because there is not sufficient understanding of how the users' tasks should be supported, and there are not obvious pre-existing design solutions. When the design team is in unfamiliar territory, care must be taken to avoid rushing into detailed design, requirements specification, or implementation of the wrong product. The challenge is to get the right design and requirements in an efficient, cost-effective manner.

Goal-oriented analysis and design help designers understand how users can be supported, and guide the wise use of development resources without cutting quality. Iterative spiral design and development and incremental deliveries provide the flexibility required to deal with the unknowns associated with innovative projects. Finally, participatory design ensures that all the necessary expertise is applied to the design process.

2.1 Goal-Oriented Analysis and Design

In our approach, determining system goals and selecting prototype objectives occur very early in the design process. Attempts to develop a detailed requirements specification or implementation early would be a costly waste of resources. The effort spent on identifying and validating goals and objectives is essential to keep the design and requirements on target. The goals and objectives are identified, validated, and prioritized by studying user tasks and defining scenarios that characterize the supported activities. These scenarios concretely embody the goals and objectives in a form that challenges the thought and imagination of the design team. They guide the planning of informal and formal evaluations. They provide a framework that guides the understanding of design risks and helps ensure that design shortcuts do not sacrifice quality. Even in projects where detailed requirements are provided up front, it pays to identify or develop such a framework early.

2.2 Iterative, Spiral, and Incremental Prototypes

Our iterative design approach addresses two sets of difficulties associated with the design of innovative software: Users often can't specify requirements very well and developers often can't "design it right the first time." In contrast to a requirements-centered, traditional waterfall model, an iterative prototyping approach allows users to interact with specific designs within the context of specific, realistic scenarios. They can respond to a concrete situation with specific, helpful feedback (e.g., "The scenario is unrealistic. The software doesn't help me with the hardest part of the task. I can't find the information I need to make the next decision."). Thus, users can voice their requirements better when the design process is iterative.

Iterative development allows discoveries during the design process to be incorporated into the design. It also allows the project to accommodate contributions from users, software engineers, subject matter experts, and graphic design specialists. Consequently, iterative design helps users and developers alike in dealing effectively with the uncertainty that accompanies innovative software.

Our iterative approach is closely related to spiral development, because each iteration of the prototype has different objectives. Product goals and objectives help determine the scope for each prototype. In our approach, early prototypes are far from implementations. Early designs address key design risks and core capabilities. Subsequent iterations will add new capabilities and refine existing ones, and move closer to implementation.

A related strategy to iterative development is the use of incremental deliveries. When feasible, early restricted-scope deliveries for use in the workplace let the users benefit from the new application earlier. This provides them with concrete evidence that you are building a useful application. As the scope of the application is broadened over time and as the existing capabilities are refined, users (and sponsors) can see progress, and can see the potential for improvements with additional funding. Also, as a result of an earlier, longer period of using the new application, users can provide more feedback during the project.

2.3 Participatory Design

Our approach includes participatory design because the team is composed of users and others with equally valuable expertise. The important ingredient is that the user's viewpoint is reflected in the design of the prototype. Not everyone will contribute in the same manner. In fact, depending on personal preferences, users will not contribute in the same manner for every project. Some users will prefer to help design the next prototype while others will prefer a more evaluative role, providing feedback on each design candidate. The following areas of expertise should be represented on the design team:

- user task performance (usually represented by a user)
- subject matter expertise (technical details about what's being monitored or controlled by the software—in our case, usually represented by a user)
- design and implementation of advanced software
- human-computer interaction design
- graphic design

Each team member contributes to the design process from his (or her) strengths resulting in a type of "design dialog." For instance, the following hypothetical scenario illustrates how a design team with the above areas of expertise can make significant progress on an issue in a short period of time—progress which, separately, they probably could not make at all:

> A user objects to proposed interaction, saying that he needs to see the interim results of a timer while the motor is driving, not just the final motor drive time. A short discussion between the subject matter expert and the software expert satisfies everyone that they know how to provide a reasonable, continual timer display. Then, the human-computer interaction expert suggests that even more information might be important—an indication of the expected drive time. The user concurs, saying that, in fact, that he'd like to see two expected times, one for single-motor and one for dual-motor drives. A short discussion among the user, human-computer interaction expert, and graphic designer yields a first cut at a graphic design that allows the user to watch the time for the motor and maintain awareness when it approaches alert or alarm limits.

3. Development Activities and Products

This section summarizes the development activities and products in our approach. Our approach includes the following activities performed over the span of a project:

- Select and Scope the Project
- Determine and Revise System Goals
- Develop Task Descriptions and Scenarios
- Select Prototype Objectives
- Design the Prototype
- Formally Evaluate the Prototype
- Define Requirements for Product Implementation

Figure 1 ("Development Activities Across the Project Life Span") shows how these activities and their products are related to one another. The arrows loosely indicate logical and temporal relationships. For instance, a project is usually selected and scoped before goals for the software system are determined (temporal), and the goals will usually depend on the reasons why the project was selected in the first place (logical). After goals have been decided for the entire software system, and as some task descriptions and scenarios are developed, specific objectives are selected for the first prototype iteration. The prototype is iteratively designed and evaluated within the design team. In this stage, informal evaluations are focused using scenarios and lead to regular revisions of the prototype and occasional revisions in the prototype objectives. When the prototype is ready, a formal evaluation is conducted. The developers decide if the improved understanding should result in minor modifications to the project goals, task descriptions, and scenarios. Then, objectives are selected for the next prototype. The cycle continues until the system goals have been met. Then a requirements document is written as a companion to the prototype.

Table 1 provides a list of development products. You should use this to identify products as the project progresses and to get some ideas about the length and content of those products. Not every project should generate every product listed in the table. The table is provided to remind you of the types of products to consider generating and to give you some indication of the length and level of formality with which they have been handled on projects in our lab.

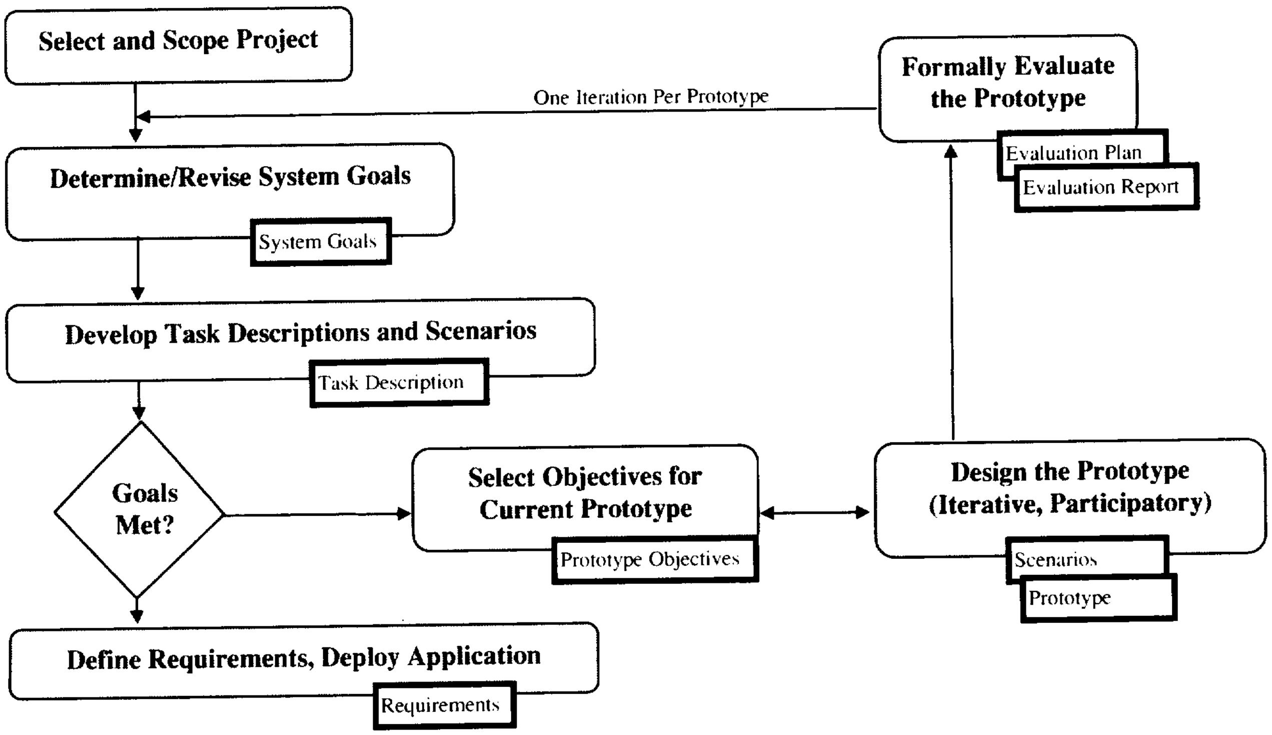
Select and Scope Project
One Iteration Per Prototype
Formally Evaluate the Prototype
Evaluation Plan
Evaluation Report
Determine/Revise System Goals
System Goals
Develop Task Descriptions and Scenarios
Task Description
Goals Met?
Select Objectives for Current Prototype
Prototype Objectives
Design the Prototype (Iterative, Participatory)
Scenarios
Prototype
Define Requirements, Deploy Application
Requirements

Figure 1. Development Activities Across the Project Life Span

Table 1. Development Products and Their Recommended Lengths

PRODUCT	WRITTEN LENGTH	FORMALITY
System Goals	a page or less	informal
Prototype Objectives	a page or less	informal
Task Descriptions	enough to develop scenarios and user descriptions	informal
Scenarios — generic	little detail (½ pp narrative)	somewhat formal
nominal	detailed	somewhat formal
failure	detailed	somewhat formal
variations	detailed	somewhat formal
integration	detailed	somewhat formal
Prototype — storyboard evaluation integration operational	code (document for storyboards)	formal
Formal Evaluation Plan	1-5 pp	formal
Formal Evaluation Report	5-10 pp	formal
Requirements	needed when prototype is to be re-implemented	formal

3.1 Select and Scope the Project

The first thing that must be done is to select and scope the project so that it will be successful. This is a difficult step, and probably most mistakes in selecting and scoping projects result from an incomplete consideration of what is required for project success. Because resources for advanced development are extremely scarce, our circumstances require vigorous support from the users and the proponents of the advanced technology we are applying. While users are often kind, morally supportive, and generally in favor of advanced automation, they won't get really excited about your project unless they believe it will make their job easier. Consequently, the following list includes not only items related to completing the project successfully, but also items related to making the new application useful. Here are some characteristics to consider when selecting a new project and scoping it:

- available expertise
- probability of successful implementation
- viewed by user as important to his job
- continually useful (frequent use)
- viewed by user as difficult, laborious, or error-prone
- useful early, allows incremental development where initial products (or iterations) are used while later ones are being developed
- allows demonstration of full range of proposed functionality

These considerations behind project selection and scoping are of central importance to our approach. They begin by influencing the determination of system goals and continue by guiding resource decisions about design risks and shortcuts.

3.2 Determine and Revise System Goals

Probably the greatest reservation held by newcomers to this approach is that the project might be allowed to wander toward first one goal, then the next, without knowing when the project should end or if success has been achieved. To prevent this from happening, system goals are identified at the beginning of the project. Sources of knowledge for setting these goals include the corporate knowledge of the users' tasks and how software can support those tasks, discussions with users, and discussions with people who can fund the project. System goals may need to be reviewed after each prototype evaluation for two reasons: to ensure that the overall system continues to follow the goals which are set for it, and to make minor adjustments to those goals based on an improved understanding of the users' tasks and how they can be supported. Consequently, the system goals should help to keep the project focused on solving specific problems, while allowing minor adjustments in the direction of the project.

The system goal statement could be as simple as a paragraph or two—rarely longer than a page. It's important, but doesn't need to be lengthy. It's probably good to have the goal statement in written form for team coordination and agreement. It is also desirable to state the goal in measurable terms (although, for the sake of brevity, you shouldn't include an evaluation plan in the goal statement). Measurable terms help the team determine when the goals have been met, and they are concrete enough to improve communication across the development team. In formulating the statement, consider what you want to demonstrate at the end of the project, what will make users (and funders) say it was a worthwhile project. It is also good to indicate priorities (not all goals are equally important, and they will often conflict). The system goal statement should build on the results of project selection. Try to get it mostly right at the beginning of the project and consider revising it when the objectives for the next development iteration are addressed.

3.3 Develop Task Descriptions and Scenarios

Designing is not just a matter of "doing it till you get it right" or, as a colleague termed it, "Now It's Through, What Do You Think?" (NITWYT). Instead, we use STEP—Scenarios, Task descriptions and Evaluation to focus Prototyping. The STEP approach helps to focus design efforts toward effective task support. Developing task descriptions helps to maintain awareness of the task being supported and the type of support that is needed. Scenarios are developed to focus the design and evaluation on realistic user tasks. Prototypes provide a concrete basis for design evaluation.

There is usually some confusion about the difference between scenarios and task descriptions. Task descriptions are abstract and general. The task description is a system-like description of the setting within which the new application will fit and an overview of the tasks performed in that setting. A scenario is a specific traversal through that system.

The task description should be written, since it contains design information not written anywhere else. The description contains information on the setting, tasks, and activities. Elements relating to the intelligent system include a description of the automation architecture, including the allocation (manual, automated, shared) of tasks between users and intelligent software and the information needs of the users, to perform tasks and coordinate with the intelligent software.

The level of detail of a task description depends on the demands of the project for which it is written, and on the maturity of the prototyping effort. It doesn't have to be done all at once. Initially, it can be an overview that focuses on typical tasks and typical users. When determining what is important for the task description, resist the urge to "dive into the details" prematurely. A good rule of thumb is that the task description should include just enough detail to guide the design of the next iteration of the prototype. The task description doesn't need to use models or formal analysis. While the task description in our method is shorter than a traditional one, it helps the team focus on goals, objectives, and priorities for design and evaluation.

Scenarios are specific concrete instances of a set of tasks, the surrounding situation, and how the software is used in carrying out the tasks. Described simply, a scenario tells a story. When data are received, specific values of that data are identified (not a range of possible values). When a decision point is encountered, one decision is taken (not a range of possible decisions). The form in which the scenario is expressed depends on the way it is to be used. It could simply be a narrative description (either spoken or written) of events that could happen during the use of the new application. At a detailed level, it might include a rigorously defined data structure that defines the information that is input during the scenario. Types and levels of scenarios vary over the course of a project, as the prototyping objectives change to reflect the maturity of the design.

Scenarios provide a focus for both design and evaluation efforts. Scenarios contain concrete, specific information, which assists a team with diverse backgrounds in communicating and coordinating their design efforts. Furthermore, scenarios tend to help the development team maintain its focus on design to support user tasks. Scenarios provide a context for evaluation and help users to focus evaluation on task support. Instead of simply asking users how they like the new application, a more focused question can be asked: "How well does it support you in performing this task?"

See Sections 4 and 5 for more details about developing task descriptions and scenarios.

3.4 Select Objectives for the Current Prototype

Using a spiral development model (Boehm, 1988), a subset of the overall system goals is selected for each prototype. With each additional prototype, a larger subset of the system goals will be addressed. The formal evaluation of a given prototype will probably reveal that some of its objectives were not met. Thus, the objectives for the current prototype should include unmet objectives from the previous prototype as well as some new system goals that had not previously been addressed. This process should continue until the system goals have been met and the project is completed.

The order in which system goals are selected for prototyping depends on a number of things:

- For HCI issues, prioritize to support understanding, then utility, then usability.
- Support for common tasks is usually prototyped before support for unusual tasks.
- Support for tasks critical to the success of the application is usually prototyped early.
- Difficult software techniques are usually prototyped early.
- High-risk issues are prototyped before low-risk issues.

The general progression in which HCI objectives should be addressed is illustrated in Figure 2 ("Changing Focus Across the Project Life Span") (also see Woods, et al., 1996). The upper portion of this figure shows a progression in which HCI issues are addressed. Early efforts should focus on identifying the right user tasks to be supported (understanding). Once critical user tasks have been identified, the focus should shift to finding the successful strategies for supporting those tasks (utility). Finally, efforts should shift to identify good display forms for implementing those strategies (usability). This general progression helps to steer the project into a productive expenditure of development resources. Following this progression, the development team confirms its understanding of which tasks are critical before expending resources to find effective strategies for supporting those tasks.

Sometimes, it is necessary to prototype alternative strategies for task support to make sure that you have truly identified which tasks should be supported. Likewise, it is sometimes necessary to prototype alternative implementations of support strategies to test the relative value of those strategies. Because new project goals are added to each prototype iteration, a given prototype iteration may address some issues at greater maturity levels than others. For instance, a given prototype might focus on finding a good strategy to address one problem and on finding a good implementation of another strategy. Thus, while the progression shown in Figure 2 is about right, it should be used as a suggested order, not as a required sequence.

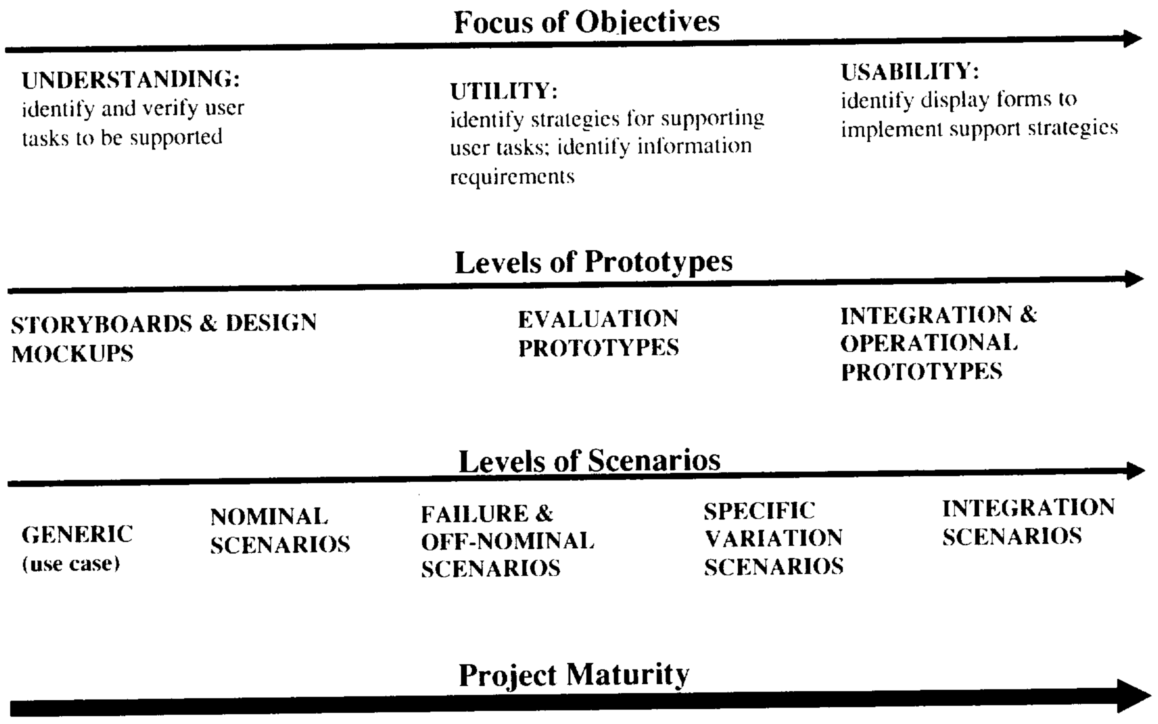
Focus of Objectives
UNDERSTANDING: identify and verify user tasks to be supported
UTILITY: identify strategies for supporting user tasks; identify information requirements
USABILITY: identify display forms to implement support strategies
Levels of Prototypes
STORYBOARDS & DESIGN MOCKUPS
EVALUATION PROTOTYPES
INTEGRATION & OPERATIONAL PROTOTYPES
Levels of Scenarios
GENERIC (use case)
NOMINAL SCENARIOS
FAILURE & OFF-NOMINAL SCENARIOS
SPECIFIC VARIATION SCENARIOS
INTEGRATION SCENARIOS
Project Maturity

Figure 2. Changing Focus Across the Project Life Span

The sequence in Figure 2 doesn't consider the relative risks of advanced software techniques. They need to be considered along with the HCI objectives. First address those objectives which, if unmet, pose the greatest risk to the project. For example, if the project involves a tricky case-based problem-solving technique and failure to apply the technique properly will kill the project, then an early prototype should include an attempt at implementing the proposed case-based approach. If the success of the project depends on displaying operational situations for review, then the first prototype should include the first attempt at that display. Note that, as high-risk objectives are addressed and the next prototype is evaluated, the relative riskiness of the objectives changes. If the initially highest risk objective has been addressed adequately with the previous prototype, it is no longer a high-risk objective. Other objectives can take focus in the next prototype.

Ultimately, the decision about the order in which to address system goals is based on a consideration of risk: high-risk issues are prototyped early. There are three reasons for this risk-driven approach. The first is that if the project is doomed to failure because it simply cannot be done, it's better to learn that information early—before development resources have been wasted. The second reason for addressing high-risk objectives first is so that you will have more prototype iterations to find the right way to address the most critical objectives. If your first attempt is successful, you can address some of the lesser objectives next and possibly finish the project ahead of schedule. If not, you can use what you've learned from the first prototype to formulate the next attempt. If early attempts prove that the goal cannot be met, there may be sufficient time and resources to revise the goal so that the project can still provide a useful software application. If, on the other hand, you address the riskiest goal last and fail, you're simply sunk. Finally, the elements of system design are interdependent. Settling on a specific implementation of one aspect of design sometimes constrains others. Using the risk-based approach ensures that the most critical aspects of the design are the least constrained by the remainder of the system.

3.5 Design the Prototype

Designing the prototype is so highly iterative, that it is misleading to think of the component activities occurring in a particular sequence. However, it is important to give attention to the component activities.

The prototype is a tool for determining the requirements of the application. In some cases, the prototype will evolve into the final, delivered application. In other cases, the functions embodied in early prototypes will be re-coded in a more capable implementation language to form the delivered product. Regardless of whether the final implementation is the result of a gradual evolution or the result of a re-implementation late in the project, earlier prototypes are used as a means of identifying appropriate requirements for improving user task support by the application.

All other products for the project are developed to ensure that the application is a good one. These other development products are interdependent with the prototype. Not only do the other products guide the development and evaluation of the prototype, but also the prototype helps to refine the other design products. The prototype provides the users an opportunity to evaluate the accuracy of the information in the scenarios, the task analysis, and the requirements. By interacting with the prototype in the context of the scenario, users are better able to judge how well the new application will support their task performance. Thus, one of the primary functions of the prototype during most of the project is to refine the requirements of the intelligent system.

The prototyping section below (Section 6) provides information about how to design human interaction during the prototyping process and about levels of prototypes, which serve different functions as the development project matures. The levels of prototypes include:

- storyboards
- evaluation prototypes
- integration prototypes
- fielded (operational) prototypes

Another key to good design is to make certain that it represents a collaboration of several viewpoints and levels of expertise. As was illustrated earlier, this participatory design enables the team to design quickly so that the resulting prototype is acceptable from many viewpoints at once. The reason for the speedy progress toward an acceptable design is the simultaneous informal evaluation by team members representing multiple viewpoints and levels of expertise during the design process. Unacceptable designs are identified quickly and modified based on a clarification about what makes them unacceptable. The informal evaluations are focused on relevant issues by using the design scenarios.

Figure 1 identifies only two explicit products from the design process: scenarios and the prototype. Because of the highly iterative nature of the design activities, we recommend waiting until after a design is completed before trying to capture the requirements and revisions to task descriptions in document form. This should save you a considerable amount of editing. The scenarios and prototype, on the other hand, need to be updated during the design process in order to coordinate the participatory design effort.

3.6 Formally Evaluate the Prototype

Evaluations in our approach are more formative (providing information about how to improve the prototype) than summative (evaluating how well the project objectives were met). Both informal and formal evaluations emphasize formative evaluation: they guide how each development product (prototype, scenarios, task description, goals, evaluation plan) should be improved. The purpose of evaluation is to determine which features of the prototype (also objectives, task description, and scenarios) need to be changed. The informal evaluations take place during iterative design, thereby guiding the design. The formal evaluations are also formative, identifying issues that need to be addressed in the next prototype. If the evaluation indicates that the changes should be numerous, it might be necessary to reexamine priorities so that the most important changes are made in the next iteration of the prototype.

Formal evaluations involve several users, mostly from outside the design team. They help to ensure that the design hasn't been biased by the idiosyncrasies of one or two users. This also helps to ensure that users who haven't been intimately involved in the design of the prototype can also use it. These evaluations are planned more formally, and more formal data collection techniques are used. In addition to examining things like understanding of the user tasks, improving the usefulness of the prototype, and improving the usability of the prototype, formal evaluations also address specific issues for the next prototype iteration.

The Evaluation Plan for a formal evaluation should tell how the application will be evaluated. It should focus on what will make the current project a success (project goals, current prototype iteration goals). It should identify specific evaluation activities, including evaluation scenarios and interview questions. For a formal evaluation of a major prototype iteration involving several users, it should identify a detailed usability test procedure.

Section 7 contains a detailed discussion of the evaluation process.

3.7 Define Requirements

When the project goals have been met, then the design phase of the project is over. Requirements can be written, for implementing and deploying the application. The prototype embodies the bulk of the requirements, and a small, supplemental document identifies the critical portions of the prototype and the purpose behind the designs that appear in the prototype. Putting the requirements set (prototype plus document) in the language of the user makes it easier to understand and evaluate the requirements. This also makes the requirements less expensive to generate, and aids programmers re-implementing the prototype. The prototype shows the new implementers a good implementation, and the requirements tell the purpose behind the design of the prototype. If the new implementers need to change the design to fit the new environment, they can make informed design decisions.

Since the requirements document supplements the prototype rather than trying to carry the entire specification burden, it can be much shorter than the traditional requirements document. Traditional requirements documents are exhaustive accountings of software functions, data interfaces, and human interaction design. They are expensive to write and often difficult to read. They are difficult to evaluate because they are too detailed and written in the language of the developer, not the user. Their length gives the user an impression of completeness, but makes it difficult to verify completeness.

4. Develop a Task Description

The development of task descriptions and generic task scenarios is commonly called task analysis. Task analysis for our method of designing human interactions with intelligent system differs from a traditional one by being much briefer, more flexible, and including an extra phase where a team architecture is considered. A traditional task analysis can be exhaustive, including multi-level descriptions of current tasks, reorganized tasks in light of the new applications, new allocations of tasks, etc. However, projects rarely have sufficient resources for such an exhaustive, traditional task analysis. Furthermore, by doing all the task analysis work before any design begins, the traditional methods tend to make the task analysis a static product. An unfortunate consequence is that if the analysts are mistaken about the task, there is no graceful recovery.

Finally, being static, traditional task analyses don't accommodate the appearance of a new team member, the intelligent system. By contrast, consider how a new human assistant would be introduced into the work situation. Some parts of the task would be given completely to the new assistant. As the assistant demonstrates competence, he may be given more tasks. A way of sharing information would be "worked out" with the assistant. More importantly, a way of doing business would be developed to change the allocation of tasks dynamically during critical times so that the senior partner (the human) can make the really tough decisions. Thus, in order to accommodate an intelligent system that can provide varying levels of task assistance, task description methods need to flexible and need to be able to accommodate intelligent system characteristics that cannot be known at the outset of the project.

4.1 Uses of the Task Description

It is important to avoid the common mistake of assuming that the most exhaustive task analysis will be the most useful. While we agree that understanding user tasks is imperative, our recommendation is to identify the uses of the task description and to do no more than is needed to satisfy those uses. In this manner, the design is not delayed and effort is not wasted writing parts of a document that are never used. Also, while we agree that the bulk of task description is done before development, we recommend revising the description throughout the project before designing each new prototype. This preserves flexibility and also discourages developers from thinking of the task description as a static product that must be perfected before beginning any design efforts.

The principal uses of task analysis are to:

- Identify, confirm, or revise goals, objectives, and priorities for design and evaluation:
  - critical functions of the software and user strengths that should be supported
  - task difficulties and bottlenecks, the main problems for users that the software should help overcome
  - current human tasks which are performed frequently or consume considerable time, which could be expedited by software assistance
- Identify the setting—other software, tools and resources that are used, the situation in which use of the software is embedded (helps identify conflicts and needs for consistency)
- Identify users—types of users and their characteristics
- Help identify task allocations and team architecture for the system of humans and intelligent software
- Help identify user interface design elements and information needs—flow of tasks, steps, choice points, task variations
- Perhaps eventually lead to user models (how the user thinks about use of the software—user actions on objects when using the software, preparation and cleanup) or to formal analysis of sub-task microstructures (We have not found it helpful to do such detailed modeling or formal analysis.)

4.2 Elements of the Task Description

While the uses of the task description should be the definitive driver of what information to include, it is sometimes helpful to see a list of candidate task description elements. You will need to consider whether the items on the list below are related to the uses your project will make of the task description. This list is provided to help you to avoid overlooking information that might be important for designing your application.

- Description of users
  - Level of expertise in subject matter
  - Expected frequency of use of new software application
  - Level of expertise with software in general
  - Other relevant user biases and expertise which should influence HCI design
- Purpose of overall person-system task
- Task list, task descriptions—list or diagram of activities and tasks, perhaps showing ordering, grouping, and dependencies among activities
- How software will support that overall task
- Identification of what is currently difficult (may want to get quantitative measures so that effectiveness can be demonstrated later)
  - error prone
  - time constrained
  - laborious

- Information Requirements
  - information needed by the user at each step
  - information the user provides the software (and how to make it easy to provide that information)
- Management of intelligent system—team architecture
  - keeping user informed (current state, controls issued, effects of controls issued)
  - providing controls to user
  - sharing tasks
  - handing off tasks between the user and the intelligent system
- Failures of monitored system
  - how to avoid them
  - how to recover from them
- Failures of new software application
  - how to avoid them
  - how to recover from them

It should be noted that the above information can be made even more useful to the design process when combined with scenario information (Section 5). For instance, the above information includes the identification of information which must be exchanged between the user and the new software application (user needs to see elapsed time). Additional design constraints include how this information needs to be displayed (user needs to compare elapsed time to expected time) and the type of interaction needed (user needs to be able to reset the timers at any time).

4.3 Sources of Task Analysis Information

Gathering task analysis information efficiently requires some knowledge of sources and how to use them. If it exists, a corporate knowledge base is a good place to become familiar with user tasks. This knowledge base should include written task descriptions from previous projects. This should help you gain a better understanding of the tasks performed by these users, what makes those tasks difficult, and the strategies that have successfully helped users with those tasks in the past. If that information is not written, it may still be available by asking people who have developed similar applications. Unless previous applications support the exact same task as the current one, this source is probably good for gaining a general familiarity with the users' tasks.

The next recommended source is the user representative on your design team. In the initial visit, you want to gain some understanding of the relevant tasks, what makes them difficult, what strategies have been tried to deal with those difficulties, which strategies were successful, and which strategies were not. In addition, it's a good idea to ask about good written sources for learning more about the tasks. Written sources allow you to become more familiar with the users' tasks without having users continually "spoon-feed" you. Reading them communicates your respect for the value of their time. Written documentation is also probably less likely to make small errors in details than the users themselves would. However, written sources should be discussed with the user. Users can help you consult the most reliable sources. Usually, you have two objectives when looking at written documentation: (1) Build better intuitions about your users' tasks, and (2) get a lot of details when building specific HCI designs. It's probably not productive to study this documentation enough to try to duplicate the users' expertise—They have already studied, they have already practiced, they are good at it, and they are available for consultation. Below are some written information sources we have found useful for our projects:

- console handbook
- training manuals
- operations checklists
- reports (post-flight, post-phase)
- flight controller written flight logs
- telemetry logs

Finally, it will be necessary to observe users performing the tasks. At a minimum, this is to acquaint you with small details no one considers important enough to mention. (Oh, sure, we always write these figures down on a scratch pad because we need them later.) Depending on the application, observations can also reveal things about the task that users never noticed. (For example, because they are so intent on performing the task, they may not even notice that they have to place their fingers on the screen to mark places for repeated scanning and data comparisons.) Also, because some tasks require coordinated efforts of many people, there may not be a single individual who is intimately familiar with every aspect of the task.

In the interest of good risk management, you should consult with multiple users to gain a good understanding of the task early in the project. The user representative on the design team will have unique characteristics and may approach tasks in a slightly different way from some other users. Your broader experience with users early in the project will give you a better perspective from which to view the inputs from the user representative. It will also help your communication with the user representative by giving you a more common understanding with him or her from the beginning. Later, evaluating each prototype iteration will give you the opportunity to verify (and refine) the generality across the user population of the task information provided by the user representative.

4.4 Allocation

Task analysis may include allocation of tasks to the intelligent system, to the human, and to both (shared tasks). In this activity, all the development team members can have active roles. The user knows what tasks have been error-prone, laborious, and time-consuming. The user also knows the times when things get so hectic they are difficult to manage--times when some of the work needs to be off-loaded to the intelligent system. The subject matter expert knows what tasks are critical and demand the highest levels of accuracy. The human factors engineer knows the cognitive/perceptual strengths of humans (creativity, insight, dealing with the unexpected, evaluation of trends, etc.). The software engineer knows what software systems do well (vigilance; comparing large sets of numbers repeatedly; executing rapid, routine sequences of commands; making elaborate logical evaluations; entertaining multiple hypotheses simultaneously; etc.). As a team, they should be able to allocate tasks so that the new application will be useful, usable, reliable, and safe.

As stated earlier, these methods are to be applied with thought and creativity, not with mechanical rigor. In light of this consideration, the initial task allocation might be attempted as part of the generic scenario descriptions. The advantages of this suggestion are that (1) the entire team can be involved in something more concrete (easier to visualize) than the typical task description, (2) it maintains a focus on the overall objectives of the user tasks, and (3) there is less of a temptation to "dive into the details" prematurely.

4.5 Team Architecture

Task allocation can lead to definition of a team architecture. A team architecture specifies how human and intelligent system activities will be coordinated. It is an important aspect of building reliable systems because it also provides alternative ways of achieving goals when planned activities are not effective:

- degraded performance due to high workload
- goal or plan changes due to failures in the domain system being monitored

The team architecture identifies alternative ways of accomplishing tasks, and it defines points of coordination between the human and the intelligent system. Even an extraordinarily talented software development team cannot anticipate every situation that the intelligent system will encounter. A good intelligent system-human team architecture will preserve the human capability to take over a task when an unanticipated situation is encountered.

Providing summaries of intelligent system activities and making raw data continually available to the user is a good way to preserve human options for taking over intelligent system tasks. When it is necessary for the intelligent system to interrupt the human, it should assist the human in responding to the interruption. Sometimes message priorities are used to assist the human in prioritizing responses to these interruptions.

A common way of implementing dynamic task assignment is to provide alternative task allocations in the form of multiple modes of operation. For example, the intelligent system may have different control modes that vary its level of autonomy in executing activities. Modes enforce task reassignment by constraining what the intelligent system can do in a particular mode. Modes can also provide a context for the presentation of information. Intelligent system capability, information, and style of interaction can all vary by mode.

As with the task allocation, you are encouraged to consider initially addressing the architecture with scenarios. They help to maintain a focus on the user's tasks, they help to make development team contributions more concrete, and they encourage the right level of detail for early architectural decisions.

4.6 Initial Generic Scenario Description

Generic scenario descriptions, the first type of scenarios to be constructed, are an important product of task analysis. They are abstracted narrative profiles of the most common types of situations the new application is intended to support. They should read more like a narrative than a system description or a list of requirements. The intent of the generic scenario is to identify how the new application will support users, to outline the task allocation, and to outline the user-intelligent system team architecture. Generic scenarios assist the development team in establishing the objectives of the new application.

Generic scenarios should illustrate what makes the new application useful. While they are important, they need not be long. Their purpose is to build a common understanding of how the new application will support user task performance. When done well, they will guide decisions about

- allocating tasks to the person, the intelligent system, or both
- the important information to capture in the task analysis
- how to demonstrate the prototype to show off its capabilities and usefulness
- how to evaluate the utility and usability of the prototype
- additional scenarios to build for evaluating prototypes as the project matures

4.7 Iteration and Progression Over Project Life Span

Construction and evaluation of the first prototype results in improved understanding of the users' tasks, their allocation, and the team architecture. This information can be helpful in documenting the system for user training and for future software maintenance. As understanding is improved and details stabilize, they should be identified in the task description to improve coordination among the development team and to expedite future system documentation and evaluation. Improved understanding of allocation and architecture policies are identified as much for development team coordination as for system documentation. Efficient expenditure of development resources is a prime consideration. Details are captured when they are readily available (as they are uncovered) rather than after they are partially forgotten. Furthermore, details are captured only to the extent that they will be useful for team coordination, design or evaluation decisions, or specific training or maintenance documents. Traditional, final-form system documentation is not attempted until the system is formally delivered.

Iteration is a means of spending fewer development resources while controlling the risk of having an incomplete task analysis. Initially, you should perform enough task analysis to gain a general knowledge of the users' tasks that will be supported by the new application. In addition, the specific tasks addressed by the first prototype iteration should receive a little more attention. If designing the prototype reveals an incomplete understanding of the task, then the task analysis should be revisited. Also, when additional functions are added to future prototype iterations, additional task analysis will probably be required. Thus, by iterating between design and task analysis, you can be assured that the details you add to the task description are indeed useful to the design process.

4.8 Implications for Designing Human Interaction

The task analysis yields requirements for human interaction design. It identifies information that should be exchanged between the human and the intelligent system, the task architecture for tasks shared by the human and intelligent system, and restrictions on the way in which the human and intelligent system will interact.

If the application is innovative, the task analysis is almost guaranteed to be inaccurate. Consequently, while the task analysis is a good place to begin the design of interaction between the intelligent system and the human, the process is iterative. Problems may be discovered in the task analysis during human interaction design, and you may need to refine the task analysis by comparing it to a scenario which, in turn, may force a refinement of the scenarios. These changes may also have implications for the evaluation plan and the goals for the next prototype iteration. This activity helps to inform your human interaction design choices and improve the prototype.

4.9 Implications for Evaluation Plan and Scenarios

Task analysis and generic scenarios can help focus prototyping by leading to an initial definition of an evaluation plan and fragments of evaluation scenarios. While we don't recommend trying to complete these products before designing the prototype, we have found rough drafts of these products helpful in focusing the design efforts toward the upcoming evaluation.

4.10 Final Cautions

In summary, it is imperative to understand the users' task in order to design a software system to support that task. That understanding will influence not just the prototype designs; it will influence the development and revision of every development product. However, it is also imperative to avoid two common mistakes associated with task analyses. First, you should not try to have a completely finished task description before you begin your prototype design. Even if you could write an accurate description at this point, the prototype itself would probably render it obsolete. Second, you should not write a more exhaustive task analysis than you plan to use in your project. There is no guarantee that anyone beyond the current project will ever want to read the task analysis, and it will probably be rendered obsolete when the new software system is deployed. You will experience less of a tendency to commit these mistakes if you make certain that you have user involvement throughout the design process because the users themselves can be the best providers of task information.

5. Build Scenarios

In this section, we describe how to select events for a scenario, how to identify what type of scenario it should be, and how to construct scenarios. Scenarios are selected and developed to influence both design and evaluation of prototypes.

5.1 Scenario Selection

Selecting of events for scenarios is similar to selecting a project in that two questions should be kept in mind: "Is it worth doing?" and "Can we do it?" The user representative on the development team should help with the assessment of what is valuable to demonstrate. "If we include this event (in the scenario), would it demonstrate valuable assistance to you?" (Is it worth doing? Will it illustrate the successful support of a part of the task which is currently difficult? Does it highlight the aspects of your job that you currently find difficult, which need better computer support?) The subject matter expert should assess whether the events in the scenario are probable, possible, and plausible. The software engineer, human factors engineer, and graphics designer should all assess whether the scenario can be supported with the next iteration of the prototype (Can we do it?).

5.2 Levels of Scenarios

As a project progresses, different types of design scenarios are needed. Each type of scenario identified below serves a different purpose. We have found the following progression to apply to our projects:

1. generic scenarios (developed as part of the task description)
2. nominal scenarios
3. failure and off-nominal scenarios
4. specific variations and less frequent scenarios
5. integration scenarios

This progression, also illustrated in the lower portion of Figure 2, is only approximate. Generally, there should be a progression from simple scenarios to complex; from frequently occurring to rare; from scenarios featuring absolutely required capabilities to those featuring nice-to-have capabilities. For formal evaluation, events are extracted from these design scenarios and recombined into new scenarios.

5.2.1 Nominal Scenarios

Nominal scenarios describe the most common situations that will be supported by the new application. Unlike the generic scenarios, the nominal scenarios are concrete enough to be specified with a time-ordered event list. In fact, in many instances, a simulated data stream with these common scenarios should be built to evaluate the prototype.

Nominal scenarios are used to design the first truly interactive prototypes. They don't include off-nominal events because the prototype is not ready for that level of sophistication. These are the first prototypes to include both user interaction and intelligent system functioning. The nominal scenarios are the first to have sufficient detail for testing an interactive prototype. As a consequence, they are the first that can provide concrete evidence whether the newly introduced task of managing the intelligent system has been designed well. It is important that when you introduce new intelligent system functions, you also develop scenarios that enable the users to evaluate their ability to perform the emergent tasks of controlling and coordinating with the intelligent system.

5.2.2 Failure and Off-Nominal Scenarios

Once the prototype has been proven useful in the context of the nominal scenarios, it is time to expand its ability to deal with errors. This is the role of the failure and off-nominal scenarios. There are three classes of error that should be included in this group of scenarios.

The first class is failures in the monitored process. Most of the software we develop monitors external processes to help a human operator recognize and identify failures and off-nominal conditions. An example of a failure in a monitored process would be an aborted end effector grapple for the Shuttle arm.

The second class of failures to represent in these scenarios is human error. The design of human interactions with an intelligent system should include the prevention, reversal of, and recovery from human error. Examples of this type of error can range from a data entry error to a Shuttle arm operator initiating a procedure before all the preconditions have been met.

Finally, the third class of failures is that of the intelligent system errors. It may be painful to admit, but your intelligent system will make mistakes. Also, even though there may be no fault in the intelligent system, the user may need to reject its conclusions or override a recommendation. Scenarios that illustrate the ability to override individual conclusions and the ability for the user to take over intelligent system tasks are important for building user confidence. These scenarios show users that, even if the worst thing happens (they don't trust the expert system), they can always take over its tasks. The objective is not to build a perfect system, but rather to build a system tolerant of failures. An interesting example of this type of scenario was provided early in the DESSY project (Malin, Thronesbery, & Land, 1993). The scenario was that there would be a loss of signal from the Shuttle for a few minutes, during which the arm would have been moved to a new position—which would appear to the software system as a sudden change from one position to another without having occupied any of the interim positions. While it is not common to receive arm configuration data in this sequence, it does occasionally happen (because of data dropouts), and using that scenario helped to make DESSY a more robust application.

Failure and off-nominal scenarios can usually be built more easily once the common scenarios are well understood by the design team. While it is probably not possible to be exhaustive in testing all combinations of failures and errors, it is a good idea first to test each failure in isolation (e.g., single-fault, rather than multiple-fault) and later to look at some of the most probable failure combinations. Often, in scoping operations monitoring intelligent systems applications, the users will agree that the complexity of dealing with multiple failures results in a diminishing return for the large amount of development effort. If they are outside the scope, then multiple-failure scenarios aren't needed in prototype evaluation.

5.2.3 Specific Variations

Specific variations and less frequent scenarios are usually developed next. Examples of this type of scenario for the remote manipulator subsystem end effector DESSY project would include end effector checkout scenarios. Because there is no payload to grapple, the sequence of events that must be accommodated is slightly different from a normal grapple or release. In addition, a checkout often involves backup systems that probably wouldn't be involved in the common scenarios. Also included in the specific variations scenarios would be some less common types of failures or errors.

5.2.4 Integration Scenarios

Integration scenarios should be employed to ensure that the new application can be integrated into the software environment and the task environment where it will ultimately be used. The software environment integration includes connecting to a data stream, interpreting it correctly, dealing with erratic data, and making certain that real-time processing constraints are met. Task environment integration includes ensuring that information is available to the operator when it is needed and ensuring that the task loading of the operator is not overwhelming. An important problem to avoid is the situation where the new intelligent system is most demanding during a time when the operator is already busy with other tasks.

5.3 Scenario Construction

Once you have decided what the next scenario should be and the major events that should occur, you must then construct the scenario, with specific events and time-stamps. The following description of a three-pass construction process should help you get going:

First, observe the user in performing the task with few (if any) interruptions so that you can write a gross-level description of the scenario. For the generic scenarios, this is as far as the construction process goes. The only thing remaining is to verify the accuracy with the user. For other scenarios, the user might arrange for playback data for those events. However, when narrative descriptions of scenarios are being developed, it might be sufficient for the user to describe what data changes, when, and what he or she must do about it—while sitting in front of the current operational displays.

The purpose of the **next** pass is to add details. This is accomplished by having the user step through the scenario slowly, with interruptions to answer detailed questions. You can supplement this information by reading reference and training manuals.

The **final** pass is to edit the details so that they are correct and realistic. Correctness can be verified by the user and subject matter expert and by referring to manuals. Realism is a little more difficult, but can be approximated by building a simulated data stream, running the prototype, and having the user-designers and subject matter experts evaluate the scenario as they see it unfold. Later, it would be good to compare it to recordings of actual data streams, if they are available. For software systems we have developed, recordings of data streams during Shuttle missions were helpful for ensuring realism.

Sometimes this sequence cannot be followed. For instance, when building a completely new facility, there are no users to be observed performing the task to be supported. In such a case, the three-step process identified above cannot be followed literally. Probably, the best approach in such a case is to build the scenario in an iterative fashion, using requirements and operations concepts documents and consulting with domain experts. The goals of the three steps can still be used in this case: First, try to identify the general events which should occur in the scenario, then add details, and finally, verify the accuracy of those details. It is important to consult with the domain experts during the verification of the details of the scenario.

A scenario will be constructed to guide the design and evaluation of a specific prototype iteration. However, once it has been used to evaluate the prototype, it will continue to be useful in evaluating subsequent iterations of the prototype as well. Consequently, there will be refinement of the scenario based on the evaluation, as well as refinement of the prototype.

6. Design the Prototype

At the heart of our methods is designing the prototype. Designing the prototype is participatory (involving users, subject matter experts, software engineers, and human interaction engineers) and iterative. The iterations involve informal evaluations (discussed in Section 7). Design scenarios are used to focus the team on user task support and provide a common language for people with varying perspectives and expertise.

In the following discussion, we will focus on the design of the human interaction, though it should be kept in mind that there are a number of concurrent design activities required to make the prototype interactive, to make it intelligent, and to make its processing accurate. In this section we discuss levels of prototypes and their relation to levels of evaluation.

Before you can get started, you must decide the medium in which you plan to prototype. A comparison of the following levels of prototypes and the purposes served by each to your goal statement for the next iteration should help you to decide whether to use paper storyboards, computer display storyboard, a special-purpose prototyping language, or a high-performance programming language. As a rule of thumb, initial feedback for a new human interaction design should be obtained from a low-cost implementation, whereas refinements require increasing approximations to the final implementation medium.

Following the principles of spiral development, we have identified different levels of prototypes, corresponding roughly to the maturity of the project:

1. storyboards and design mockups
2. evaluation prototypes
3. operational and integration prototypes

The general progression of these levels is illustrated in the middle portion of Figure 2.

6.1 Storyboards and Design Mockups

Initially, it will probably make sense to build a storyboard of the user interaction—before a functioning, interactive software prototype is attempted. The goal of such a prototype is to verify (and refine) the initial, generic scenario, the initial task analysis, and the general manner in which the new application will support the users' tasks. By starting with a storyboard, you will have invested less effort before getting your first evaluative feedback of the human interaction designs. You are also not dependent on the completion of initial software engineering efforts before obtaining initial feedback on the type of human interaction you are planning.

Depending on the tools at hand, the storyboard may be paper-based, World Wide Web-based, or may be supported by some other tool. A critical factor is that developing the storyboard should not be labor intensive. Consequently, there is typically no automated processing behind a storyboard. Sometimes computer-based storyboards may allow navigation from one screen to the next, depending on the button pressed, but it would not involve intelligent system processing. The storyboard is used to explore the accuracy of the scenario and the design of the human interaction.

The purpose of the storyboard is to elicit feedback from the user (usually the user representative on the design team). Since this is likely an early experience in the project for the user representative, the storyboard should have some initial, orienting description telling how the software is to support the user during the course of the scenario. The initial description should be short (a page or less). It should be followed by a series of frames or "screen snapshots" which show important parts of the interface as the scenario develops. Since the intent is to test human interaction designs without investing much implementation time, the screen snapshots should be hand-drawn. As you gain confidence that the designs are meeting the user's needs, you can invest more into an interactive prototype design, rather than a hand-drawn sketch of how a display might look. Each of the frames should also be supplemented by some text pointing out:

- scenario developments that caused the display to look the way it does
- design decisions that you think support the users' task performance
- design decisions for which you especially want feedback from the user representative
- specific information questions for the user representatives (or any other design team member) that you need in order to improve the human interaction design

The practice of anticipating the interaction with the user representative around the storyboards will help to orient you for similar activities when meeting around subsequent interactive prototypes.

6.2 Evaluation Prototypes

Following the storyboard is a series of evaluation prototypes, whose primary purpose is to elicit an informative evaluation from users to refine the requirements of the new application. Evaluation prototypes will probably constitute the largest number of prototypes for the project, beginning with the first implementation, which offers user interaction, and culminating in an implementation which accomplishes the project goals of supporting specific users' tasks.

At the time in the project when evaluation prototypes are to be developed, you will need to determine what prototyping medium or computer language to use. On one hand, a special prototyping medium is ideal for the evaluation prototypes because they can be modified, sometimes dramatically, without a great deal of effort. This will encourage the exploration of alternative ways of supporting users' tasks, and it will encourage users to provide a frank evaluation (rather than just approving of a design because the developers seem to have worked so hard). Sometimes prototyping languages do not have the speedy performance of a lower-level programming language. So, if your system has some difficult performance constraints, you might be forced to translate the system into another language, later, as it becomes an operational prototype. Translating the system into another language is difficult, error-prone, and consumes development resources. Furthermore, different languages place different restrictions on graphic forms available to implement a design. As a consequence, the developer may prototype graphic forms that cannot be implemented in the final application, or graphical forms that would serve the user even better may be overlooked in the prototype because the prototyping language did not easily accommodate them. Consequently, as early as the initial evaluation prototypes, you will be faced with making a calculated risk in selecting the language in which to write the prototype.

The selection of a development language can be difficult, but perhaps the following discussion will help. If you anticipate a number of false starts in designing the intelligent system and the human interaction, then the special prototyping language should be favored. If real-time processing constraints are severe and the intelligent system and human interaction are relatively straightforward, the low-level language should be favored. However, for some projects, where both intelligent system and software performance requirements are expected to be difficult to achieve, it might actually be better to identify requirements (with evaluation prototypes) using a prototyping language and then to develop operational and integration prototypes using a lower-level language. Other factors to consider include the:

- programming expertise of the human interaction designer
- availability of programming expertise to convert prototype to operational system
- tools available to you at the time of prototype development

Since the purpose of the evaluation prototypes is primarily to elicit feedback from the user representative, you should plan the meeting with the user representative, just as you did with the storyboards:

- provide a scenario context for the user representative to help him or her stay task oriented
- point out design decisions and get feedback about whether they achieved their goals
- be prepared to ask for specific information which you need to improve the prototype designs

6.3 Operational and Integration Prototypes

After the evaluation prototypes are the operational and integration prototypes, in which the new application is nearly fully operational and integrated into the software and task environment where it will be used. Software integration involves connecting to data streams, dealing with noisy data, exchanging information with other processes, and sharing computing and display resources with other software applications. Task integration involves evaluating the new application in the workplace, with all the other concurrent activities, interruptions, etc., which typically occur while the user is performing the tasks supported by the new application.

An operational prototype is a functional implementation that can support the user. Occasionally, it may need to be re-implemented in a more efficiently executing language. At any rate, if the application is a useful one, it will need to continue evolving after the initial deployment. Keeping an easily modified prototype around will enable you to continue the iterative development so that the application's evolution will continue to be useful. For instance, if users want a new capability added, the new designs can be refined in the prototype, so that only the best designs need to appear in the deployed system product.

As with other progressions identified in our method (Figure 2), the progression according to levels of prototypes is not hard and fast. For example some integration prototyping may occur before all the evaluation prototyping is completed. Also, it might make sense to have more than one level of prototype within the same project. For example, when adding a new capability to an evaluation prototype, it might make sense to storyboard that capability to refine the scenario and new interaction requirements before adding more code to the evaluation prototype.

7. Evaluation

During evaluations, users determine if the prototype is providing the right task support in the context of a specific scenario. They report their evaluations either with words (by answering questions or volunteering information) or with actions (by performing the task quickly, without hesitation, and with few errors). The results of the evaluation will indicate where the prototype, scenarios, and task understanding need improvement.

7.1 Formal vs Informal Evaluation

We use three types of evaluation: informal, formal, and outside. Informal evaluations are an integral part of the design process. They take place on a continual basis during the design and development of a prototype iteration. Formal evaluations, on the other hand, are used between prototype iterations. They involve evaluators who were not involved in design, and are used to ensure that the design team hasn't "taken a wrong turn." Occasionally, an outside review can help to supply a perspective that is difficult to achieve from within the design team.

7.1.1 Informal Evaluation

Informal evaluations are performed by development team members, including the user representatives. They are extremely important in producing a usable, useful prototype in a very short amount of development time. In fact, we have referred to the informal evaluations in this Field Guide under the discussions of scenario generation, task analysis, and prototyping. Informal evaluation is most productive in rapid, collaborative design sessions where there are representatives with user task knowledge, subject matter expertise, human factors expertise, and software development expertise. Note that, depending on the individuals involved, two people could represent all four viewpoints. Using this method, the developer leads the user (and subject matter expert) through a scenario and a demonstration of the prototype. When a point of discomfort is encountered, modifications are made. The type of modification depends on the type of difficulty: human factors, task support, or subject matter accuracy. Modifications may be needed for both the scenario and the prototype. Suggestions for the modifications could come from anyone—the value of having representatives from so many viewpoints is that the modification can be given a preliminary review from all viewpoints at once. If an idea is obviously impractical from any of these viewpoints, it won't live long, and not much effort will be invested in it—another idea can be offered in its place. Ideally, changes can be made to the prototype on the spot to see if it works as everyone anticipated. If the suggested change is too extensive to make on the spot, it's probably a good idea to try to sketch out the implications of the change so that everyone is clear about what is intended. This way, the software developer can be more certain that the solution will be about right before investing the time to implement this extensive change for the next group design session. The strength of this type of interactive, collaborative design session is that many development iterations can be explored in a very short time interval.

A potential weakness is that it could degenerate into supporting a single user's personal preferences. This design pitfall has been dubbed "design by Mikey likes it," emphasizing the fact that, even among actual users, personal preferences differ. (Just because Mikey likes that breakfast cereal doesn't mean I will.) The goal of human interaction design is not to cater to a single user's personal preferences, but to support task performance by all users. One way to avoid this pitfall is to consult with multiple users during the initial task description and generic scenario definition and, later, to involve multiple users in the formal evaluation. Another way to avoid this pitfall is to bring a strong task orientation to the collaborative design (evaluation/modification/iteration) session. Directive questions can help provide a task orientation:

- Will this help you to perform your task?
- Can you find all the information you need for the next decision?

Another way to help ensure a continual task focus is to work within the context of scenarios:

- As the microswitch failure begins, does this display provide you with enough information to recognize what is happening?
- Can you distinguish this failure from other possible failures with the information in this display?
- Is there other contextual information that should be provided so that you can recognize what failure is developing?
- Is there more information that should be shown so you can decide what to do about the developing situation?

During an informal evaluation, behavior observation typically amounts to watching for unexpected behaviors, noticing when users seem to have difficulty (provide obviously wrong responses, take longer to respond than expected, appear confused, or appear frustrated). Try to select a moment when you will not interrupt the user's task performance, but before he has forgotten what was causing the difficulty. Then, ask what was going on:

- I expected you to push the blue button, here. What caused you to push the green one? How might the wording be changed (or How might the task be re-arranged) so that you would have pushed the blue button (made the expected response)?
- That last question seemed to be harder to answer than I expected. Did it seem difficult to you? What could we do to the prototype to make that question easier?

After a few of these questions, most users will be giving you all kinds of constructive criticism (and creative ideas) with very little additional prompting.

7.1.2 Formal Evaluation

Whereas informal evaluations tend to have extensive involvement of a single user representative, formal evaluations usually involve many users—users who have not been participating in the design of the prototype. Formal evaluations help to ensure that you aren't tailoring an interface to a single user. They are less frequent than the informal evaluations and are normally done for major prototype iterations. Another distinguishing factor from the informal evaluation is that modifications are not made during the formal evaluation session (no cooperative design sessions). Usually, multiple users will complete the formal evaluation for the same version of the prototype.

Formal evaluations are typically more objective than informal ones. Objectivity is an important characteristic, ensuring that you are reliably measuring usability and usefulness across an entire group of users. Objective measurements include quantitative measurements (e.g., response times or number of correct answers). They also include multiple choice questions and short completion answers that can be tabulated and summarized across users in the evaluation report. Even subjective judgments from users are treated more rigorously in the formal evaluations. For instance, you can provide multiple choices to a preference question (Which capability would be most helpful if added in the next prototype iteration?).

7.1.3 Outside Review

Occasional outside reviews are recommended. They can be helpful for all aspects of design; however, most outside reviewers will not already have an in-depth knowledge of your users' tasks and the best ways to support them. Probably the most efficient use of outside reviewers is to provide them with a description of the users' tasks, what makes those tasks difficult, your strategies for addressing those difficulties, and the prototype and scenarios. By giving the outside reviewers all this information, you will make it easier for them to comment on all aspects of the design and to offer suggestions for improvement and alternatives to consider. Outside reviewers might be consulted early in the project to ensure that you haven't overlooked any obvious strategies for addressing users' tasks needs. Outside reviewers are probably even more valuable in concentrating on the usability of a more mature prototype, performing a heuristic usability analysis. The outside reviewer is especially important at this point because development team members will most probably have lost objectivity for evaluating the usability in the later stages of the project. Also, because of the fresh look at the prototype that outside reviewers can bring, they can help evaluate the consistency with which interactions are handled across the application. Finally, an outside reviewer can often zero in on specific usability problems whereas users may only be able to express that they are having difficulty with specific parts of their tasks.

7.2 Preparing a Formal Evaluation: A Products View

This section is illustrated by Figure 3. Evaluation objectives are based on the prototype objectives, which are, in turn, based on a knowledge of the supported user tasks. Interview questions and evaluation scenario events are based on the evaluation objectives. Scenario events are used to develop evaluation scenarios and to determine what session training will be needed so new users can perform the tasks to support the evaluation. Finally, the interview, evaluation scenarios, and session training are combined into an evaluation agenda so that each evaluation session will be conducted according to a specific plan.

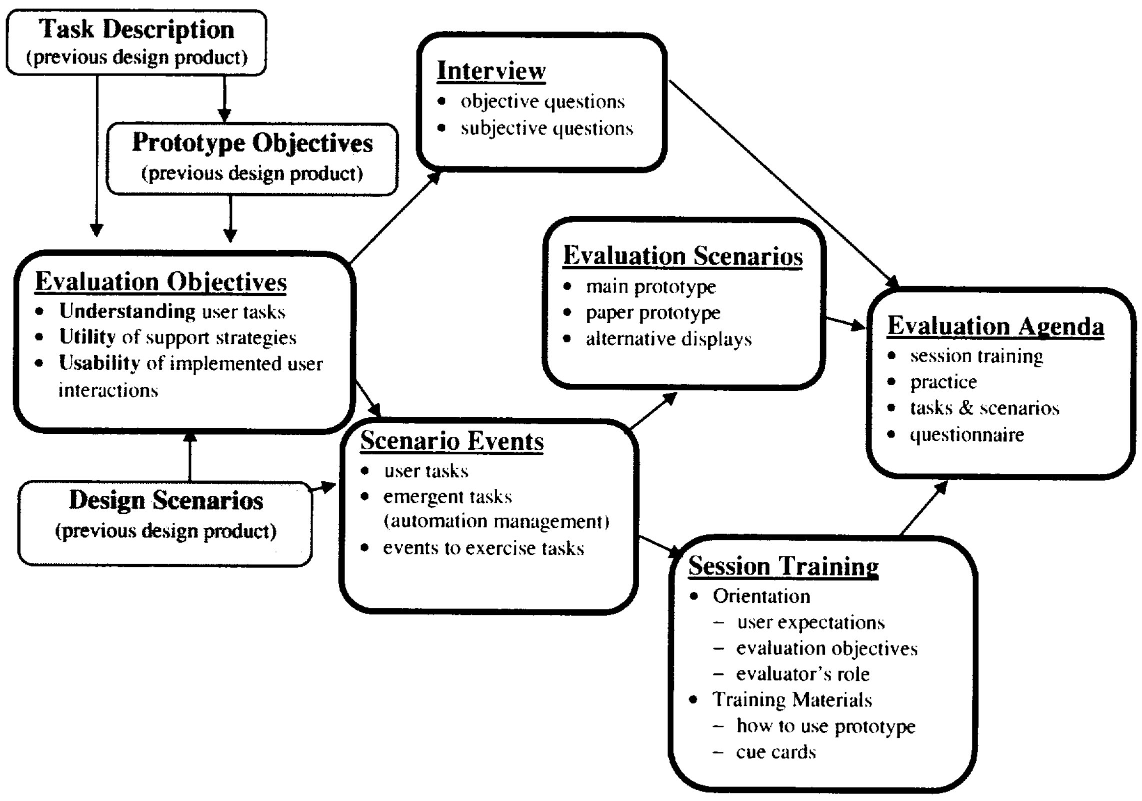

Figure 3. Evaluation Plan Products

The arrows in this diagram are not intended to imply a strict sequence for creating each product, but rather a loose set of dependencies. The need for caution against a sequential interpretation of the arrows is illustrated by the following: In the course of determining the evaluation objectives, designers discover that a prototype objective was overlooked because of an incomplete understanding of the supported user task. The prototype objectives should be modified to fit the improved understanding, as well as all the other products with dependencies on the prototype objectives. Designers need to be alert for this type of opportunity for improving the prototype. The paragraphs below describe each of the evaluation plan products in more detail. Although a strict sequence of actions cannot be stated for developing an evaluation plan, the designer can plan a formal prototype evaluation by attending to each of these products in a somewhat iterative fashion.

Finally, the products described below should probably all appear in written form, but need not be standalone, formal documents. For an example write-up of the supported users' tasks, prototype objectives, and evaluation objectives, see http://tommy.jsc.nasa.gov/~clare/methods/sit_eval/ and http://tommy.jsc.nasa.gov/~clare/methods/artis_eval_plan.rtf.

7.2.1 Previous Design Products

Evaluation is an integral part of design. If the design is attempting to achieve a particular objective, the evaluation should see if that attempt was successful. To ensure unity of purpose between evaluation and the design being evaluated, the evaluation makes use of information and understanding gained earlier in the design process: supported users' tasks, prototype objectives, and design scenarios.

The *task description* identifies those parts of the user's job which are to be made easier by the software you are designing. The relevant parts of the task description should be identified in your evaluation plan for a given formal evaluation.

The *prototype objectives* are the specific objectives to be achieved by the current prototype iteration. There should be a strong relationship between the supported users' tasks and the prototype objectives. However, depending on the maturity of the project, the current prototype may not attempt to address all of the difficulties facing the user in the performance of the supported tasks. The prototype objectives identify which portions of the users' tasks are being addressed by the current prototype.

Finally, the *design scenarios* that have been used to help design the prototype can be instructive in identifying the scenario events that should support the evaluation. In design, these scenarios complete the supported user task definition by supplying specific details and they exercise the prototype in accomplishing its new objectives. Since both of these features are needed in the evaluation, it makes sense to borrow from the design scenarios. However, in the interest of making sure that the prototype doesn't just support one scenario, it is a good idea to borrow events from the design scenarios rather than just copying it wholesale for the evaluation.

7.2.2 Evaluation Objectives

The evaluation objectives provide direction to the evaluation. When conflicts arise, solutions must be based on meeting the important evaluation objectives. They should be understood in relation to the prototype objectives, their role in improving the prototype, whether emergent tasks are supported, and how they add to corporate design knowledge.

7.2.2.1 Evaluation Objectives: Based on the Prototype Objectives

The evaluation objectives are guided by the prototype objectives and by design scenarios which are, in turn, influenced by the prototype objectives. Because the design scenarios should be oriented to address the prototype objectives, evaluation scenario construction should go a little easier if you use events from the design scenarios. (Evaluation scenarios shouldn't be identical to the design scenarios, however, so that you can be sure the prototype isn't oriented to supporting just one case.) Scenario events are identified to address as many of those objectives as possible. When objectives cannot easily be addressed by scenario events, interview questions are constructed to allow the evaluators to address those objectives. This is the relationship of the evaluation objectives to the other evaluation plan products, as illustrated in Figure 3 ("Evaluation Plan Products").

7.2.2.2 Evaluation Objectives: To Improve the Prototype

To identify evaluation objectives, you should look beyond the stated prototype objectives to ask, "What parts of the prototype need improving?" The questions below should suggest additional important objectives to the evaluation planner:

1. Does the human interaction design provide sufficient task support?
2. For those designs that need to be improved, how should they be improved?
3. What new capabilities should be added to the next prototype iteration?
4. What changes need to be made to the scenarios to make them more realistic?
5. What changes need to be made to the scenarios to make them reveal the usefulness and usability of the prototype?
6. What changes should be made to the other design products (project goals, task description, evaluation plans)?

Another way to ensure that needed improvements are found during the evaluation is to consider how the evaluation report will be used. Figure 4 ("How the Evaluation Report Is Used") shows that the evaluation report provides inputs for improving the prototype design, revising requirements for the final implementation, and adding to the corporate design knowledge. The requirements path was added because some of our projects involve handing off a prototype for final implementation by an external development organization—one that specializes in writing and certifying operations software. The value of looking at this wider scope is to ensure that the evaluation covers all the issues that should be addressed in the final report.

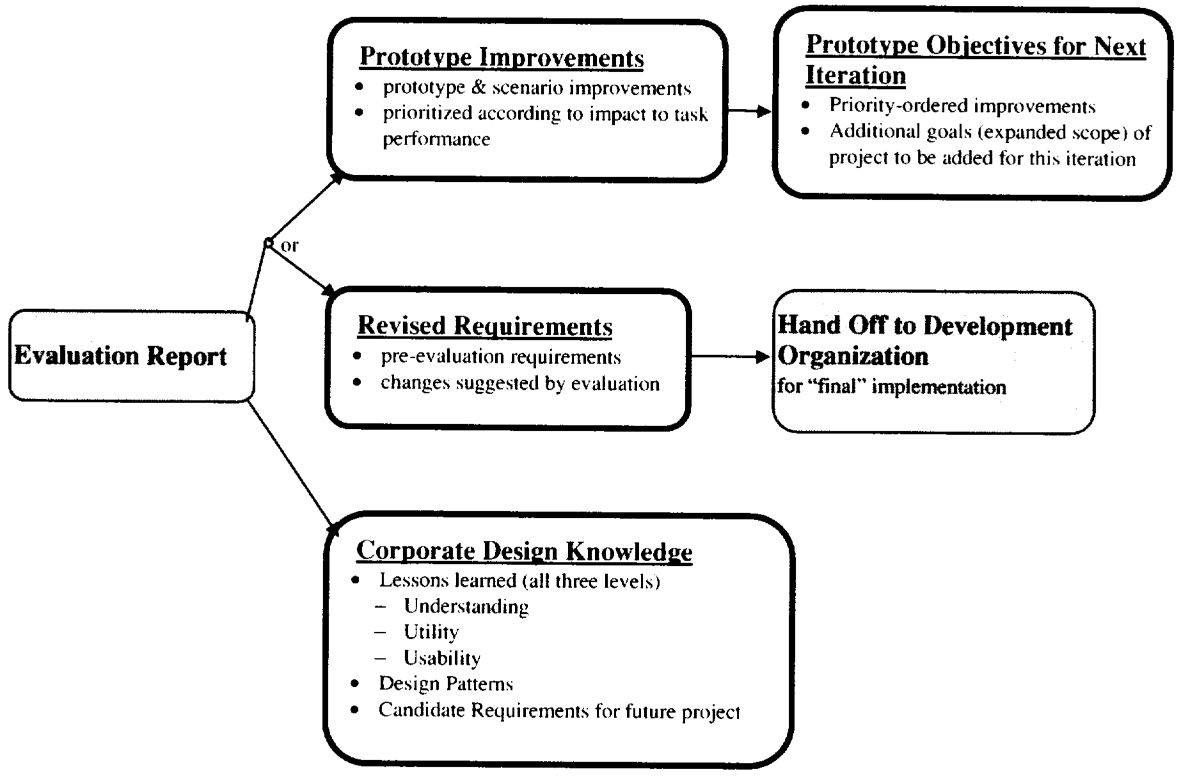
Prototype Improvements
• prototype & scenario improvements
• prioritized according to impact to task performance
Prototype Objectives for Next Iteration
• Priority-ordered improvements
• Additional goals (expanded scope) of project to be added for this iteration
or
Evaluation Report
Revised Requirements
• pre-evaluation requirements
• changes suggested by evaluation
Hand Off to Development Organization
for "final" implementation
Corporate Design Knowledge
• Lessons learned (all three levels)
– Understanding
– Utility
– Usability
• Design Patterns
• Candidate Requirements for future project

Figure 4. How the Evaluation Report Is Used

7.2.2.3 Evaluation Objectives: To Evaluate Support for Emergent Tasks

Another set of evaluation objectives can arise from the fact that innovative automation often causes new user tasks to emerge. For instance, intelligent systems, while providing valuable assistance, need to be managed. It is important to evaluate how easily the intelligent system can be managed. Below are some candidate objectives for you to consider in relation to the emergent task of managing the new intelligent system intelligent system:

1. If the user loses confidence in the intelligent system at this point in the scenario, can he/she stop the intelligent system or safely ignore it?
2. If the user has special evidence that the intelligent system is wrong in a particular instance, can he/she override the intelligent system conclusion (e.g., change the assessment from "operational" to "questionable)?"
3. If, up to this point, the intelligent system has been performing this task alone, can the user take over the task now?
4. Can the user describe what the intelligent system is doing at any given point in its execution?
5. Is adequate feedback provided to the user showing the intelligent system management actions that have been taken? (e.g., Is it obvious when the user has switched to manual mode? Can users see which manual commands have been issued?)

7.2.2.4 Evaluation Objectives: To Add to Corporate Knowledge

Woods et al. (1996) describe the design process as occurring on three levels: understanding, utility, and usability. At the first level, the designer works to understand the users' tasks, what makes them difficult, and how users are managing the difficulty. At the second level, the designer works with strategies for addressing some of the things that make the users' tasks difficult—the application is made useful. Finally, at the third level, the designer works with specific implementations of those strategies—the application is made usable as well as useful. After completing the design of an application, a designer will have gained some knowledge which can be useful in similar future projects concerning these three levels of design: understanding, utility, and usability. While adding to this corporate knowledge probably shouldn't be a primary driver for the evaluation, the knowledge for satisfying these objectives is normally gained during the course of seeking to improve the prototype design.

- **Verifying and refining an understanding** of the users' tasks. The understanding should already be about right because of your previous involvement with the user representative in the early formative stages of building the prototype. The formal evaluation provides an opportunity to verify the understanding you have obtained from the user representative and to refine it by interacting with several additional users who have not been involved in the design process. Beyond addressing prototype improvements, the long-term knowledge identifies the types of tasks which are difficult for users to perform. Candidate objectives to consider:
  - What are the tasks?
  - What makes the tasks difficult?
  - Which ones are done frequently?
  - Which ones are time-consuming?
  - Where do errors usually occur?

   - What strategies, practices, job aids, etc., help to manage the task difficulty? (This is oriented toward things external to the prototype which help task performance.)

- **Verifying and refining usefulness.** The refinements could be more extensive than with understanding the tasks. Again, the strategies for supporting the user should not be too far afield, because of the input from the user representative on the development team. Depending on how many more prototype iterations are planned, new strategies might be explored to see how the larger group of users responds to them. The long-term knowledge to be gained is the types of strategies that are effective in addressing difficulties in user task performance. Candidate objectives to consider:
   - What parts of the prototype really help with task performance? Why?
   - What parts are of little help? Why?
   - What parts are in the way?
   - What additional assistance would be desirable?
   - Was all the relevant information presented when a decision was needed?
   - Can users see at a glance the current state (configuration) and status (health) of the intelligent system and of the monitored system?
   - Can users see what control actions are available? Can they see what actions have been taken?
   - Can users take over part of the automated system's task manually? If so, is it apparent how that can be done? Can it be done without confusing either the automated system or the users?

- **Improving usability.** Depending on the goals of the project, the last iteration should probably be done to refine the usability to a highly finished level. The long-term knowledge is which display and interaction forms are successful for implementing strategies identified above. Candidate objectives:
   - What interaction forms were easy to use?
   - Which display forms were effective? Which were not?

7.2.3 Interview

In our evaluations, we are primarily interested in observing people using the prototype to perform a task in the context of a scenario. We are also very interested in the verbal reports that person can give us telling how well the prototype helped and where it can be improved. The interview questions sometimes help to explain the observations and verbal reports from the more objective portions of the evaluation. In addition, the interview can address evaluation objectives which might not otherwise arise during the execution of a user task:

- What needs to be changed?
- How does it need to be changed?
- What should be added for the next prototyping iteration?

Objective questions (multiple choice) tend to be preferable when trying to combine the results from several evaluators so that the design team will understand how well a particular aspect of the design worked or how much users would like to see a proposed feature in the next iteration. However, subjective (open-ended) questions need to be included as well, to allow users to communicate needs and viewpoints the evaluation planners haven't anticipated.

This collection of questions to be addressed has been called an interview because it is conducted in an informal manner, with a developer asking questions orally and an evaluator answering them orally. However, in some respects, it resembles a questionnaire, because most of the questions are written—many are written with multiple choice response options. They are written so that the developer can be sure that all the objectives received adequate coverage with each evaluator. They are delivered in an interview format so that it will feel like less work to the evaluator and to allow an opening to ask follow-up questions to the evaluator's responses. Sometimes even subtle, non-verbal responses indicate that there is a lot more to learn about a topic if only you'll ask a follow-up question.

Frame questions so that evaluators will respond from their expertise, rather than having them give opinions about matters in which they are not expert. For example,

- GOOD: Present a scenario and two displays, asking evaluators which new coding scheme helps them to locate anomalous data more quickly.
- NOT AS GOOD: Ask what coding scheme (bold print vs. blue text for anomalies) they prefer for a type of information display.
- DIFFERENCE: No task information to orient the evaluators; phrasing of the question implies that personal preference is being sought; less concrete assistance for judging coding schemes.

7.2.4 Scenario Events

Scenario events are the key to constructing evaluation scenarios efficiently without compromising the quality of the evaluation. The design team scenarios are constructed to ensure that the prototype addresses the prototype objectives. Because evaluation scenarios have a very similar function, it would be most efficient just to reuse the scenarios that were originally used to design the prototype. Unfortunately, this practice could result in a prototype which addresses only one case: the specific scenario used in both the design and the evaluation. To make the evaluation a better test of a more general task support, events are extracted from the design scenarios. They can be modified slightly and recombined to form new scenarios, which can then be used in the evaluation. This practice improves efficiency by reusing selected scenario events but also results in a more independent assessment of task support.

7.2.5 Evaluation Scenarios

Scenario events are integrated into connected stories so that the user can evaluate the prototype in a realistic context. The scenarios help the user to stay task-oriented in his/her evaluation, which helps to prevent the evaluation session from deteriorating into some disconnected set of personal preferences. In other words, the scenario helps him/her evaluate how well the prototype supports task performance rather than whether he/she personally prefers blue backgrounds.

Because part of the evaluation is intended to explore directions for future prototypes, some of the scenarios may call for capabilities that have not yet been built into the current prototype. To address these questions, it might be necessary to make paper prototypes or to mock up small, disconnected, exploratory prototypes to elicit better answers from the evaluators about future capabilities. These techniques can also be used to investigate alternative approaches the prototype could have taken.

7.2.6 Session Training

Session training includes an orientation supplemented by training materials. The *orientation* helps evaluators understand why you are building the prototype, and they need to understand their role in evaluating it. Fortunately, much of what you have done to prepare for the evaluation can be used in the introductory materials. For instance, the evaluators should be informed of the high-level evaluation objectives (e.g., "usefulness, rather than usability"). It is important to avoid setting false expectations, which could damage the future credibility of the development team. Tell how the evaluation report will be used in your project (see Figure 4, "How the Evaluation Report Is Used").

- What is the overall goal of the project?
- If this is one drop in a plan of incremental drops,
  - What are the others?
  - When are the others expected?
  - How is the current drop useful in the short run?
  - How will it be useful in the context of future drops?
  - Where do evaluator inputs fit within the plan of incremental drops?
- What are the evaluation goals? Which are primary and which are secondary?
- What will be done with the evaluators' comments, suggestions, and reactions?

Training materials are used to make the evaluators familiar with the prototype so that they can evaluate it. The evaluators you select should be future users of the system. As such, they should already be familiar with the supported tasks and they should easily recognize the scenarios. However, because the prototype is new, they need to be trained to use the prototype to perform the task. How much training they receive depends on the way in which they are expected to use the new software. If you were developing a walk-up-and-use information kiosk, the evaluation probably should not have any training. However, most of our prototypes are for people to use on a frequent basis. Consequently, it is more important that they be able to use the software well after receiving some training than to be able to use it without any training at all. We usually tell them how to perform each task and lead them through an execution of the task using the prototype.

If the task is somewhat complex, we make graphical cue cards illustrating how each task is performed. If cue cards are used, they should also be evaluated. An excessive dependence on the cue card may indicate weakness in the user interface—perhaps a task orientation needs to be integrated more tightly into the interface. If the evaluator indicates the cue card was useful but could be discarded after becoming accustomed to the new software, then it should probably be delivered along with the software—perhaps included as online help, as well.

Finally, part of the user training is to set expectations for the users' role in the evaluation. They aren't being evaluated, the software is (that's why we call them evaluators, not subjects). The evaluators are helping you by providing insights from the user's viewpoint. You are not only concerned with how well the software supports them in performing a task, but you are also interested in how to improve your prototype. Finally, you need to let them know what aspects of the prototype are of greatest interest at this point in time. In other words, they are the evaluators, and you are providing them with a view of the evaluation objectives. Initial prototypes will be concerned more with whether the right tasks are being supported in reasonable ways. Evaluations of later prototypes will be centered more on how effectively that support was delivered.

Consider using the following steps in the session training:

- Explain low-level interaction details.
- Introduce cue card.
- Lead user through task.
- Gain experience with the basics of the human-computer interaction.

7.2.7 Evaluation Agenda

The agenda identifies everything that should happen during an evaluation session and the approximate amount of time to be spent on each. Table 2 identifies a typical agenda. The order is important and has been rather constant across our previous evaluations. The specific times will vary, depending on the evaluation—often the times will vary with each evaluator.

Table 2. Typical Times for an Evaluation Agenda

15 min	introduction & project description
15 min	session training & practice
45 min	task execution with prototype (2 or 3 scenarios)
15 min	interview and debriefing
30 min	take notes & prepare for next evaluator
2 hours	total (schedule evaluators 2 hours apart)

This is an example of a high-level agenda. You'll want to add more details before the evaluation. The web page at http://tommy.jsc.nasa.gov/~clare/methods/ contains links to some examples of evaluation agendas for specific projects.

7.3 Conducting a Formal Evaluation

The most important results of the evaluation arise from observing the evaluator interacting with the prototype to perform a task in the context of a scenario. If the observations are done well, they can be quite valuable. For instance, if you notice unexpected behaviors (slowness, wrong answers, mistakes, signs of confusion), you can ask the evaluator what was going on. Was the prototype doing something different from what the evaluator expected? Was the prototype providing inappropriate information? Was a message worded badly? Was the meaning of graphic form unclear?

No matter how carefully the evaluation is planned, it is unlikely you'll have it right the first time. If so, you probably spent too much time planning it. Consequently, we recommend doing "dry runs" for development team members who have been less active in planning the evaluation. If the user representative fits that description, he/she will be able to provide valuable feedback for improving the evaluation sessions.

The following list is a compilation of advice (lessons learned) for guiding your evaluators through a session:

- **Keep evaluation objectives in mind.** Sometimes deviations from the plan are required in order to ensure that the main objectives are met.
- **Try not to explain too much.** Part of the purpose of the evaluation is to see how clearly the prototype communicates with the user.
- **Try not to defend the application as implemented**. You are there to learn how to improve it. Defensiveness on your part can shut off the flow of helpful information. Your dedication to improvement should always show—this is important in gaining the evaluator's trust that you will act on the results of the evaluation, that the evaluator's efforts will be used to a good effect.
- **Correct misunderstandings when they threaten the value of the evaluator's responses.** Even if you are testing a walk-up-and-use system, if the initial instructions are bad, you should step in and correct them so that additional features of the prototype can then be evaluated. Be sure to note any corrections that were necessary—if you had to interrupt to explain, something about the prototype probably needs improving.
- **Temper the evaluation objectives with a concern for the evaluator.**
  - Don't stress the evaluator.
  - Don't allow the evaluator to leave with ill feelings toward the project.
  - Don't allow the evaluator to leave with misconceptions of the project.

7.4 Reporting Results of a Formal Evaluation

Once the evaluation is complete, the development team needs to determine the design implications and prioritize them for future development iterations. For the most part, the report should answer each of the evaluation objectives:

- How well did the prototype perform with respect to each evaluation objective?
- What parts of the prototype need improving?

- What did the evaluators see as important new capabilities to add to future prototypes?
- What additional lessons were learned about the supported user tasks?
- What additional lessons were learned about strategies for supporting those tasks?
- What additional lessons were learned about the specific implementations of those strategies which were used in the prototype?

The improvements and new features should be prioritized so the design team can make a reasonable decision about when to implement them.

Before writing the report, review Figure 4 ("How the Evaluation Report Is Used").

8. Further Reading

The reader may wish to gain more in-depth knowledge about some of the information discussed in this "Field Guide." The following list is an attempt to identify some of the more important sources for this purpose. The references have been categorized to make it easier to find specific information.

8.1 Related Documents

A related set of documents has been written describing other aspects of our approach. These documents emphasize the role of an intelligent system as a team player:

Land, S. A., Malin, J. T., Thronesbery, C., and Schreckenghost, D. L. (1995, July). *Making Intelligent Systems Team Players: A Guide to Developing Intelligent Monitoring Systems.* NASA Technical Memorandum 104807. Available from Springfield, VA: NTIS.

Malin, J., Schreckenghost, D., Woods, D., Potter, S., Johannesen, L., Holloway, M., and Forbus, K. (1991). *Making Intelligent Systems Team Players: Case Studies and Design Issues. Volume 1. Human-Computer Interaction Design, Volume 2. Fault Management System Cases.* NASA Technical Memorandum 104738. Available from Springfield, VA: NTIS.

Malin, J. and Schreckenghost, D. (1992). *Making Intelligent Systems Team Players: Overview for Designers.* NASA Technical Memorandum 104751. Available from Springfield, VA: NTIS.

Malin, J., Schreckenghost, D., and Rhoads, R. (1993). *Making Intelligent Systems Team Players: Additional Case Studies.* NASA Technical Memorandum 104736. Available from Springfield, VA: NTIS.

Roth, E., Malin, J., and Schreckenghost, D. (1997). Paradigms for intelligent interface design. In T. Helander, P. Landauer & P. Prabhu (Ed.), *Handbook of Human-Computer Interaction* (2nd ed). Amsterdam: Elsevier, pp. 1177-1201.

8.2 Iterative, Collaborative Design

Boehm, B. (1988, May). "The Spiral Model of Software Development and Enhancement," *IEEE Computer, 21*(5), pp.61-72.

Malin, J. T., Thronesbery, C.G., and Land, S. A. (1993). "End-Effector Monitoring System: An Illustrated Case of Operational Prototyping," *Proceedings of the Seventh Annual Workshop on Space Operations and Applications Research (SOAR'93)* (NASA Conference Publication). Houston, TX: NASA-Johnson Space Center, pp. 461-467.

Thronesbery, C.G., and Malin, J. T. (1993). "Operational Prototyping: A Development Approach for Improved Support of User Task Performance." *Proceedings of the MITRE User Interface Symposium.* McLean, VA: The MITRE Corporation.

Thronesbery, C.G., and Malin, J. T. (1993)."Operational Prototyping: A Development Approach for Innovative Technology Applications," Paper presented at the *Joint Applications in Instrumentation, Process, and Computer Control (JAIPCC '93) Symposium.* Houston, TX: University of Houston-Clear Lake.

Woods, D.D., Patterson, E.S., Corban, J.M., and Watts, J.C. (1996). Bridging the Gap Between User-Centered Intentions and Actual Design Practice. *Proceedings of the Human Factors and Ergonomics Society 40th Annual Meeting.* Santa Monica, CA: Human Factors and Ergonomics Society.

8.3 Use of Scenarios and Task Descriptions

Jeffries, R. (1997). The Role of Task Analysis in the Design of Software. In M. Helander, T.K. Landauer, & P. Prabhu (eds). Handbook of Human-Computer Interaction (2nd ed). Amsterdam: Elsevier, pp. 347-359.

8.4 Evaluation

Wixon, D., and Wilson, C. (1997). The Usability Engineering Framework for Product Design and Evaluation. In M. Helander, T.K. Landauer, & P. Prabhu (eds). *Handbook of Human-Computer Interaction* (2nd ed). Amsterdam: Elsevier, pp. 653-688.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE July 1998	3. REPORT TYPE AND DATES COVERED NASA Technical Memorandum

4. TITLE AND SUBTITLE Field Guide for Designing Human Interaction With Intelligent Systems	5. FUNDING NUMBERS
6. AUTHOR(S) Carroll G. Thronesbery* & Jane T. Malin	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lyndon B. Johnson Space Center Houston, Texas 77058	8. PERFORMING ORGANIZATION REPORT NUMBERS S-841
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER TM-1998-208470
11. SUPPLEMENTARY NOTES * Metrica	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/unlimited Available from the NASA Center for AeroSpace Information (CASI) 7121 Standard Hanover, MD 21076-1320 (301) 621-0390 Subject category: 66	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The characteristics of this Field Guide approach address the problems of designing innovative software to support user tasks. The requirements for novel software are difficult to specify a priori, because there is not sufficient understanding of how the users' tasks should be supported, and there are not obvious pre-existing design solutions. When the design team is in unfamiliar territory, care must be taken to avoid rushing into detailed design, requirements specification, or implementation of the wrong product. The challenge is to get the right design and requirements in an efficient, cost-effective manner. This document's purpose is to describe the methods we are using to design human interactions with intelligent systems which support Space Shuttle flight controllers in the Mission Control Center at NASA/Johnson Space Center. Although these software systems usually have some intelligent features, the design challenges arise primarily from the innovation needed in the software design. While these methods are tailored to our specific context, they should be extensible, and helpful to designers of human interaction with other types of automated systems. We review the unique features of this context so that you can determine how to apply these methods to your project Throughout this Field Guide, goals of the design methods are discussed. This should help designers understand how a specific method might need to be adapted to the project at hand.

14. SUBJECT TERMS man-computer interface; computer programs; software engineering; software tools; computer systems design;	15. NUMBER OF PAGES 46	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited