USER-CENTERED DESIGN OF SPACECRAFT GROUND DATA SYSTEMS AT NASA'S GODDARD SPACE FLIGHT CENTER

Jeffrey Fox
Pacific Northwest National Laboratory*
901 D Street SW, Suite 900
Washington, DC 20024
jeff.fox@pnl.gov

Julie Breed
Karen Moe
Robin Pfister
Walter Truszkowski
Dana L. Uehling
NASA/Goddard Space Flight Center
Greenbelt Road
Greenbelt, MD 20771
julie.breed@gsfc.nasa.gov
karen.moe@gsfc.nasa.gov
pfister@killians.gsfc.nasa.gov
truszkow@pop500.gsfc.nasa.gov
dana.uehling@gsfc.nasa.gov

Adriane Donkers
User Works, Inc.
1738 Elton Road, Suite 138
Silver Spring, Maryland 20903
adonkers@userworks.com

Elizabeth Murphy
Bureau of the Census
Statistical Research Division
Washington, DC 20233
elizabeth.d.murphy@ccmail.census.gov

Abstract

Spacecraft ground control data systems are by necessity sophisticated and complex. One of the challenges for system designers is to understand and incorporate the needs of the users so that their systems are easy to use and minimize user errors. User-centered design (UCD) can help achieve these goals. UCD techniques can be applied to the design of ground systems to improve quality and functionality so that the users can perform their work more effectively and efficiently. This paper provides a high-level survey of UCD efforts at NASA's Goddard Space Flight Center (GSFC). The paper describes some of the techniques that have been used and the benefits they have produced via case studies. The goal of the paper is to share lessons learned from performing user-centered design at NASA/GSFC with the wider ground systems community and to provide pointers for interested readers to find more detailed information.

Key words: User-Centered Design, Prototyping, Cognitive Modeling, Usability.

Introduction

The ultimate impact of a system, no matter how innovative its design or capabilities, depends on how well its users interact with it. Often, as systems become

*Pacific Northwest National Laboratory is operated for the U.S. Department of Energy by Battelle under Contract DE-AC06-76RLO 1830.

more complex, they become harder to use. An example of this is mission operations software that requires Spacecraft Control Team (SCT) members to monitor thousands of parameters distributed across multiple screens.

Modern ground control software systems employ graphical user interfaces, expert systems, software agents, and data visualization in an effort to alleviate these problems. However, each of these approaches also introduces its own new demands on the users. In addition, new operational models that reduce the number of staff or even eliminate full-time operations (i.e., "lights-out" operations) provide new challenges and can significantly increase the workload for the remaining limited on-call staff.

In his seminal book *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Shneiderman[1] notes that the degree to which a system meets the needs of its users can be measured via attributes such as:

- Time to learn
- Speed of performance
- Rate of errors by users
- Retention over time
- Subjective satisfaction

To optimize these attributes of a system, the design of the system, as well as the process of designing the system,[2] must focus on the users and their needs. This is known as user-centered design (UCD).

The goal of UCD is to ensure that systems are designed with consideration of their users' capabilities and limitations. UCD considers the software, hardware, users' tasks, environment, and all their interactions. Properly employing UCD will increase the usability of those systems, making them easier to use, increasing user satisfaction, and reducing error rates. UCD concepts come from many fields, but most commonly from human factors engineering. In addition, UCD has an ever-increasing impact on overall systems design as the percentage of the software code that is dedicated to the user-system interface increases.

Various UCD techniques and activities are used at NASA/GSFC to design high quality mission operations software and advanced control center environments. Table 1 lists a sampling of some of those UCD techniques. Table 2 lists some projects that have benefited from the application of those techniques.

Which techniques are used and when they are used depends on the type and size of the project (e.g., style guides are most useful for large development teams to control consistency, while expert evaluations are equally effective for any size project). Likewise, the positive effects of some techniques are maximized when they are used at certain phases in a project's lifecycle (e.g., task analyses are most useful when performed before any new software or hardware is developed). Similarly, the type of project (R&D vs. deployment) influences technique selection.

Table 1: UCD Techniques Used at NASA/GSFC

- User design working (focus) groups
- Cognitive modeling
- Education of software designers
- Expert evaluations
- Rapid software prototyping
- Scenario-based design
- Task analysis
- Usability testing
- User interface guidelines, style guides and standards
- Workstation and control room design

Table 2: Projects employing UCD

- EOSDIS
- Data Distribution Facility
- Desktop Satellite Data Processor
- Hubble Space Telescope
- Spacecraft Emergency Response System (SERS) for SMEX and MIDEX missions

The remainder of this paper describes several of the UCD techniques that have been successfully used at NASA/GSFC in the design of software. Unfortunately, there is only space to discuss a few of these techniques: cognitive modeling, software rapid prototyping, and usability testing. Along with the descriptions are examples of how these techniques were used on actual projects and the positive results of using these techniques. The references cited provide opportunities for further learning about UCD.

As you read through these different techniques, you will note common themes and goals. You should also note that these techniques are complementary and, when used together, produce an even more usable design.

Cognitive Modeling

An important goal of user-centered design is to understand the users and how they interact with the systems that they use. Using both syntactic knowledge (knowledge of procedures, such as pressing the "delete" key to erase a character) and semantic knowledge (knowledge of a domain, such as a theory) gained by

experience with a system, users build a mental model of how that system operates. McGraw[3] defines a mental model as an organized set of domain concepts, their relationships and functionality. In computer-based systems, a user's model of how to perform a task is directly influenced by how she perceives those data being presented by the computer. For example, let's say that in order to successfully accomplish a task, a particular switch on a spacecraft must be open. The spacecraft controller must not only know that the switch is to be open, but also be able to interpret the current state of the switch as displayed on the computer's screen (perhaps displayed graphically as a circuit diagram or simply as a "yes/no" field). Also she must know how to tell the computer to send the command to open that switch (perhaps via a typed command or by clicking on a button).

The task flows more smoothly when the data displayed and the method of interaction match the users' expectations of how the system will operate, that is, their mental models. Thus, it is important that the designer incorporate the user's model of the task into the design of the computer's interface so that there is no disconnect between what the operator thinks she sees and does, and what the computer actually is presenting and executing. This problem is compounded by the fact that, to a certain degree, all users possess somewhat different mental models.

In contrast with mental models, a cognitive model is a researcher's attempt to represent conscious and subconscious mental processes and outcomes. A cognitive model is a theory-based, graphical representation of inferred relationships between hypothesized components of human thought. These overt models of the mind may be qualitative or quantitative, specific or general, and they are typically embedded in graphical models of human performance (e.g., Wickens[4]). Most cognitive models encompass the basic constructs of human information processing: sensory processing, perception, short- and long-term memory, and analytic processes such as problem solving and decision making. Different models vary in their level of detail and, sometimes, in the theory that underlies their depiction of the flow of information and control within the model. For detailed treatments of cognitive models in the psychological and human factors literature, see Best,[5] Eberts,[6] Newell,[7] and Solso.[8]

Cognitive models aid analysts in getting beyond the behavioral aspects of human activities to their underlying goals, objectives, and thought processes. Cognitive modeling provides a theory-based framework for cognitive task analysis; which documents operational decision points, information requirements, and the analytic processes followed in making decisions on the basis of available information, incomplete and uncertain though that information may be. Cognitive task analysis provides critical input to user-interface design and evaluation. Such models serve as frameworks for investigating and documenting the role of human information processing in operational settings. They help in the process of understanding the problem-solving and decision-making activities performed in current spacecraft ground control environments. Sources on cognitive task analysis include Redding[9]; Roth and Woods,[10] Roth, Woods, and Pople,[11] and Schlager, Means, and Roth.[12]

In a specific operational environment, early and continuing cognitive modeling and cognitive task analysis can help to identify analysts' information requirements and can support the design of automated job aids. When a job or position is being defined or redefined, it is worth the effort to document cognitive tasks and information requirements as a basis for the design of user interfaces, position manuals, and training programs. When a change is contemplated for a computer system, design of user interfaces and operational procedures that are based on a thorough cognitive task analysis is likely to yield a more usable and operationally suitable product than will result from design that gives low priority to human information-processing issues. A design's support for cognitive capabilities and compensation for cognitive limitations should be evaluated throughout the project's lifecycle.

Example

At NASA/GSFC, cognitive modeling has provided a basis for developing an automated approach to predicting the demands imposed on operators' cognitive resources by various aspects of user interfaces. Modeling the cognitive demands imposed by user interfaces led to development of a series of Computer-Human Interaction Models (CHIMES) prototypes. Early CHIMES prototypes demonstrated the feasibility of assessing the cognitive demands imposed by text-based user interfaces. The underlying model included a functional hierarchy of operational activities, from subtask to mission, and a set of cognitive attributes associated with each level of the functional hierarchy.[13] A demand analysis using CHIMES could be performed early and iteratively throughout the design and implementation of a textual user interface. CHIMES evolved into a tool for checking a user-interface design both for internal consistency and for compatibility with design rules.[14]

The early CHIMES methodology and toolset were applied in an analysis of the user interface to the Planning and Resource Reasoning (PARR) tool, a

scheduling aid developed by the Bendix Field Engineering Corporation.[15] The purpose of the analysis was to evaluate the PARR user interface from a human factors perspective in order to improve its ability to support efficient, effective scheduling of spacecraft events by the Earth Radiation Budget Satellite (ERBS) scheduler. Analysts observed and interviewed the ERBS scheduling operator about her functions and tasks and examined the various paper-based job aids that she consulted in building the weekly schedule. The scheduler's job was modeled as it was currently being done, in a largely manual mode, and as it could be done with the aid of the PARR tool. The analysis permitted a comparison between the largely manual job and the aided job, showing that task demands on the ERBS scheduler were significantly lower after the introduction of the automated scheduling aid. The analysis identified several PARR displays that could benefit from some redesign.

In the current spacecraft ground control environment, with its emphasis on increased automation to reduce mission costs, cognitive modeling and cognitive task analysis can provide inputs to decisions on system design. It cannot be assumed that partially or fully automating a function will eliminate the need for human involvement in mission operations. For the success of lights-out automation, it is essential to define the cognitive tasks that operators and analysts currently perform, many of which are not fully documented in existing procedural manuals. If fault detection and resolution are to be automated, cognitive models of humans performing those tasks can provide valuable guidance. Cognitive issues in lights-out automation are under investigation from several conceptual and empirical perspectives (e.g., Mitchell, Thurman, & Brann,[16] Murphy & Norman,[17] Truszkowski[18]). These investigations are likely to lead to the development of cognitive models of the combined cognitive system, that is, the cooperative activity of intelligent software agents and human analysts, where mutual understanding will be critical to success.

Software Prototyping

Traditional software development is a very formal and structured process. This process is generally represented by the "waterfall" model that starts with requirements analysis and is followed sequentially by design, development, testing, operations, and finally maintenance. For this approach to be successful, all of the detailed requirements must be known and documented prior to the onset of design. Any missed requirements or functionality tend not to be discovered until late in the product's life cycle during testing or even after deployment. Changes at these points tend to be very time-consuming and costly.

Software rapid prototyping can help alleviate this problem. A rapid prototype is used to simulate a system's functionality and user interface and can serve as a model for the technical demonstration of a system. UCD is specifically concerned about understanding the needs of the end user in order to get a better understanding of requirements early on in the project. The most common reasons for software prototyping are to gain a better understanding of the users' requirements and to allow the developer to confirm that a specific approach will accomplish the needed functions with adequate system performance. Generally, prototyping is iterative and leads to a more cyclic design approach.

Prototyping generally takes one of two forms: "throwaway" or "evolving."[19] A throwaway prototype is usually discarded once it has served its purpose. For evolving prototypes, the functionality is added and improvements are made until the prototype becomes the operational system.

Alternatively, Pressman[20] classifies prototypes into three types - a model, a working prototype, and an existing program. The model, either paper- or computer-based, is a mock-up that depicts the user interface in order to convey a sense of look, feel and functionality of the system. The working prototype is an implementation of specific functions of the system. In the case that a system is already in place and meets all or most of the needed functionality, it can serve as the basis for prototyping additional functions or improving on existing functions.

According to Pressman, an important factor in rapid prototyping is working with the user to define and agree on the scope, purpose, and nature of the prototype - particularly whether it is to be discarded or turned into an operational system.

Boar[21] describes factors, such as the characteristics of the project and the user as well as the application area and the complexity, that can be assessed to determine whether a software project can benefit from prototyping. Once it is decided that a prototype is needed, the form of prototype should be considered.

Example:

The overall purposes of prototyping in the Earth Observing System Data and Information System (EOSDIS) are:

- To address risks to development
- To enable technology transfer into the system and out to the global change research community
- To reduce long term costs through evolution.

The specific goals are to better understand the needs of the user community and to mitigate identified risks. For EOSDIS, multiple approaches are used. Most prototypes investigate system functionality that will later be infused into the EOSDIS Core System (ECS). All of the prototypes start out as intended throwaways, but some that show promise for infusion are evolved to the point that they can be integrated into the system.

One prototype, known as EOSDIS Version 0, began in the early requirements definition phase of ECS to show proof-of-concept for interoperability among distributed systems and to better define interoperability requirements to be levied on the ECS. After a successful demonstration in 1992, this prototype evolved in nature and purpose as functionality was added. It started as a prototype to define requirements and later evolved into a prototype that served as a demonstration of technical capabilities, and then to an operational system. Every six months during its evolution, an on-line, pseudo-operational version of a complete end-to-end system was evaluated by user community representatives. These representatives stated their likes and dislikes, gave recommendations on future improvements, and as a group they agreed on the priorities of functions to be added in the next phase.

While Version 0 evolved, the ECS contract was defined and an end-item deliverable contract was awarded. The development was to follow a formal systems engineering lifecycle process. However a few components that involved end-user interfaces were put on an incremental track with the intention of getting early user feedback on "uncertain driving requirements." The Incremental Track Design process involved various mechanisms to elicit user feedback on prototypes. The key mechanisms included Evaluation Packages, Prototype Workshops, a Client Design Working Group, and Client Workshops.

Evaluation Packages (EP) were a delivery and evaluation mechanism for incremental and other prototype developments. The goal was to maximize visibility for end users, data archive staff, and NASA to get feedback and acceptance. The key challenge of this process was to provide just the necessary amount of structure to enable an evaluation without creating an administration overload.

Prototype Workshops (PW) were typically held after each EP so that each PW included one EP plus other selected prototypes that appeared particularly promising. The PW was held to allow “tirekickers,” data archive representatives, and NASA to review all of these latest tools. Surveys were conducted during the PW to evaluate each of the presented prototypes.

Because there were so many functions for which the designers needed more insight, and the fact that EPs and PWs took time and resources to pull together, a Client Design Working Group (CDWG) was formed to generate scenarios, workflows and models (paper and computer mock-ups) for the remaining client functionality.

Some prototyping efforts were more successful than others. The Version 0 system was very successful and is still operational today. The CDWG also was very successful and ensured buy-in of the end-user community. Incremental Track prototyping succeeded in identifying broad issues in the overall system architecture. It highlighted difficulties in installing and maintaining distributed systems. It also helped the project identify the threshold of the tolerance of the end user's burden to gain access to the system. Currently, the Version 0 system is being augmented to support ECS functionality and will serve as the end-user interface for EOSDIS.

Usability Testing

Usability testing is one method of evaluating how well a system meets its users’ needs. Usability tests can be performed at any stage of a product’s lifecycle, from the paper prototype stage to operations. The main characteristic of a usability test is that the proposed user of the system is observed while performing a series of tasks with a fully functional system or with either a software or paper prototype. It is not a demo to the user, but a chance for the user to work with the system “hands-on.”

The tasks usually include those that will be frequently performed or that are most critical. While the user performs these tasks, many types of data are collected. Some of the quantitative measures include:

- Time to complete a task
- Number of errors
- Number of times help is required
- Time to locate specific information

Some of the qualitative measures include:

- Observed frustration
- Participants' comments
- Ratings
- Severity of error

These data are then analyzed to determine where the system is making it difficult for the user to complete the tasks. Suggestions for changes to the system are then presented to the project development team. Alternatively, developers can act as observers of the

actual tests to get more direct user feedback and gain insight into the usability testing process.

Usability testing is one component of usability engineering,[22] which takes a full life cycle approach to managing the user interface. Usability testing can take many forms. Ideally, usability testing should occur at regular intervals or at major milestones in a project's life cycle, especially for large, complex systems. This is known as formative usability testing.[2] Alternatively, a system may only be tested as part of a final deliverable. This is what happens for many systems due to funding constraints or the late inclusion of a usability engineer on the project team. This is known as summative testing.[2] The disadvantage of this latter approach is that the results of the testing often arrive too late to be incorporated into a revised design.

Usability testing can be done in a formal lab, with a one-way mirror, video recorders, audio recorders, screen-capture software, and an intercom system; however, a "discount" [22-24] approach to usability testing typically is used at NASA/GSFC. In the discount approach, the tests are often run in the users' actual offices where they complete tasks on their own computers or with paper prototypes. The observer(s) sits beside the user and records data either on paper or on another computer. Even with this "low tech" approach, designers gain valuable information that allows them to greatly improve their applications.

Example

Usability testing for the Hubble Space Telescope (HST) exemplifies a discount approach. In this project, the control center software is currently being modernized. Throughout the development cycle of the windowing graphical user interface, various usability tests were conducted. In one of the first tests, three users sat at a computer and completed a series of tasks while the observer took notes. The tasks included (1) creating a new real-time page (real-time pages are graphical pages that can contain hundreds of updating data points, referred to as mnemonics, that are used to monitor the HST's health and safety), (2) adding components (e.g., alphanumeric fields, stripcharts, labels) to a page, (3) modifying characteristics of the components, and (4) saving the page. After observing the users, the observers identified several problems that needed to be addressed. The problems were not critical, but the design could be greatly improved if they were fixed. Examples of some of the problems included:

- Inconsistencies (e.g., changing the color of mnemonics was done differently in two different dialogs)
- Lack of prompts to guide the user (e.g., no confirmation message when logging in, no indication of how to enter a search string)
- Lack of use of standardized user interface conventions (e.g., use of gray to indicate those menu items not currently availability)
- Dispersion of related functions (e.g., information that was related and worked together was separated into two windows)
- Grouping of unrelated functions (e.g., editing of two components was done in the same window and it was thought the options worked together when they were, in fact, independent)
- Missing features (e.g., ability to modify several components at one time, ability to zoom to see greater detail).

The feedback that was gathered from the users and the suggested design modifications were reviewed with the developers. In some cases, the suggested changes were accepted and could be easily implemented. In other cases, further information from the users was requested. Some concepts were modeled using a graphics package, and together with the developers the design was modified. Many of the problems in this initial study have been addressed; and, in follow-up meetings, users reported that the design was much improved.

Another example illustrates usability evaluations conducted with paper prototypes. Here the users were shown a paper mock-up of the tool and were asked to verbally describe how they would use it. This technique of capturing the user's thoughts is called protocol analysis.[25] One study evaluated a tool intended to warn flight operation controllers of anomalous conditions via the display of mnemonics. The results showed that users could easily understand the tool. However, it was found that a particular action of the tool was not anticipated and did not match the user's mental model of the system.

More specifically, the tool represents an anomalous condition by displaying a red light for the sub-system that contains the out-of-limits mnemonic(s). The system then allows the user to click on the light to display a second level indicator that shows which sub-sub-system contains the mnemonic(s). The users liked the concept of seeing the warning light and then clicking on it to get additional information pertaining to the anomaly. However, they had other ideas for what additional information should be presented.

Instead of seeing the sub-sub-system, the users preferred to get further detail on the mnemonic (e.g., current value, and highest and lowest values). Once they had more detail on the mnemonic they would then like to have additional options for further analysis, such as the ability to plot the most recent data or examine the

limits of the mnemonic. This simple evaluation with a paper mock-up only required about 15 minutes of each of the operators' (5 total) very valuable time. Yet, in this short period of time, a problem with a critical component of the software was identified and a solution recommended. Early identification saved the developer considerable time in code development and thus provided a large return on investment.

Conclusion

This paper has provided just a glimpse at user-centered design and its positive effects on the design of ground data systems at NASA/GSFC. You are encouraged to see the large set of papers and books that provide more details on the UCD techniques referenced in this paper, as well as books on other techniques listed in the introduction[26-29] and some more general UCD references.[30, 31] Also, you may wish to visit the Web sites of some of the larger professional societies focussing on UCD:

- The Human Factors and Ergonomics Society (HFES), http://hfes.org/
- The Association for Computing Machinery Special Interest Group on Computer-Human Interaction (ACM SIGCHI), http://www.acm.org/sigchi/.
- The Usability Professionals' Association (UPA), http://www.UPAssoc.org/.

You are invited to contact the authors via email to learn more about user-centered design or to discuss the material presented in this paper.

References

[1]Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Reading, MA: Addison-Wesley, 1986.

[2]Hix, D. and Hartson, H. R. *Developing User Interfaces: Ensuring Usability Through Product & Process*. New York: Wiley, 1993.

[3]McGraw, K. L. *Designing and Evaluating User Interfaces for Knowledge-Based Systems*. New York: Ellis Horwood, 1993.

[4]Wickens, C. D. *Engineering Psychology and Human Performance* (2nd ed.). New York: HarperCollins, 1992.

[5]Best, J. B. *Cognitive Psychology* (3rd ed.). New York: West Publishing, 1992.

[6]Eberts, R. Cognitive modeling. In G. Salvendy (Ed.), *Handbook of Human Factors and Ergonomics* (2nd ed., pp. 1328-1374). New York: Wiley, 1997.

[7]Newell, A. *Unified Theories of Cognition.* Cambridge, MA: Harvard University Press, 1990.

[8]Solso, R. L. *Cognitive Psychology* (3rd ed.). Boston: Allyn and Bacon, 1991.

[9]Redding, R. E. Perspectives on cognitive task analysis: The state of the art. *Proceedings of the Human Factors Society 33rd Annual Meeting* (pp. 1348-1352). Santa Monica, CA: Human Factors Society, 1989.

[10]Roth, E. M. and Woods, D. D. Analyzing the cognitive demands of problem-solving environments: An approach to cognitive task analysis. *Proceedings of the Human Factors Society 34th Annual Meeting* (pp. 1314-1317). Santa Monica, CA: Human Factors Society, 1990.

[11]Roth, E. M.; Woods, D. D.; and Pople, H. E. Jr. Cognitive simulation as a tool for cognitive task analysis. *Ergonomics, 35*, pp. 1163-1198, 1992.

[12]Schlager, M. S.; Means, B.; and Roth, C. Cognitive task analysis for the real(-time) world. *Proceedings of the Human Factors Society 34th Annual Meeting* (pp. 1309-1313). Santa Monica, CA: Human Factors Society, 1990.

[13]Sheppard, S. B. and Murphy, E. D. *CHIMES: Computer-Human Interaction Models* (Report prepared for NASA-Goddard under Contract No. 018750, Task 29-116). McLean, VA: CTA INCORPORATED, 1988.

[14]Jiang, J.; Murphy, E. D.; Carter, L. E.; Bailin, S. C.; and Truszkowski, W. Automated evaluation of graphical user interfaces (GUIs) using CHIMES. *Proceedings of the Second Annual Mid-Atlantic Human Factors Conference* (pp. 57-63). Washington, DC: George Mason University/NASA-Langley, 1994.

[15]Murphy, E. D. *Application of CHIMES to the Planning and Resource Reasoning (PARR) Tool* (Report prepared for Computer Sciences Corporation under Contract No. 018750, Task 29-116). McLean, VA: CTA INCORPORATED, 1989.

[16]Mitchell, C. M.; Thurman, D. A.; and Brann, D. M. The human factor in 'lights-out' automation: Using field study data to identify critical human factors design issues. *Proceedings of the Human Factors and Ergonomics Society 42nd Annual Meeting* (pp. 364-368). Santa Monica, CA: Human Factors and Ergonomics Society, 1998.

[17]Murphy, E. D. and Norman, K. L. Beyond supervisory control: Human performance in the age of autonomy. *Third Automation Technology and Human Performance Conference Abstracts* (p. 26). Norfolk, VA: Old Dominion University; full paper to appear in M. Scerbo (Ed.), *Automation Technology and Human Performance*. Hillsdale, NJ: Erlbaum, 1998.

[18]Truszkowski, W. "Lights out" operations: Human/computer interfaces/interactions (HCI). *1996 Technology workshop – Autonomous "lights out" operations workshop: Operational challenges and promising technologies* (Presentation viewgraphs, pp. 355-367). Greenbelt, MD: Mission Operations and Data Systems Directorate, NASA-Goddard Space Flight Center, 1996.

[19]Smith, M. F. *Software Prototyping: Adoption, Practice, and Management*. New York: McGraw-Hill, 1991.

[20]Pressman, R. S. *Software Engineering: A Practitioner's Approach* (3rd edition). New York: McGraw-Hill, 1992.

[21]Boar, B. *Application Prototyping: A Requirements Definition Strategy for the '80s.* New York: Wiley-Interscience, 1984.

[22]Nielsen, J. *Usability Engineering.* San Diego, CA: Academic Press, 1994.

[23]Szczur, Martha, Usability testing on a budget: A NASA usability test case study, *Behaviour & Information Technology*, Vol. 13, No. 1, 1994.

[24]Uehling, D., *Usability Testing Handbook.* NASA DSTL-94-002, 1994.

[25]Ericsson, K. A. and Simon, H. A. *Protocol Analysis.* Cambridge, MA: The MIT Press, 1984.

[26]Carroll, J. M. Introduction: The scenario perspective on system development. In J. M. Carroll (Ed.) *Scenario-Based Design: Envisioning Work and Technology in System Development.* New York : Wiley, 1995.

[27]Gilmore, W. E.; Gertman, D. L.; and Blackman, H. S. *User-Computer Interface in Process Control: A Human Factors Engineering Handbook.* Boston: Academic Press, 1989.

[28]Kirwan, B. A & Ainsworth, L. K. (Editors). *Guide to Task Analysis*. London: Taylor & Francis, 1992.

[29]Mayhew, D. J. *Principles and Guidelines in Software User Interface Design*. Englewood Cliffs, NJ: Prentice Hall, 1992.

[30]Salvendy, G. (Editor). *Handbook of Human Factors 2nd Edition.* New York: Wiley, 1997.

[31]Sanders, M. S. and McCormick, E. J. *Human Factors in Engineering and Design*. New York: McGraw-Hill, 1992.