Intelligent Systems and Advanced User Interfaces

for

Design, Operation, and Maintenance of

Command Management Systems

(Combined Annual Reports)

Final Report

William J. Potter, Technical Monitor
NASA Goddard Space Flight Center
NAG 5-2226

Christine M. Mitchell
E24-X13
Center for Human-Machine Systems Research
School of Industrial & Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0205
404 894-4321
404 894-2301 (fax)
cm@chmsr.gatech.edu

August 1998

Background

Historically command management systems (CMS) have been large, expensive, spacecraft-specific software systems that were costly to build, operate, and maintain. Current and emerging hardware, software, and user interface technologies may offer an opportunity to facilitate the initial formulation and design of a spacecraft-specific CMS as well as a to develop a more generic or a set of core components for CMS systems.

Current MOC (mission operations center) hardware and software include Unix workstations, the C/C++ and Java programming languages, and X and Java window interfaces representations. This configuration provides the power and flexibility to support sophisticated systems and intelligent user interfaces that exploit state-of-the-art technologies in human-machine systems engineering, decision making, artificial intelligence, and software engineering. One of the goals of this research is to explore the extent to which technologies developed in the research laboratory can be productively applied in a complex system such as spacecraft command management. Initial examination of some of the issues in CMS design and operation suggests that application of technologies such as intelligent planning, case-based reasoning, design and analysis tools from a human-machine systems engineering point of view (e.g., operator and designer models) and human-computer interaction tools, (e.g., graphics, visualization, and animation), may provide significant savings in the design, operation, and maintenance of a spacecraft-specific CMS as well as continuity for CMS design and development across spacecraft with varying needs. The savings in this case is in software reuse at all stages of the software engineering process.

Assumptions Underpinning Early Work

The initial analysis conducted during the early portion of the first year of this study concluded that extensive reuse *can* and *should be* facilitated. Reuse will be greatly facilitated by making previous design experience available to developers via a case-based reasoning system. Development of a common look and feel and the use of standard commercial software will enhance commonalties across systems, and encourage reuse of components developed for one system in subsequent systems. A generic CMS architecture, which each new mission instantiates and extends, will anchor future designs to a common parent. Finally, a case-based designer's associate can guide mission-unique extensions and automatically archive new features into an integrated development environment so that the new features are easily available to designers of future systems.

Understanding Command Management Systems: How they operate? How they are developed?

The following activities were conducted and presented to Goddard management in presentations and in reports. They also the comprised background investigations that provided the basis for the papers cited in Appendix A and Ms. Jennifer J. Ockerman's M.S. thesis as well as Mr. John G. Morris' Ph.D. thesis (in progress).

- Port CMS applications to Georgia Tech
  SAMPEX CMS

WIND/POLAR
FAST (as available)
SOHO (as available)

- Identify Commonalties and Differences Across CMSs
  Attend CMS Design Reviews
  Review CMS Design Documents (SAMPEX, WIND/POLAR, SOHO)
  Interviews with CMS Developers
  Interviews with FOT responsible for developing requirements
- Conduct Task Analysis of CMS Operations
  SAMPEX
  WIND/POLAR

Articulation of CMS Commonalties and Causes of Low Reuse

- CMSs are more similar than different.
- Low reuse in part stems from failure to *standardize* on common components.
- Low reuse in part stems from a *lack of availability* of experience/information about previous designs.

Assumptions

- Reuse would be facilitated by *evolving* a corporate memory of existing systems that is *easily accessible* to developers of new systems.
- Reuse would be facilitated by providing designers with an aid with which designers can instantiate mission-specific features of a common core system. A data or case base would facilitate reuse of existing design concepts and components. As necessary, new design features, i.e., features that are added or extensions of existing features, are specified via the aid's development environment and *automatically* added to the associate's database of designs and design features.

Activities

The primary activities supported by this grant culminated in two pieces of research: The Design Browser, which comprised the primary portion of Ms. Jennifer J. Ockerman's M.S. thesis, and the Designer's Associate, which comprised the primary portion of Mr. John G. Morris' Ph.D. thesis (writing almost completed). Both of these concepts or theories were implemented as proof-of-concept computer systems NASA command management and are compatible with current systems and languages in the NASA Goddard Mission Operations environment. Finally, both systems were evaluated with actual NASA users, e.g., FOT, command management system designers, and NASA technical monitors, both quantitatively and qualitatively. In all cases, users believed and demonstrated that the systems had great potential to enhance reuse and should be fielded as commercial software based on the university proof-of-concept prototypes.

The Design Browser

With the proliferation of large, complex software systems, reuse of previous software designs and software artifacts, such as operations concepts, requirements, specifications, and source code, is an important issue for both industry and government. Reuse has long been expected to result in substantial productivity and quality gains. To date, this expectation has been largely unmet. One reason may be the lack of *tools* to support software reuse.

The Design Browser is a software architecture intended to support designers of large software systems in the early stages of software design, specifically conceptual design. The Design Browser is based on principles derived from cognitive engineering; naturalistic decision making, particularly Klein's Recognition-Primed Decision-Making Model; and Kolodner 's approach to case-based reasoning.

Essentially, the Design Browser is a database, or more specifically a *case* base, of conceptual design artifacts from previous design efforts. It provides an example of a formal *institutional memory*—a case-based retrieval system. It also proposes a *model* that specifies a category structure, with which to group concepts and a vocabulary to help designers *recognize* when a previous design or a portion of a design is relevant. The model of design concepts and vocabulary forms a *conceptual framework*, which is instantiated as a user interface for the case base retrieval system. Designers and other users of the Design Browser use this interface to search, recognize, and retrieve useful portions of previous designs.

For the Design Browser, cases are insights that generalize, or situations that highlight both desirable and undesirable aspects of a design. The Design Browser's case base contains *stories* derived from previous design experiences. Stories are defined by decomposing an entire artifact into smaller and 'interesting parts'—each part defining a case.

The Design Browser represents *stories* about design features in two ways: hierarchical text-based explanations and graphical illustrations. Hierarchical explanations make the large amount of information documenting a design somewhat more manageable and potentially more recognizable. Graphical representations complement and enhance text-based descriptions.

Each story has three parts: a *generic description*, *examples*, and *illustrations*. The generic description is hierarchical and consists of a *topic* and associated *details*. The topic is a high-level description of the components, functions, or 'lessons learned' from several designs. Details summarize features common to several designs, and they also provide lower-level information about specific designs. Details have a direct link to the second part of the story—*examples*. Examples are application-specific descriptions of system features. They may include *outcome* information, such expert opinions about the effectiveness of a particular feature. The last part of a story, *illustrations*, is one or more graphical representations that complement the text of a topic, detail, or example.

The cases/stories are organized, indexed, and accessed through the conceptual model. The model is likely to be, at least at the lowest levels, domain or organization dependent. At higher levels, it is built on the assumption that *all* artifacts, design or otherwise, can be usefully organized by decompositions along physical components and functional behavior lines. The model for the Design Browser consists of three dimensions. First there are *building blocks*, that is, the physical decomposition of the artifacts that comprise the cases. Second there are *functions*, that is, the functional decomposition comprised of sets of domain-general objects, grouped by a recognizable purpose. And third, *system issues*, that is, important system information that does not fall neatly into either the physical or functional decompositions. In software engineering, a system issue may include the rationale and outcome for choosing with a particular type of software, e.g., object oriented. Functions manipulate building blocks; issues provide a higher level view of

an entire design. Figure 1depicts the generic features of the Design Browser, the user interface and its organization of topic detail, example, etc. as well as pull-down menus to access the elements the comprise the physical and functional decompositions as well as system issues.

As a proof-of-concept demonstration, the Design Browser was implemented for a NASA satellite command and control system—the command management system (CMS). The CMS Design Browser (Figure 1) proof-of-concept served as a test bed in which the Design Browser's effectiveness could be evaluated. Three groups of users were involved in the evaluation: the eventual users of the systems (FOT), once designed and implemented; system designers; and NASA technical monitors responsible for ensuring the timeliness, quality, and integrity of the resulting software. The evaluation included four Goddard representatives from each group. Empirical results showed that the various teams involved in the specification, implementation, and management of command management software system design *all* found the CMS Design Browser quite useful.

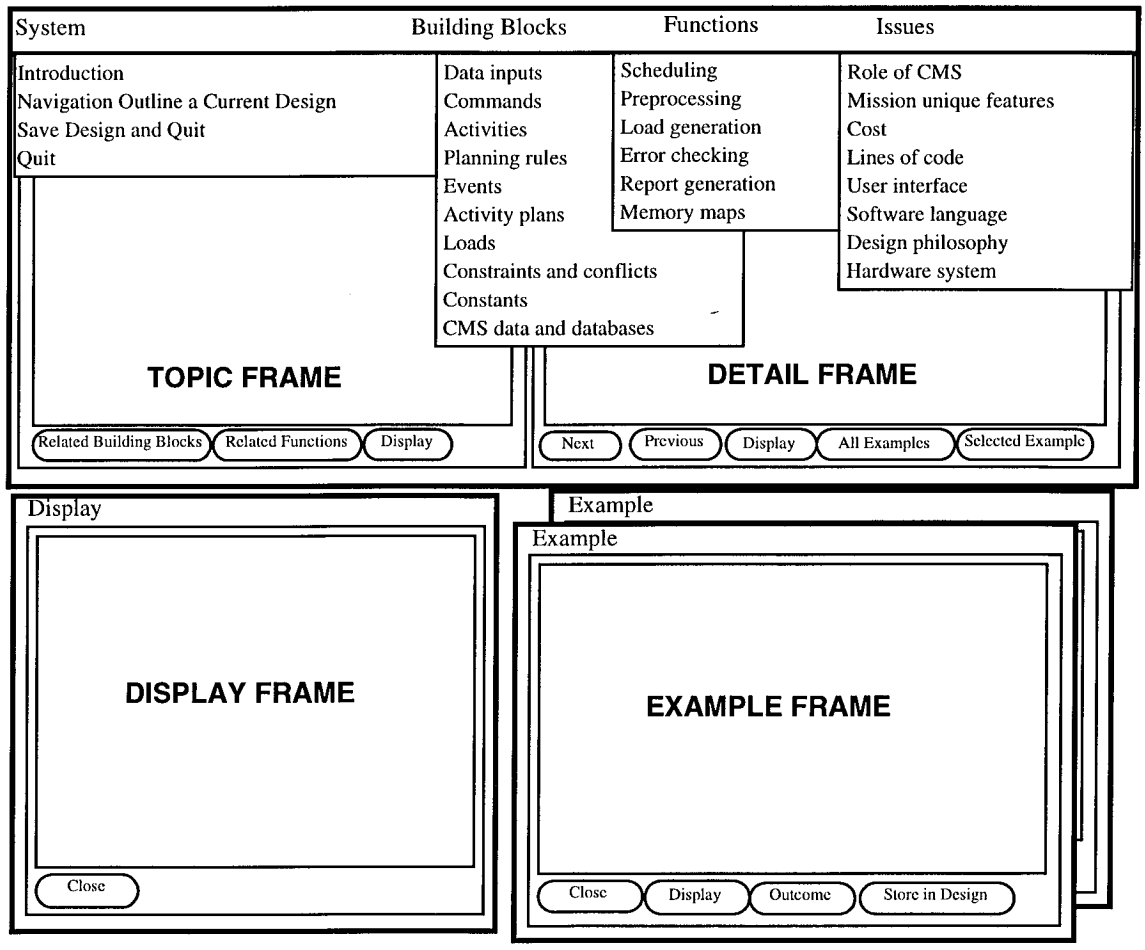

Figure 1. The Command Management Design Browser

Citations related to Ockerman's Design Browser, M.S. thesis research

This research resulted in a number of papers presented at international conferences, an M.S. thesis, an invited chapter in a text documenting innovative research in recognition-primed decision making, and a submitted journal paper.

Ockerman, J. J., Mitchell, C. M., & Potter, W. J. (1994). Case-based design browser to aid human developers in reuse of previous design concepts. *Proceedings of 1994 IEEE International Conference on Systems, Man, and Cybernetics*, 1757-1762.

Ockerman, J. J. & Mitchell, C. M. (1995). A case-based design browser to facilitate reuse of software artifacts. *6th IFAC/IFIP/IFORS/IEA Symposium on Analysis Design, and Evaluation of Man-Machine Systems,* Cambridge, Massachusetts.

Ockerman, J. J. (1995). *A case-based design browser to aid human developers reuse previous design concepts.* M.S. Thesis, Center for Human-Machine Systems, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, Atlanta, GA.

Mitchell, C. M., Morris, J. G., Ockerman, J. J., & Potter, W. J. (1996). Recognition primed decision making as a technique to support reuse in software design. In C. E. Zsambok & G. A. Klein (Eds.), *Naturalistic decision making* (pp. 305-318). New York, NY: Erlbaum.

Ockerman, J. J., & Mitchell, C. M. (1997). Case-based design browser to support software reuse. submitted for publication.

Mitchell, C. M. (1998). Software reuse: Using recognition-primed decision making as a theory to design aids for software reuse. *Fourth Conference on Naturalistic Decision Making*, Warrenton, VA, 114-123.

The Designer's Associate

The Designer's Associate is the second and major portion of this research. Building extensively on the preliminary analysis, and the knowledge acquired and lessons learned during the design, implementation, and evaluation of the CMS Design Browser, the Designer's Associate attempts to provide an institutional memory for all phases of the typical commercial design process (Figure 2). The Designer's Associate is an aid both to facilitate reuse and concurrently to incorporate a current, evolving design into a reuse library.

The Designer's Associate promotes software reuse by making most of the design artifacts from previous efforts both readily accessible and reusable. It provides tools to retrieve, reuse, adapt, and create a new artifact and to archive it as part of the normal software design process. The approach underlying the Designer's Associate (DA) has two aspects. The first represents a commitment to a specific type of software reuse practice, *compositional reuse*. The second represents a response to one of the key factors that complicates software reuse, the *lack of an effective institutional memory*.

Prieto-Díaz identifies two software-engineering techniques: compositional and generative. In the compositional case, a set of reusable artifacts captures design knowledge. Designers locate suitable artifacts within the set and adapt them to current needs. Compositional techniques are amenable to a recognition-based strategy, that is, a model such as Klein's recognition-primed decision making and Kolodner's computational implementation of case-based reasoning and *recognition*. When employing a recognition-based strategy, designers respond to the current situation in a manner that proved effective in

similar, past situations For example, reusable artifacts can be linked to the types of situations (i.e., design problems) for which they may be of use.

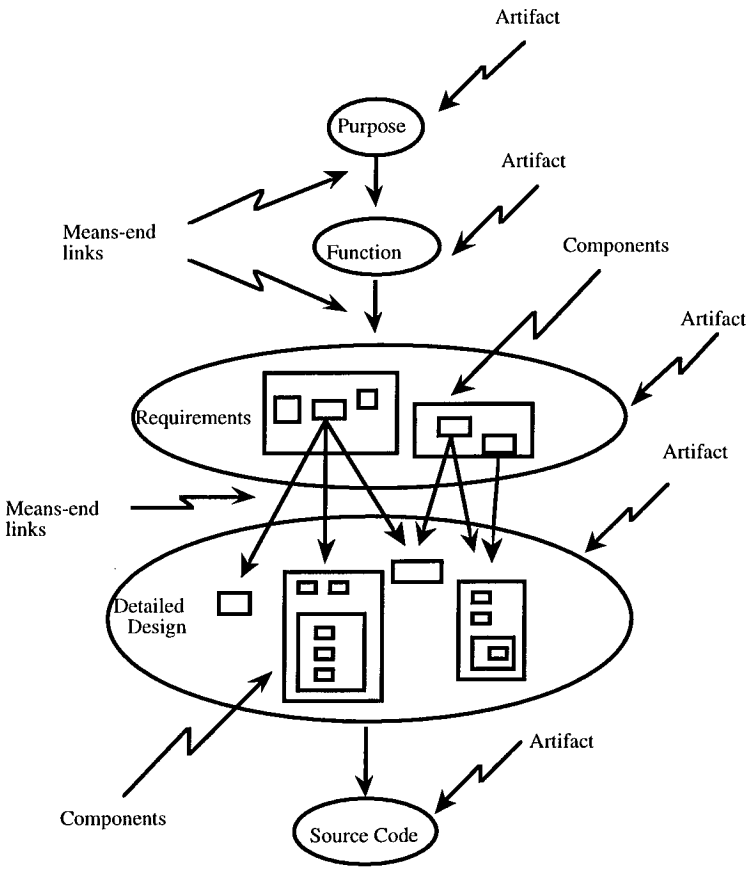

Figure 2. The Structure of the Designer's Associate and Components of a Typical Software Engineering Process

Like the Design Browser, the Designer's Associate includes the creation of a common conceptual framework for designers within a large, loosely integrated design organization and stores previous design experience in a hierarchic-heterarchic data base. The Designer's Associate uses case-based retrieval techniques to facilitate compositional reuse. Cases are classified within the case base in a manner that attempts to anticipate the types of situations in which a given case might be of interest. In addition, the Designer's Associate, building on users' needs for a design *tool* as well as a design *library*, incorporates editors, notebooks, and description templates that enable designers to create a new design from extending previous design artifacts or modifying them suitably to the needs at hand (Figure 3). The Designer's Associate lays the groundwork for an institutional memory in the form of a computer-based library of design artifacts (e.g., operations concept, requirements specification, design specification, source code).

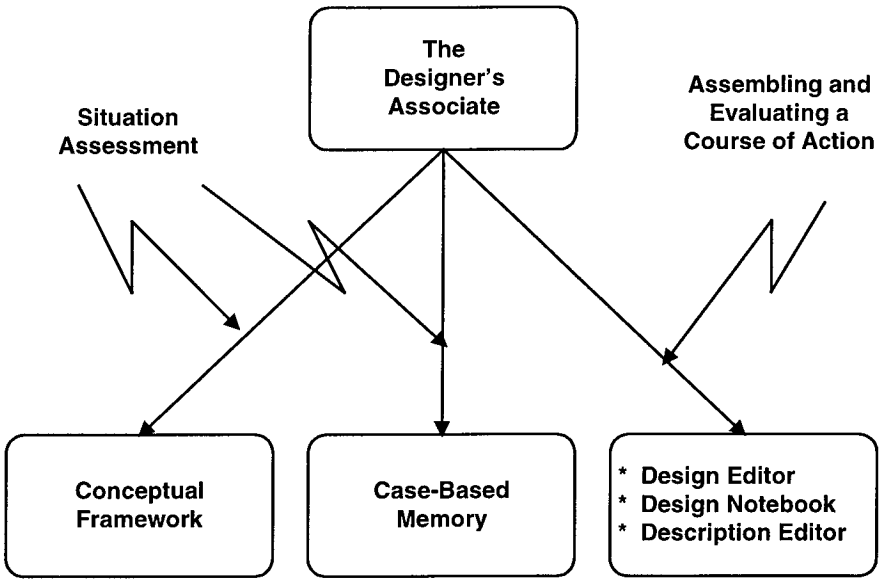

Figure 3. The Designer's Associate as a Recognition-Primed Decision Making Model

The library spans multiple development efforts and contains each artifact in its entirety. It organizes artifacts both hierarchically (e.g., by project, system, and subsystem) and as cases (i.e., as problem-solution pairs). The Designer's Associate implements this approach by providing a suite of tools for maintaining such a library as a side effect of the normal design process. These tools provide the necessary capabilities to recognize, reuse, adapt, or create artifacts central to the software design process and automatically archive the result. Thus, the artifact library grows incrementally as a side effect of tool usage.

The architecture has three facets: case structure, case indexing and retrieval, and authoring and reuse tools. The case structure describes how cases and design artifacts are organized within the library. The case indexing and retrieval describes a vocabulary and strategy for case indexing as well as the mechanics of case retrieval. The authoring and reuse tools describe key strategies for supporting the creation, retrieval, reuse, adaptation, and archiving of design artifacts.

Like the Design Browser, the Designer's Associate was implemented in proof-of-concept form for NASA satellite ground control command management systems. An empirical evaluation was constructed in order to answer two questions. First, do representative users find the Designer's Associate a usable tool? Second, does the Designer's Associate have the potential to improve user ability to retrieve and reuse existing design artifacts? Specifically, can it help designers with domain-specific experience, designers without domain-specific experience, and managers with domain-specific experience but limited software development skills? The evaluation included three groups of users: software designers familiar with the domain and task, NASA technical monitors responsible for the software development in the domain, though not necessarily familiar with the task, and professional software designers who were unfamiliar with the domain and task but are representative of 'experienced, new hires' that software organizations often include in teams for large software development projects.

Overall, the results of this evaluation are quite interesting. For high-level performance measures, e.g., efficiency by group collapsed across tasks, one might assume that, in general, that lack of experience in the domain might hinder designers' performance. If this were the case for groups of users for the Designer's Associate, then one would expect the efficiency of inexperienced designers to be significantly less than that of designers with more experience. The data did not support this hypothesis. The overall efficiency for Group 1(experienced designers) was 81%; Group 2 (inexperienced designers) was 75%; and Group 3 (NASA technical monitors) was 78%. Although some differences between the mean efficiency of each group was suggested by inspection, no significant difference was found at $\alpha = .05$.

One interpretation of this result is that the Designer's Associate lessens that handicap with regard to software reuse tasks that might normally accompany inexperience in a given task domain. This result has been replicated previously and may prove quite useful as the purpose of training, always an expensive proposition for highly trained practitioners, seeks to lessen differences between experienced and inexperienced experts. Moreover, current workforce turnover results in many 'trained novices'—that is, practitioners with domain-independent skills whose need is assistance to bridge rapidly the 'experience gap.' Systems that provide institutional memory and domain guidance, such as the Designer's Associate, can provide just this type of assistance.

Citations related to the Morris' Designer's Associate Ph.D. thesis research

As with the Design Browser, the Designer's Associate project resulted in numerous papers presented at international conferences, a book section, and, though still in progress, Mr. John G. Morris' Ph.D. thesis, with several related journal papers experts. Publications to date are listed below.

Morris, J. G., & Mitchell, C. M. (1995). A designer's associate for command and control software development. *Proceedings of the 1995 IEEE International Conference on Systems, Man, and Cybernetics*, Vancouver, BC, 3844-3849.

Morris, J. G., Mitchell, C. M., & Potter, W. J. (1995). A designer's associate: Support for the design of software for complex dynamic control systems. *Proceedings of the 6th IFAC/IFIP/IFOR/SEA Symposium on Analysis, Design, and Evaluation of Man-Machine Systems*, Cambridge, MA, 827-832.

Mitchell, C. M., Morris, J. G., Ockerman, J. J., & Potter, W. J. (1996). Recognition primed decision making as a technique to support reuse in software design. In C. E. Zsambok & G. A. Klein (Eds.), *Naturalistic decision making* (pp. 305-318). New York, NY: Erlbaum.

Morris, J. G., & Mitchell, C. M. (1997). A case-based support system to facilitate software reuse. *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, 232-237.

Mitchell, C. M. (1998). Software reuse: Using recognition-primed decision making as a theory to design aids for software reuse. *Fourth Conference on Naturalistic Decision Making*, Warrenton, VA, 114-123.

Morris, J. G. (1998). *Designer's associate: Support for the design of software for complex dynamic control systems.* Ph.D. Thesis, Center for Human-Machine Systems Research, School of Industrial and Systems, Engineering, Georgia Institute of Technology, in progress.

Conclusions

This grant funded two interesting and provocative research projects both with respect to software engineering and human-machine systems engineering. As software becomes increasingly integrated into artifacts in both every-day life, such as coffee pots, VCRs, radios, cars, as well as the increasingly slippery slope to safety-critical systems such as aircraft software, air traffic control, power plants, etc., the interface between the designers and the resulting software has become a problem within the research domain of human-machines systems engineering—not simply software engineering.

Both projects are running at NASA Goddard in their proof-of-concept form. Subsequent to the evaluation, the Designer's Associate has been converted to Java using Oracle to implement its case base. The latter system will also be installed at Goddard for the benefit of both the research and development communities.

These two projects showed that engineering approaches to solving human-machine systems problems are very productive when applied in the domain of software engineering. The latter discipline typically is domain-neutral; that is, software engineers know that they will be involved in a variety of application domains, but typical software engineering training and tools rarely take domain understanding and transition explicitly into account. Various research communities were very impressed with the obvious success of Klein's recognition-primed decision making model as a *theory* and the application of case-based reasoning as a *computational structure* to help 'trained novices' bride the learning gap more quickly and to facilitate software reuse. Both do so by building and supporting a common conceptual framework that is consistent with an organization's internal design process.

At least three research communities provided positive feedback on these efforts. First are *the naturalistic decision making researchers*, who typically focus on real-time command-and-control decisions; our work highlighted the importance of the design of software as well as its users. Second are software-engineering *researchers*, who, as mentioned above, rarely address the problem of helping designers with generic software engineering skills transition into new and complex application domains; this is an essential component to ensure the integrity of safety-critical software. Third, and finally, the human-machines systems engineering community has two cases in point of Rasmussen's initially bewildering statement that effective design and operation of safety-critical systems is a triad among the designer, the computer-based controller, and the human operator/supervisory controller.*

---

* My personal thanks to Ms. Dolly Perkins of NASA Goddard Space Flight Center. I suspected she 'fooled' Bill Potter, the NASA Goddard technical monitor for this research, and me. I thought the proposed research entailed my 'typical' control room research concerning models of operators and teams, and the use of such models for the design of 'intelligent' workstations, aids, and tutors. Bill always knew he was interested in software and how to make develop strategies to make software better, cheaper, and feaster. I was surprised to find myself in the bowels of software engineering and software reuse. I think, like many other naïve persons, that I began by thinking software reuse, meant 'code reuse.' This research has taught me many things, among the most important is that source code is the *last* place to start if the intention is to really accomplish reuse on the part of software designers and to engender a culture of reuse within an organization. Reuse must permeate the whole design process both *conceptually* and *practically*.

Acknowledgements

Jennifer Ockerman, John Morris, and I offer our deep appreciation to NASA Goddard staff and operational personnel who taught us about command management systems—so *we* knew at least one application! The software designers for the ACE command management systems were also very helpful in teaching us another variation in the command management domain of application.

Next, we sincerely thank all those who participated in the evaluations and assessments of both projects. Though for many engineers and software designers the fun is in the building, research and true progress require rigorous evaluations to identify and explain both good and bad ideas. Our thanks to the many 'users'—typically Command Management System operators at Goddard, designers—typically, software engineers from CSC who were either currently or recently working on NASA Goddard command management systems, and NASA technical monitor—whose often unhappy function was to enforce and/or measure reuse. We hope these ideas and proof-of-concepts are helpful. Also, we are grateful to many members of Georgia Tech's Center for Human-Machine Systems Research that supported both projects with both domain and technical knowledge and, at the end, provided the pool from which the last set of users in the evaluation for the Designer's Associate was drawn—professional software engineers without domain experience.

Finally, my personal 'thank you' to Bill Potter, who was a great technical monitor for research that was very exploratory in nature. He was always supportive intellectually and, as important, provided us access to the busy people at NASA Goddard who were the key to our understanding software and the problems of software reuse, developing theories for how to support it, and implementing and evaluating two proof-of-concept systems. Bill, I hope you did not loose too many friends in the process!

Appendix**

Papers

Mitchell, C. M., Morris, J. G., Ockerman, J. J., & Potter, W. J. (1996). Recognition primed decision making as a technique to support reuse in software design. In C. E. Zsambok & G. A. Klein (Eds.), *Naturalistic decision making* (pp. 305-318). New York, NY: Erlbaum.

Morris, J. G., & Mitchell, C. M. (1995). A designer's associate for command and control software development. *Proceedings of the 1995 IEEE International Conference on Systems, Man, and Cybernetics*, Vancouver, BC, 3844-3849.

Morris, J. G., Mitchell, C. M., & Potter, W. J. (1995). A designer's associate: Support for the design of software for complex dynamic control systems. *Proceedings of the 6th IFAC/IFIP/IFOR/SEA Symposium on Analysis, Design, and Evaluation of Man-Machine Systems*, Cambridge, MA, 827-832.

Morris, J. G., & Mitchell, C. M. (1997). A case-based support system to facilitate software reuse. *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, 232-237.

Morris, J. G. (1998). *Designer's associate: Support for the design of software for complex dynamic control systems.* Ph.D. Thesis, Center for Human-Machine Systems Research, School of Industrial and Systems, Engineering, Georgia Institute of Technology, in progress.

Mitchell, C. M. (1998). Software reuse: Using recognition-primed decision making as a theory to design aids for software reuse. *Fourth Conference on Naturalistic Decision Making*, Warrenton, VA, 114-123.

Ockerman, J. J., Mitchell, C. M., & Potter, W. J. (1994). Case-based design browser to aid human developers in reuse of previous design concepts. *Proceedings of 1994 IEEE International Conference on Systems, Man, and Cybernetics*, 1757-1762.

Ockerman, J. J. & Mitchell, C. M. (1995). A case-based design browser to facilitate reuse of software artifacts. *6th IFAC/IFIP/IFORS/IEA Symposium on Analysis Design, and Evaluation of Man-Machine Systems,* Cambridge, Massachusetts.

Ockerman, J. J. (1995). *A case-based design browser to aid human developers reuse previous design concepts.* M.S. Thesis, Center for Human-Machine Systems, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, Atlanta, GA.

Ockerman, J. J., & Mitchell, C. M. (1997). Case-based design browser to support software reuse. submitted for publication.

** For copies of these references please contact Dr. Christine M. Mitchell, Center for Human-Machine Systems Research, School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0205, +1 404 894-4321, +1 404 894-2301 (fax), cm@chmsr.gatech.edu.