Technical Report
CMU/SEI-91-TR-30
ESD-TR-91-30

Requirements Engineering and Analysis Workshop Proceedings

Requirements Engineering Project

December 1991

Technical Report
CMU/SEI-91-TR-30
ESD-TR-91-30
December 1991

Requirements Engineering and Analysis Workshop Proceedings

Requirements Engineering Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

John S. Herman, Capt, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.
This report was funded by the U.S. Department of Defense.

This document is available through the Defense Technical Information Center. DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145.

Copies of this document are also available through the National Technical Information Service. For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161.

Copies of this document are also available from Research Access, Inc., 3400 Forbes Avenue, Suite 302, Pittsburgh, PA 15213.

Table of Contents

List of Figures

List of Tables

Requirements Engineering and Analysis Workshop Proceedings

1 Executive Summary

Inadequate, incomplete, erroneous, and ambiguous system and software requirements are a major and ongoing source of problems in systems development. These problems manifest themselves in missed schedules, budget excesses, and systems that are to varying degrees unresponsive to the true needs of the sponsor. These difficulties are often attributed to the poorly defined and ill-understood processes used to elicit, specify, analyze, and validate requirements.

The Software Engineering Institute (SEI) hosted the Requirements Engineering and Analysis Workshop in Pittsburgh, Pennsylvania, on March 12-14, 1991. The intention of the workshop was to focus discussion on issues and activities that could help the Department of Defense (DoD) to deal more effectively with the requirements of mission-critical systems. The SEI workshop built upon work performed previously at the Requirements Engineering and Rapid Prototyping Workshop held by the U.S. Army Communications-Electronics Command (CECOM) Center for Software Engineering in Eatontown, New Jersey, on November 14-16, 1989.

The workshop participants were divided into four working groups: Requirements Engineering Process and Products, Requirements Volatility, Requirements Elicitation, and Requirements Engineering Techniques and Tools. A summary of the findings of each working group follows.

1.1 Requirements Engineering Processes and Products

The Requirements Engineering Processes and Products group investigated process-related problems in requirements engineering. The group looked at the recommendations made by the CECOM workshop participants and considered the reasons why those recommendations have not yet been implemented.

The group adopted the major issues of concern identified by the CECOM workshop: coping with uncertainty and change, reducing the difficulty in validation, resolving multiple stakeholder conflicts, and tracking requirements progress. The following strategies were proposed to facilitate adoption of the CECOM recommendations:

- Encourage innovation in the acquisition process including adaptation of DOD-STD-2167A (Military Standard, Defense System Software Development) and associated data item descriptions (DIDs) to support an evolutionary process model.
- Emphasize up-front requirements engineering, including the early introduction of specification and analysis methods and the encouragement of prototyping techniques for early validation of operational concepts and user interfaces.

- Develop a multi-disciplinary team approach, to include users, domain experts, analysts, developers, testers, and maintainers in the analysis and decision-making process.
- Capture lessons learned in the acquisition process, including both successful and unsuccessful acquisitions, then make those lessons learned available to other projects.

1.2 Requirements Volatility

The Requirements Volatility group investigated the causes of volatility in requirements and possible steps toward reduction or management of such volatility. The most important point to be made is that requirements will change throughout the life of a system, no matter how good the requirements are initially. Recognizing that requirements will change permits requirements engineers to plan for changes and reduces the unwanted side effects of the changes.

Although there were no specific recommendations from the group, the following issues were noted as needing further investigation. Requirements should be structured in a number of ways. Modular structuring is desirable because it reduces the effects of change. Hierarchical structuring is desirable because it provides appropriate levels of abstraction of requirements. View-oriented structuring is desirable because the different parties (e.g., tactical, strategic, etc.) have different understandings of the system. We need to understand how the different structuring approaches can be combined to produce requirements that are appropriate for the readers.

Volatility can be reduced by the use of prototyping in parts of the system that are not well understood at the outset of requirements engineering. More investigation of prototyping techniques is needed, however, particularly for the user interface.

The current process for system development does not easily accept the notion that requirements will change during system development. Therefore, other development processes that will allow for changes in requirements should be considered.

1.3 Requirements Elicitation

The Requirements Elicitation group determined that the primary issues in elicitation are communication, traceability, and process understanding. Communication is a major source of difficulty because elicitation is primarily a process of communication by its nature. Traceability is problematic because cost, schedule, and personnel factors force frequent updates of existing systems or systems under development. Process understanding can cause elicitation problems when the process is not defined adequately and the interrelationships among processes are loose, informal, and not well understood.

To address the identified problems, the group developed a number of recommendations, including:

- Improve communication by fostering contact between all stakeholders and removing management constraints. This can be achieved by educating managers and removing contractual, legal, and financial barriers between communicating groups, including modifications to the acquisition process.
- Improve traceability by tracking the rationale associated with the requirements and by capturing domain knowledge, ideally via reusable domain models.
- Address general process understanding by streamlining and integrating elicitation processes. This can be achieved by creating or improving tools that support "best practices." In addition, techniques can be provided that improve the capture and organization of information in meetings, assist in the negotiating process, and support the linking, traceability, and evolution of requirements.

1.4 Requirements Engineering Techniques and Tools

The Requirements Engineering Techniques and Tools group examined the major functions of requirements engineering, identified techniques and tools appropriate to selected functions, shared experiences in the use of identified techniques and tools, and identified a research agenda for techniques and tools investigation.

The group focused on context analysis, the first and least well-understood requirements engineering subprocess. Several strategies for addressing context analysis were discussed, and the experiences of group members in the implementation of those strategies were compared. As a result, several gaps in existing technology were identified, including the need to: examine group skills and group decision-making in the context of elicitation; examine incremental formalization of requirements; and develop prototyping techniques, particularly those supportive of multiple abstraction levels, functional requirements (apart from user interface requirements), and qualitative simulation to eliminate infeasible requirements.

Major recommendations for a possible research agenda included:

- Develop a support language for requirements engineering that supports generic and domain-specific information captured across the requirements engineering process.
- Enhance participative requirements development to encourage involvement of all stakeholders in the process.
- Develop techniques for prioritizing requirements in accordance with end users' needs.
- Support evolutionary requirements by investigating the use of the domain structure as a basis for stabilizing functional requirements.

1.5 Overall Recommendations and Conclusions

Although the recommendations of each working group are documented within the workshop proceedings, a number of recommendations were common among working groups or stood out as particularly important. These recommendations include:

- adopting an evolutionary, incremental approach to requirements engineering
- adopting a cross-disciplinary team approach for requirements engineering
- focusing on up-front requirements definition and validation using prototyping
- separating of user interface requirements from functional requirements
- capturing of major decisions and rationales

In the development of its strategy and goals, the SEI Requirements Engineering Project will focus on the recommendations of the workshop participants in the definition and execution of its project plan.

2 Purpose and Structure of the Workshop

On March 12-14, 1991, the SEI hosted a workshop covering the broad topic of requirements engineering and analysis. The intention was to focus discussion on issues and activities that could help the DoD to deal with the requirements of mission critical systems more effectively. The fact that improvements in requirements engineering may result in a high payoff to the DoD in terms of the cost, schedule, and quality of resulting systems has been documented previously in the writings of Brooks [BROOKS87], in the committee report "Bugs in the Program" [SIO89], and in the findings of an earlier workshop hosted by CECOM under the auspices of The Technology Cooperative Program (TTCP) [CECOM89]. Research supporting such improvement has been underway at a number of the DoD laboratories and is currently beginning at the SEI and the Software Productivity Consortium. The workshop provided a forum in which to review previous results with practitioners, to help establish productive future directions for the various research projects, and to provide a basis for potential collaboration among projects.

To emphasize the focus on practical concerns and to complement previous investigations, our call for participation was biased towards practitioners in the requirements specification, engineering, and analysis community. Further, we purposefully minimized participation from industry and vendors to avoid overly partisan viewpoints. This allowed for more free and open discussion of policy and funding issues. We do believe that industry and vendors have significant insight to offer, and we plan to encourage such participation in the future. Our invitation process targeted military and government personnel representing operational commands, development organizations, and the research laboratories. The initiation of Operation Desert Storm preempted participation by a number of invitees who had planned to attend. Total attendance for the workshop was 38.

A questionnaire was provided to invitees asking them to assign priority to various requirements engineering issues. Based on the responses, four major themes for the workshop were established, and participants were assigned to working groups to investigate those themes. The themes were:

- Requirements Engineering Process and Products (e.g., tangible outputs)
- Requirements Volatility
- Requirements Elicitation
- Requirements Engineering Techniques and Tools

Upon arrival at the SEI, participants were provided with a notebook containing various position papers, reference materials, and the proceedings from the earlier CECOM/TTCP workshop. The first day of the workshop was a plenary session. After workshop logistics were reviewed, overviews of current activities at the SEI and in the research laboratories were presented to the entire group. An opportunity also was provided for presenting summaries of additional position papers (see Appendix A). The participants were divided into working groups in the after-

noon to begin addressing the issues. The next morning, each working group gave a brief presentation of its direction and its work plan, which was discussed by the entire group. Most of the second day was devoted to the working group sessions. The results of these sessions are presented in the executive summary and are covered in detail later in this document. On the morning of the third day, the groups reconvened in a plenary session where each working group presented its findings and fielded questions or comments from the other participants. After the workshop, drafts of the findings of each working group were prepared and circulated to that group's participants for comment. The final versions of each group's findings are contained in this document.

Conclusions drawn in this document will be available to interested parties. The results are being used at the SEI to guide the work of a new Requirements Engineering project.

3 Requirements Engineering Processes and Products

3.1 Participants

Deane Bergstrom	Software Engineering Branch (COEE) Rome Labs
Harlan Black	U.S. Army CECOM
Britt E. Bray	U.S. Army
Charles W. Cratsley, III	Naval Sea Combat Systems
William S. Gilmore	Software Engineering Institute
Alec Grindlay	Navy Software Technology for Adaptable, Reliable Systems Project (STARS) Project Manager, Space and Naval Warfare Systems
Kyo Kang (Chair)	Software Engineering Institute
Nancy Mead	Software Engineering Institute
Paul Morris	Software Engineering Institute
Sandra Peay	U.S. Army Operational Test & Evaluation Command
Jay Stanley (Scribe)	DoD Resident Affiliate, Software Engineering Institute

3.2 Introduction

Application of the waterfall process in the development of complex, large-scale, multi-function systems has created problems. It forces acquisition managers and developers to write highly detailed requirements for systems whose boundaries are extremely fluid, therefore removing some flexibility in adjusting to evolving circumstances [SIO89]. This "process misfit" has resulted in:

- Long requirements development time
- Validation difficulty
- Considerable requirements volatility and traceability problems
- High development and maintenance costs
- Brittle software resulting from continuous modifications
- Unmanageable documentation

Realizing the importance of the process model, the Requirements Engineering Processes and Products Process Group investigated process-related problems they had experienced and made recommendations to address those problems.

At the beginning, the group reviewed the process models of the three service organizations and discussed problems and possible solutions. As the discussion proceeded, the group felt that they were making recommendations similar to those made earlier at the CECOM work-

shop [CECOM89]. Therefore, the group decided to review the recommendations from the CECOM workshop and investigate why they had not been implemented yet. For each recommendation, the group identified barriers to an effective implementation and made further recommendations to cope with the barriers.

3.3 Discussions

3.3.1 Coping with Uncertainty and Change

3.3.1.1 Context of the Problem

In the development of complex systems, the real user's needs are rarely understood. This results in modifications to original requirements during the development, causing schedule slippages and increased costs.

3.3.1.2 CECOM Recommendations

To address the problems caused by changes to original requirements, the CECOM workshop recommended to:

- Make changes to acquisition policies, acquisition regulations, and DoD standards to facilitate evolutionary acquisition.
- Educate contracting officers in this evolutionary acquisition approach.
- Emphasize that system requirements cannot be fully defined *a priori*, and that requirements engineering is continuous throughout the life cycles of the system.
- Look into the barriers preventing an effective implementation.

3.3.1.3 Barriers

One of the barriers the process group faces is a general misconception that the DOD-STD-2167A mandates the waterfall process model. Although DOD-STD-2167A allows developers the flexibility of tailoring to their development, they rarely take advantage of this allowance and generally follow the waterfall process model. One of the causes for this misunderstanding might be that most of the examples illustrating the use of DOD-STD-2167A are centered around the waterfall process model. This raises two issues. First, developers may be willing to use a different developmental approach, but it may be unclear to them how to fit the approach into DOD-STD-2167A. Second, developers are led to the assumption that the waterfall model is the preferred method of development for DoD projects, and tailoring is often interpreted to mean omitting steps from the "normal" developmental process.

A second barrier is the problem of fixation, a habitual behavior where developers tend to use approaches with which they are familiar. There are a number of reasons for this tendency:

- Costs, such as training and purchasing new software tools, are associated with switching to a new development approach.

- Developers tend to be afraid of deviating from "normal" developmental methods. Project managers tend to be highly motivated by potential career advancements, which leaves them anxious and skeptical when asked to deviate from the standard ways of development. The fear of possible negative repercussions from project failures using an unfamiliar developmental approach forces developers to stay within the standard operating parameters.

- Software is often developed using the mentality used for hardware development. Hardware development tends to concentrate on buying one final product, whereas software is often intended to change. In addition, funding cycles for software have been set up similar to hardware development. However, hardware-oriented funding cycles do not adequately support the development of software intensive systems.

A third barrier is the lack of an effective mechanism for clearly communicating the user's needs to the developer. Under the traditional Army acquisition process (the process is similar in the other Armed Services), the combat developer specifies user requirements, and the material developer builds systems to satisfy those requirements. Under this system, when major misunderstandings occur that require expensive software changes, the combat developer tends to claim that the material developer did not build the system that the combat developer specified. The material developer tends to claim that the combat developer did not clearly specify the system it wanted originally. All are convinced that they have done their jobs to the best of their abilities under the circumstances, and no one accepts the responsibility for the failure. As a result, no attempt is made to correct the problem for future developments.

3.3.1.4 Group Recommendations

The group agreed with the CECOM report that an evolutionary process will solve many acquisition problems and made the following recommendations to overcome the barriers discussed in the previous section:

- Teach developers how to tailor DOD-STD-2167A and provide specific examples of tailoring for an evolutionary development. Also, teach developers techniques for tailoring other proven developmental processes.

- Provide incentives for using innovative developmental processes. Make the benefits associated with an innovative developmental process worth the risk of failure.

- Define, if possible, a developmental process that works for a particular type of systems and establish policies that mandate the use of the process for the same type of systems.

- Set up funding cycles that support an evolutionary approach, and appropriate funds as the system evolves.

- Place one person on a project throughout the entire acquisition process. This person would then be ultimately responsible for the success or failure of the project. The Army has started a program where colonels and captains are placed on projects from mission analysis and remain on the projects

throughout the entire acquisition process. Although this is a new program and there is no data to support its success, the group strongly felt that this approach would solve many problems associated with the current acquisition process.

3.3.2 Improving Requirements

3.3.2.1 Context of the Problem

The combat developer analyzes new enemy threats and specifies requirements of systems that will counter the threats. The material developer then builds systems to satisfy these requirements. The bridge that has traditionally linked these two activities together is the requirements document. Although contractually binding, this requirements document does not truly represent the real user's needs.

3.3.2.2 CECOM Recommendation

The following recommendations were made at the CECOM workshop for the problems discussed in the previous section:

- Remove the excessive DoD barriers to communication between contractors and users.
- Update acquisition policies to support evolutionary life cycles.
- Increase awareness of prototyping methodologies.

3.3.2.3 Barriers

One of the barriers to communication between contractors and users is the reluctances of the program office to release the control they currently have over this interaction. They feel that their control eliminates excessive changes to the system while it develops. While there may be some rationale to support this argument, drastically limiting the user's involvement with the development is apt to be counterproductive. Developing systems that stay within schedules and meet deadlines is of little or no use if the final system does not help users accomplish their mission.

As of now, there is no mechanism for capturing and disseminating information on requirements validation techniques. This means there is no concrete evidence to support the selection of a requirements validation technique for a typical project. Although prototyping seems to be the agreed upon approach for resolving many validation problems, mechanisms for selecting the best prototyping techniques and tools, and disseminating lessons learned from the use of these techniques, are not available. Therefore, program offices cannot find evidence of the benefits of prototyping and do not realize how much effort could be saved if prototyping were an integral part of most developments. Program managers could save time and effort investigating such issues if this mechanism were in place.

Another point is that even if prototyping were allowed and mechanisms to disseminate information were in place, many projects might not use this technique without a policy requiring them to do so. The general consensus is that if prototyping is not a mandatory step in the pro-

cess and if developers are not contractually bound to use it, they will leave this step out of the development.

Often, the realization of a developer's ability to interpret specified requirements happens after the contract is awarded. The developer's ability to perform this task should be exposed *a priori*.

With the way the current project funding cycles are set up, the majority of funds are allocated for the full-scale development phase, with little emphasis placed on the concept exploration phase. The main reason for this is because of the way DoD funds are appropriated; that is, the quicker a project can get up to full-scale development, the more likely the project will be allocated funds.

The problems of fixation and reluctance to explore new technologies exist in validation, as they did with coping with changes and uncertainties, for many of the same reasons: extra costs, career advancement concerns, lack of policies, etc.

Lastly, there are concerns that when customers are shown prototypes, they are left with the impression that the system is complete and fail to appreciate the amount of time and effort needed to build the actual product. This is a serious concern among the developers and could discourage them from using prototyping techniques.

3.3.2.4 Group Recommendation

The group made the following recommendations to overcome the barriers discussed in the previous section:

- Allow users to be more involved in the development of systems and place the users directly on the development cycles. Track and record the results of this involvement to provide guidelines for future developments.
- Develop a mechanism for capturing lessons learned from the use of requirements validation techniques. One suggestion is to organize "red teams" that periodically evaluate and document requirements engineering approaches during system development (this information would be confidential). This information can then be used to develop, revise, and modify requirements validation strategies.
- Change policies and mandate the use of prototyping. Make prototyping the norm rather than the exception. If developers decide not to use prototyping, they must provide a convincing argument to support their decision. Educate the people who are responsible for policy changes on the benefits of prototyping. Mandating prototyping would have an effect on many of the problems mentioned earlier and would offer the following benefits:
  - With mandated prototypes, users can always expect to receive and sign-off on a prototypical model of the final product whether or not the users are directly involved with the development. This will help to clear up a majority of requirements misunderstandings.

- The problem of capturing and disseminating the information on requirement validation techniques will become less significant. Even if there is no formal mechanism to do this, users and developers will gain an intuitive understanding of best approaches and tools to use through their experience.

- Prototyping should significantly decrease the amount of money squandered on systems that do not satisfy the customer's mission. Before major portions of systems are allowed to evolve into full-scale development, there will be a rough example of the final product. This will provide the users with something more tangible than volume after volume of documents that are difficult to understand.

- Award contracts to the contractors most capable of understanding the users' needs. If prototypical examples demonstrating the contractors' ability to extract pertinent user requirements and illustrating their approaches to achieving the system objectives are used as a criteria for awarding contracts, more effort will shift to the front end of the acquisition process and more incentive will be generated to understand the users' needs early in the development. With contracts awarded based on prototypes, the key to winning the contract for full-scale development is being the most capable of understanding users' needs and quickly and effectively demonstrating their understanding to the user. It is certain that as contractors begin to lose contracts due to poor validation techniques, corporate databases will quickly fill with lessons learned on understanding customer requirements.

3.3.3 Resolving Multiple Stakeholder Conflicts

3.3.3.1 Context of the Problem

DoD software systems are usually developed to service many users in various situations. Each user has certain requirements and reasons for these requirements. More often than not, many of these requirements are omitted or conflicts between requirements are unresolved before full-scale development.

3.3.3.2 CECOM Recommendation

The CECOM workshop recommended to develop and document a procedure that can be used to capture the complete set of requirements.

1. Identify and define all significant viewpoints and stakeholders.
2. Determine and define requirements from each viewpoint.
3. Communicate viewpoints and requirements to all stakeholders.
4. Jointly evaluate requirements.
5. Continually negotiate a reasonable envelope.
6. Iterate through all activities until system retirement.

3.3.3.3 Barriers

Users often have different requirements for different reasons, and these requirements can range over the entire spectrum of criticality. Some of the participants felt that the requirements which they had voiced in the past failed to be implemented in the products and, as a result, the products were difficult or impossible to use when completed.

The group felt that while the CECOM recommendations were viable solutions to the problems within requirements engineering, there are no fixed policies or incentives to assure that these recommendations are actually incorporated into the current military acquisition model.

It was also pointed out that the current life-cycle model contributes to the multiple stakeholder problem by not having an explicit phase for resolving requirements conflicts prior to development.

3.3.3.4 Group Recommendation

The following recommendations were made by the group to alleviate the problem of multiple stakeholders:

- Enhance the life-cycle model to accommodate more up-front planning.
- Acknowledge importance of multiple viewpoints.
- Provide incentives or policies to implement the CECOM recommendations.

3.3.4 Tracking Requirements Progress

3.3.4.1 Context of the Problem

As was mentioned earlier, users' needs are not fully understood prior to the system development and, as a result, requirements tend to change frequently during the development. If every new requirement arising during development were to be incorporated into the system, it is conceivable that the system might never be fielded.

3.3.4.2 CECOM Recommendation

The following recommendations were made at the CECOM workshop:

Management

- Modify award fee structures to encourage the creation and timeliness of requirements specification. Current contracts often encourage the freezing of requirements and discourage subsequent changes of those requirements.
- Develop a team approach to help reduce unrealistic expectations on the part of the user/customer.
- Educate program managers and team members that "changing your mind" as a result of new information is acceptable.
- Train government program managers in the use of acquisition models that use prototyping.

Development

- Apply the new metrics developed above on actual projects.
- Develop an explicit requirements validation plan for every project.

Research

- Develop and use effective metrics to measure requirements progress and completion.
- Develop more rigorous risk assessment and risk management techniques.

3.3.4.3 Barriers

Policies are not in place to ensure that the recommendations from the CECOM workshop will be used at a global level. The participants from CECOM mentioned that they had begun using these techniques but that the approach was not widespread.

3.3.4.4 Group Recommendation

The group recommended a team approach to systems development, which includes people from every phase of the acquisition process. It was emphasized that system engineers and testers should become more involved in the requirements engineering process because they have a better understanding of the whole system and will be more capable of determining the impacts of each requirement. Also, software testers should play an active role in requirements engineering because they are ultimately responsible for testing the requirements. Before software testers can confidently begin their work, requirements ambiguities have to be resolved. Allowing testers to become more active in the requirements engineering phases would provide more incentives to resolve requirements conflicts.

Another recommendation was to consider three systems during the development: the actual system being implemented, a "wish system," and a system in between. The idea was to freeze the requirements for the system being developed and place all new requirements in the "wish system." As we gain a better understanding of the system, we can then distill the "wish requirements" and place them into the "intermediate system." The intermediate system serves as a compass in the development of the actual system and exposes things that need to be considered in the current system for a smooth transition to the next version. The distillation process could be similar to the process mentioned earlier in the CECOM recommendations. This technique would allow end users to participate in specifying requirements without significantly affecting the development schedule of the current version.

3.4 Summary of Recommendations

The recommendations made by the Requirements Engineering Process and Products Group are summarized below.

- Encourage Innovation in Acquisition Process

  Although an evolutionary acquisition strategy has been proposed to address many problems with the waterfall model, this strategy has generally not been practiced. The group attributed slow acceptance to the problem of fixation, i.e., people tend to be resistant to changes, especially when the effects of the changes are unclear to them. The group recommended that there should be both a policy to mandate the strategy and an active education program to raise the awareness of the benefits of the new strategy. The group also recommended an adaptation of DOD-STD-2167A and DIDs to support the evolutionary acquisition. It was the opinion of the group that specific guidelines on how to use DOD-STD-2167A and DIDs with the evolutionary model should be provided with the standards, and all necessary changes be made to the standards.

- Emphasize Up-Front Requirements Engineering

  The operational concept of a target system is documented as the statement of operational needs at the beginning of the acquisition process. The specification and development activities of subsequent system requirements depend largely on this document. However, in most cases, the operational concept development is done without following any method or using any tools, and the concept documents tend to be ambiguous and subject to various interpretations. The group recommended that specification and analysis methods be introduced early in the acquisition process to address this problem. The group agreed that prototyping techniques that allow validation of operational concepts and user interface should be introduced early in the development.

- Develop a Multi-Disciplinary Team Approach

  Many decisions made early in the life cycle constrain design and implementation choices. Some seemingly simple decisions can constrain design freedom or can have a substantial cost during the maintenance phase. The group recommended that a multi-disciplinary team approach, which includes users, domain experts, analysts, developers, testers, and maintainers, be introduced from the beginning of the acquisition process. The group agreed that this approach should be adopted when an incremental acquisition is followed, as the system must be tested and maintained for each increment. The tester's and maintainer's view must be reflected when architectural decisions are made.

- Capture Lessons Learned in the Acquisition Process

  The group recommended that the lessons learned from both successful and unsuccessful acquisitions be captured and made available to other projects. The group agreed that there should be a central location where this information is maintained and disseminated. However, the group could not make any specific recommendation on how the information could be gathered, especially from unsuccessful projects.

3.5 A Summary of the Results from the CECOM Workshop

The key process-related problems identified by the Requirements Engineering Process Working Group (at CECOMs workshop) and their recommendations for those problems are summarized below.

3.5.1 Problems

1. Uncertainty and change are difficult to cope with.
2. Validation of requirements is critical to project success.
3. Multiple stakeholders make it difficult to reach closure.
4. We do not know how to track progress in requirements development.
5. Different processes are needed for different problems.
6. Differentiations are unclear between the system/software requirements analysis phase and design phase.
7. The existing inventory of systems needs to be retrofitted to new requirements engineering technology.

3.5.2 Recommendations

- Freeze requirements in small incremental builds.
- Develop more testbeds like AIN to validate interoperability earlier in the development process.
- Develop and transition techniques to isolate requirements partitions.
- Develop and transition new techniques to accommodate change in requirements and designs.
- Develop and refine practical formal requirements techniques.
- Define a multi-stakeholder requirements process.
- Develop thorough understanding of requirements "normalization." (Somewhat analogous to database normalization, this envisioned technique would enable two sets of requirements to be shown to be equivalent.)
- Define and understand requirements process models.
- Define and understand models of requirements progress.

- Perform experiments to determine what conditions make evolutionary acquisition and prototyping practical.
- Develop tools and techniques to capture merits and trade-offs among requirements.

4 Requirements Volatility

4.1 Participants

Don Harris	USAADASCH
Connie Heitmeyer	Naval Research Laboratory
Leslie E. McKenzie	Air Force Computer Acquisition Center
Jimmy Parker	USAFAS
Patrick Place (Chair)	Software Engineering Institute
Bill Reid	Defense Systems Management
Kim Stepien (Scribe)	DoD Resident Affiliate
George Sumrall	CECOM Center for Software Engineering
Bill Wood	Software Engineering Institute

4.2 Purpose of the Group

This group was given the charter of investigating the causes of volatility in requirements and making suggestions on reducing or managing such volatility. Members of the group had varied backgrounds, with the majority representing acquisition organizations and the minority representing users.

Volatility in requirements causes a number of problems for software developers. The main problems are if the changes are ignored, the delivered system will fail to satisfy the customer's current needs and if the changes are accepted, there may be delays as the system is altered to meet the changed requirements. Accepting the changes generally means cost overruns and delays in delivery schedules. Understanding the causes for volatility is important for understanding ways to reduce volatility in requirements or, failing that, to reduce the effects of volatility on requirements.

4.3 Discussion

4.3.1 Planning for Changes

We identified that requirements will change, so it does not make sense to attempt to freeze requirements at the start of system development. If requirements are frozen in such a manner, as the system develops and the environment changes, the system will become obsolete before development is complete. Freezing requirements prior to the development is the approach that has been traditionally adopted within the DoD. The group agreed that this process does not work.

There are a number of causes of change in requirements that should be taken into account. As the system develops, the users become more knowledgeable about the system and better understand their needs and the way in which a system can fulfill those needs. This inevitably

leads to changes in the requirements. So even when the requirements are perfect in terms of understandability, consistency, and completeness at the start of development, there will be a need to change the requirements as the development progresses.

The volatile nature of requirements is not well understood and is generally not planned for in system development. We did not discuss the political or contractual implications of planning for change, only the technical considerations that make changing the requirements a possibility.

The group's recommendation for managing changes to requirements include assembling a requirements development team that will work on making changes to the requirements based on new information. The new information will include the changes in the environment for the system, the increased understanding of the user's needs, and the feedback from system development. This team should be made up of representatives from all interested groups, the users, the developers, and the program offices. The representatives must have the ability to commit their organizations to the changes they make, or at least receive a speedy response to requested changes. As the system development continues, proportionally less work will be needed for changing the requirements.

It is important to plan for the requirements changes in the requirements document and structure that document so that one part may be changed without altering other parts. The style of document used on the A7E project was discussed and suggested as a model of a document designed for change. A good system is designed in a modular fashion and the group believed that it is important to construct the requirements document in a similar modular fashion.

4.3.2 Specifying Requirements for the User Interface

There was a great deal of discussion about issues of user interface because in the early stages of development, requirements for the interface are likely to be very volatile as the human factors staff, the real users, and the developers discuss the optimal user interface to a system. The interface to a system is often the user's primary area of concern. A poorly designed user interface will make a system difficult or impossible to use. If developers change the interface without good reason, it is likely that the users who have become accustomed to an existing interface will be unhappy. Prototyping the user interface as early as possible may be a way to reduce later volatility in the interface requirements.

The requirements document may need to describe the user interface to the system. In order to describe the user interface in the requirements document in an alterable way, the interface requirements should be localized within the requirements so that any changes may be made without affecting the entire document. The group agreed that although the interface requirements initially will be highly volatile, at some point, when the users have accepted the interface, they will become highly stable and will change only in extreme circumstances.

There was some debate as to whether or not the user interface should appear in the requirements document. Some time was spent discussing a hierarchy of requirements documents,

essentially a very high-level design of the system, and the group felt that at some level within the hierarchy of requirements documents, the user interface would appear.

4.3.3 Prototyping

Prototyping was discussed as a way to reduce volatility in requirements. We have already discussed that as users deploy a system, they become more knowledgeable about the way the system can satisfy their needs. If a prototype of the system is released to users (as quickly as possible) and if the prototype shows the concepts of the system, then the users will learn about their needs earlier in system development than if they waited for the first release of the system. Knowledge gained by the users about their needs and changes resulting from this increase in knowledge may be fed into the requirements engineering group and subsequently into development.

There was concern about prototyping user interfaces on equipment other than the final hardware, since doing so may build false expectations about the user interface. For example, if the user interface is prototyped on a Macintosh, then users would naturally expect an interface like Macintosh on the delivered system and may be disappointed if the delivered system has some other interface.

4.3.4 Testing

One disadvantage of volatile requirements is that it makes the job of the testers significantly more difficult. Test development generally proceeds with system development, and the tests are developed from the requirements documents. As the requirements change, so will the tests. Therefore, the tests need to be designed in a modular fashion.

4.3.5 Improving the Quality of the Requirements

The group agreed that there are three goals of system development: 1) To develop a quality system; 2) To deliver the system according to schedule; 3) To deliver the system at the agreed cost. The group accepted that any two of these goals can be achieved, but only at the expense of the third goal. If a system starts to overrun its plan, the development team has a number of options. They can either delay delivery, use more developers, or decrease the function available in the delivered system. The decrease in function implies that either the system will not perform as required in all cases, or it may be less robust. In either of these cases, there will be a decrease in system quality. Since quality is the hardest attribute to measure, it is the goal that normally suffers in development.

In order to induce quality in development, we should strive for quality in requirements. One of the causes of volatility in requirements is that the requirements are incomplete, inconsistent, or ambiguous. If we use tools to remove ambiguities and inconsistencies in the requirements document, then the developers will be better able to meet cost and schedule requirements while maintaining the quality of the system. Such tools are available but are not widely used.

Therefore, it was suggested that these tools should become part of the requirements development process.

4.3.6 Reporting Development Changes

There are times when, for practical reasons, a developer will make changes that affect requirements during the development process and fail to alter the written requirements documents. The group felt that the developers should conform to the requirements, but that this sometimes leads to a situation where a system cannot be implemented. In cases where the requirements are not applicable or harmful to the operation of the system, the requirements document should be changed to alter the "broken" requirements. Since such changes inevitably will occur, the developers must be prevented from sweeping the changes under the carpet and should be encouraged to report the changes. Further, the rationale behind the changes should be captured. At present, this is not required.

4.3.7 Using Incremental Development

The group felt that the introduction of a process of incremental development and delivery would be advantageous. Such an approach to system development and delivery affects the requirements in a number of ways.

First, the requirements must be structured so that they indicate useful subsets of the system that can be delivered incrementally. If no useful subsets can be identified, then the incremental approach evolves into existing practices of completing development of the entire system before delivery of any part of the system. A minimal approach to subsetting the requirements might be to assign a need level to each requirement. The users then should be encouraged to assign factors other than their requirements.

Users will get to experiment with early parts of the system, thus providing feedback to the developers in order to achieve higher levels of user satisfaction. As the users deploy the partial system, they will discover more about their needs and any problems with the partial system. Second, incremental development and delivery allows the developer to incorporate user feedback.

4.3.8 Tracing Changes

Given that the system is developed to the requirements, changes in the requirements necessitate changes in the system. Although current practice does not trace the requirements into the system development, doing this would be valuable. Tracing the requirements makes the task of tracking changes to the requirements into the system development simpler. It was generally agreed that advances in the development environment need to take place for requirements tracing to be feasible.

4.3.9 Using Hierarchical Requirements

The group felt that a hierarchical approach to requirements may make it simpler to adapt the requirements to necessary change. The number of levels of requirements was not precisely determined, but it was generally agreed that there should be three to five levels of requirements. The most abstract view of the requirements would be at the top of the hierarchy and the most detailed at the bottom. Changes then could be made at the appropriate level of the requirements without affecting upper levels of the requirements.

Further, the differing viewpoints of the system need to be clearly separated in the requirements document. We identified the following viewpoints: executive (strategic), commander (tactical), operator, maintainer, and hardware (or other software products).

4.3.10 Applying Technologies

Some technologies applicable to requirements engineering were discussed. David Parnas and other computer scientists at the Naval Research Laboratory developed an approach for the development and description of requirements. The approach leads to a modular description of the requirements that is easier to change than the more traditional style of requirements description. Each major aspect of the functional requirements is given a section in which all of the functions of the system that pertain to that aspect are described. The section headings of the document are:

1. Introduction
2. Constraints
3. Input and Output Data Items
4. Modes
5. Functions (closely linked with timing)
6. Timing (performance)
7. Required Subsets
8. Likely Areas of Change
9. Terms

The above technology presents the requirements in a way that makes them easy to change.

Another approach is to attempt to reduce how often the requirements need to be changed. Although this approach will not eliminate the need to change requirements (e.g., environmental changes cannot be predicted *a priori*), improving the quality of the initial requirements will reduce the need to change the requirements due to their internal flaws. For example, if the requirements are inconsistent, they must be altered to achieve consistency. Formal methods were suggested as a way to improve the quality of the initial requirements, offering the requirements developers analysis techniques that improve the consistency and completeness of the requirements. Formal methods still need some work, though, before they can be applied to ev-

ery system. However, wider use should be encouraged as a means to improve the quality of the requirements.

4.3.11 Changing the Development Process

A typical, current process for development of systems is to develop the requirements and then attempt to freeze them while development takes place. We already identified that requirements do not remain stable. Therefore, the existing process is deficient. By the time the system has been developed and delivered, the requirements will have changed.

Thus, the requirements and system development process must be changed to accommodate the volatile nature of the requirements. This means that the process of developing requirements must be re-examined with particular attention paid to the process for changing, adding, or deleting requirements. The group agreed that the development of a hierarchical requirements document would lead to valuable changes in the process of requirements engineering.

4.4 Group Recommendations

Since it is not possible to eliminate volatility in requirements, and since some level of volatility is desirable, it is important to understand the fact that requirements will change and to plan for these changes at the beginning of development.

Requirements need to be encapsulated in order to localize the effects of a change. A particular example is the separation of the requirements for the user interface from the requirements for the application. At the start of user interface testing, the user interface may need to be changed very rapidly while the application remains constant.

It is important to prototype any piece of the system where there is doubt about its operation or interface with the user. As users employ a system, they become more knowledgeable about that system, and will understand better what the piece of the system should do. The sooner users are given this opportunity, the less development time is wasted.

5 Requirements Elicitation

5.1 Participants

Robert Austin	Software Engineering Institute
K.C. Burgess-Yakemovic	NCR Human Interface Technology
Luke Campbell	Naval Air Test Center
Mike Christel (Scribe)	Software Engineering Institute
Peter Hanke	Naval Air Systems Command
Robert Helm	University of Oregon
Howard Reubenstein	The MITRE Corporation
Scott Stevens (Chair)	Software Engineering Institute

5.2 Introduction

Past research in the domain of requirements elicitation has often oversimplified the process, restricting it to a model in which a user or set of users gives information to a requirements analyst. In reality there are many more communities involved in the elicitation process, and the information flow is bidirectional between most of the involved communities. While the terminology for these communities varies across the different military services, they can be categorized as follows: operators (including end users, maintenance people, testers), developers (including contractors), requirements documenters, and customers (the sponsors/funders).

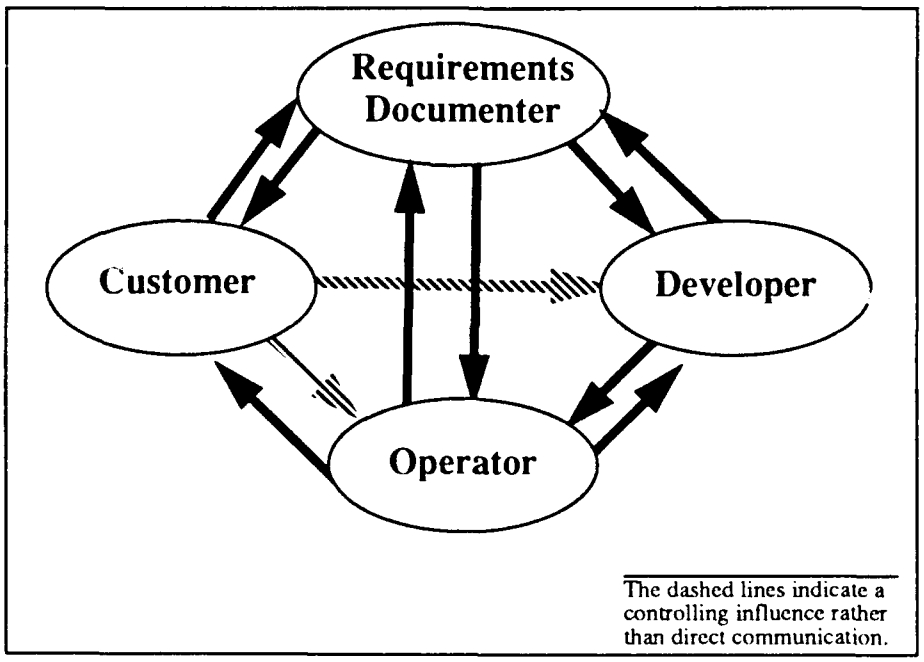


Figure 4-1 General Model of Communication for Requirements Engineering

Elicitation can be defined as the process of identifying needs and bridging the disparities among the involved communities for the purpose of defining and distilling requirements to meet the constraints of these communities. Requirements can be viewed as the clearly defined results of a negotiation between the parties involved. Elicitation is seen as a negotiation process. This elicitation process will be described further in the next section, along with examples of how requirements elicitation are currently conducted in the Navy. Some general problems with this process will be discussed, including communication issues and traceability issues. The final section will address some of these problems and offer suggestions on how to improve the process.

5.3 Current Elicitation Practices

An example of requirements elicitation from the Navy will be used as a starting point for a discussion of current elicitation practices. The method for creating software requirements within the Navy will be presented in general terms, followed by a description of this method as it is applied to the development of the AYK-14 standard airborne computer.

Elicitation procedures described in this Naval example will be discussed in greater detail. This discussion will be supplemented by the working group's experience with other elicitation practices.

5.3.0.1 An Example from the Navy

The evolution of software requirements within the Navy can be broken down into the following steps:

1. Users perceive a "threat."
2. A top-level Navy organization figures out the high-level requirements to address that threat. These high-level requirements are typically a few pages long and are referred to as the "operational requirements (OR)."
3. A system command (SYSCOM) turns the OR into a detailed requirements statement. This is typically accomplished through the following activities:
    a. In-house domain experts (the people at SYSCOM) create requirements based on their knowledge of technological innovations in the domain area and past systems in use in that domain. These experts are expected to remain informed about their domain, and have experience in extrapolating new requirements based on what has existed in the past.
    b. Compliance with constraints from the customer community, e.g., Congress, dictates that a particular standard supporting reuse must be followed by the hardware/software.
    c. Higher-level managers within SYSCOM influence the requirements statement by stressing certain points.

d. If funds are available, end users are brought into the process. These users may be first educated in the new technologies for a given domain by bringing them to the appropriate government laboratories. The users are typically interviewed in an ad hoc manner to derive their requirements.

4. The detailed specification is passed to contractor(s). This Navy specification typically contains exclusions on what the contractor cannot do, but leaves open some implementation choices for the contractor to decide.

The general process can be illustrated with an example. This example concerns the development of requirements for the AYK-14 standard airborne computer (SAC). The steps are numbered to correspond to the general description of the process above. In addition, the figure is used to reference the parties involved in the development of these requirements according to the communities named in the general model of elicitation proposed by the working group.

1. The "threat" in 1976 that initiated this project was the existence of logistical difficulties in supporting a wide class of different computers.

2. In 1976, the Office of the Chief of Naval Operations (OPNAV) developed ORs that addressed the threat and called for the development of a standard set of similar computers for the Navy to support in the future. OPNAV may have had some high-level communication with NAVAIR during the development of the OR (also referred to as the technical operations requirements, or TOR). The group within OPNAV working on the TOR is referred to as the program sponsor. The group within OPNAV where the purchasing power originates is referred to as the resource sponsor. It is possible to have an intersection between the program sponsor and resource sponsor.

3. NAVAIR developed a detailed specification from this OR in 1977. NAVAIR thus represented the requirements documenter community. The OR was approximately 3 pages long, and the detailed specification for the AYK-14 SAC, labeled the 85518, was about 100 pages.

   Approximately 10-12 people from NAVAIR added volume constraints, weight constraints, power constraints, and other details in developing the OR into a detailed specification based on prior knowledge of the available technology. Since this effort was an integration of existing capabilities into a standard system, boilerplating was done to reuse pieces of past specifications.

   No constraints were derived from past systems because this was a new system.

   Experts from other Navy Labs (NATC, Naval Avionics Center (NAC) Indianapolis) were sought for consultation when needed.

A computer resource working group (CRWG) was created and headed by the program manager from NAVAIR. The CRWG gathered information from end users, i.e., the operator community. It then passed that information to NAVAIR during the development of the detailed specification via regular (quarterly) meetings. The Operation Advisory Group (OAG) also brought together the operator community and the requirements documenter.

4. Multiple companies competed for contracts from 1978-1980, with Control Data Corporation (CDC) eventually winning the competition.

Users received the system from CDC in 1980. Requirements for an updated system were then begun, and the above steps were repeated. This trend occurs repeatedly in the military. A system is created and by the time it is released a new OR is developed to create an update to the current system. Therefore the four steps detailed above are really part of an iterative process. The next phase of the AYK-14 SAC took place from 1980-1983, and the phase after that from 1983-1987.

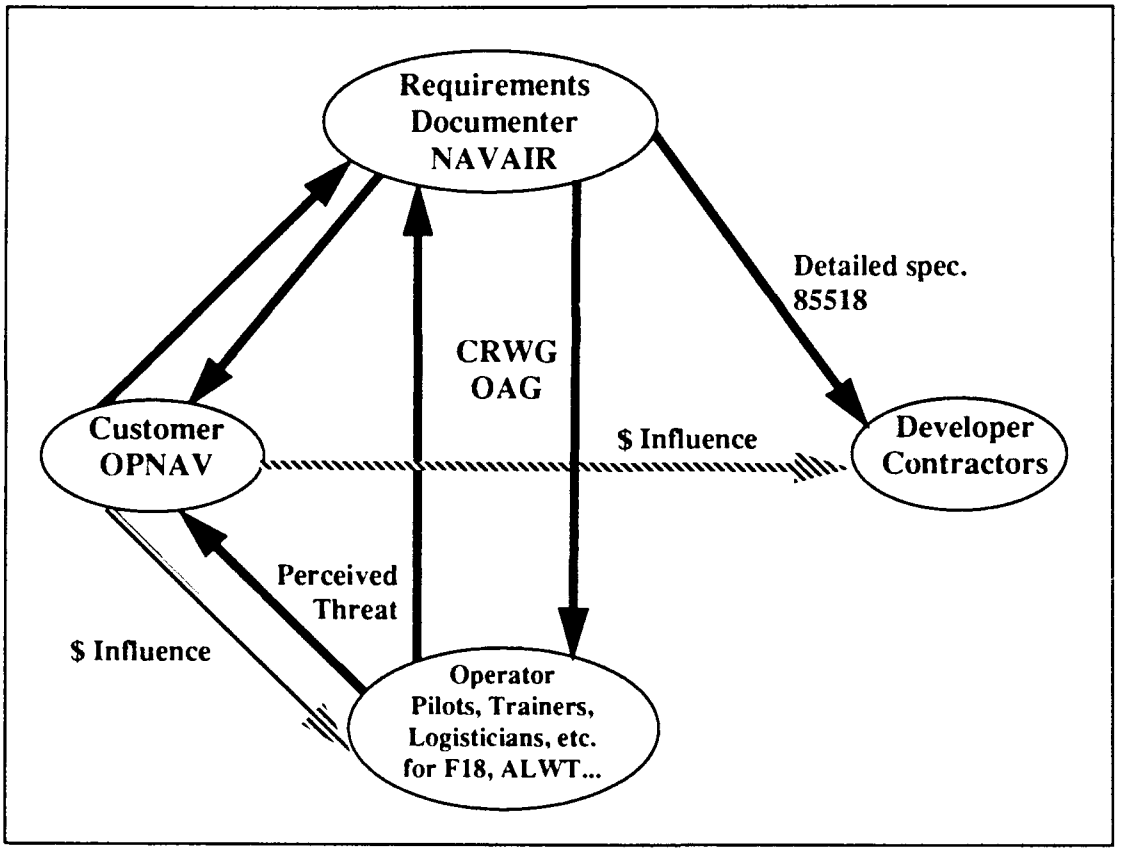


Figure 5-1 AYK-14 SAC Example of Requirements Elicitation

5.3.1 Common Practice

Requirements elicitation is primarily a communications process involving different communities. In the example from the previous section, communication between the operator community and requirements documenter community occurred through the guidance of the OAG and CRWG. There was initial discussion between the customer community (OPNAV) and the requirements documenter community (NAVAIR) during the creation of the OR. Much communication took place within the requirements documenter community, e.g., between Navy laboratories, as this community often represented the operator community rather than consulting them. In the example, there seemed to be very little communication with the developer community. This point will be returned to in the next section.

Most of the communication for elicitation takes place through meetings. In the AYK-14 SAC example, meetings were the primary means of communication between the operator and requirements documenter communities. Since meetings are used so frequently in elicitation, many of the elicitation techniques focus on improving the quality of meetings. These techniques include JAD, IBIS, Odyssey, and Delphi. The problems addressed by these different techniques range from getting the right stakeholders with authority together for the meetings so that decisions can be reached relatively quickly (JAD), to capturing and organizing the rationale behind requirements gathered during meetings (IBIS).

Logistical problems often make meetings difficult or impossible, so there are other ways the communities involved in elicitation communicate. These include phone conversations, electronic mail, and increasingly, video teleconferencing.

Because a large number of proposed systems are evolutionary, not revolutionary, communication is enhanced through the boilerplating of past documents and standards to meet current needs.

Communication is often especially difficult between the developer and the other communities. The developer has clear ideas of what can be done with given technologies, but these ideas cannot be adequately described back to the customer, requirements documenter, or end user. A means of communication that is sometimes used is the development of software prototypes and mockups to explain capabilities and constraints. However, prototypes have not been used frequently with military contracts because the acquisition process does not provide an adequate framework of prototyping.

Sometimes a given project will recognize at inception that there are certain unknowns that need to be explored before a detailed specification can be completed. With such foresight, the communication between the requirements documenter and developer communities is explicitly supported, and trade-off analyses are funded to solve these unknowns. Without such explicit support, the communication between the developers and the requirements documenters, via prototyping or other means, is often omitted.

Most elicitation processes, especially those used within the government, mandate that a set of formal documents be produced. The purpose of the different means of communication pre-

sented above is to produce this set of documents, which ideally will represent a complete, consistent, clear, usable set of requirements for a system. Without addressing whether the documents can achieve this goal, there are still known problems with the process. These problems can be grouped into two classes. First, communication deficiencies can result in certain requirements being missed, others being interpreted differently by different individuals and elicitation communities, and others being overspecified or underspecified. Second, the elicitation process has not been viewed as being part of a larger iterative process in which the requirements evolve with time.

Elicitation should not be viewed as occurring only at the beginning of a development project. During design and implementation, new requirements may be proposed and old ones modified and deleted. These changes are put into effect in the design and code, but the requirements documents are not updated and the elicitation process is not invoked to check whether such changes are really desired by the customer and operator communities.

As the system evolves, the requirements for the existing system are not modified but are only added to, increasing the likelihood of having the new conglomerate of requirements contain inconsistencies. The next section addresses these two classes of problems in more detail.

5.4 Problems with Elicitation

5.4.1 Communication Issues

Requirements elicitation is primarily a communication process. This communication is not a single direction information flow from the requirements source to the requirements analyst, but involves many different communities and bi-directional transfer of information. The information may serve one of many different functions, may be structured or informal, and may involve multiple types of media. These communication issues will be explored further in this section.

Requirements elicitation involves many different parties, and any interaction between these parties is a potential source of requirements. Requirements do not only originate with the operator community and are not derived exclusively from solicited input. For example, a sponsor such as the U.S. Congress may impose a restriction that the software for a specific project will have to work on a hardware platform for a different project. The model with the user as the sole source of all requirements is invalid for most problems, and hence any elicitation tool which assumes this model will have only limited utility.

There are often impediments to communication between one or more communities. One such constraint is a managerial goal to control costs early in a project's life cycle. A poor way to achieve this goal is to speed up the requirements process as much as possible. Such a directive is ill-founded and will result in increased project costs because of the forced restrictions on communication during requirements elicitation. The short-term effect is that initial costs are cut and less paper is produced, i.e., the requirements document is smaller.

Another impediment is the managerial concern with early usability testing. Such testing costs money in terms of developing the tests, using computer and human resources to administer the tests, making the tests available to the users, and allocating users' time to take the tests. Information gained from testing can be critical in communicating requirements for a new system, but unfortunately management often wants to forego early communication with the users in order to continue "building a real system." This argument applies to more costly forms of communication with the users as well, including early prototyping.

Cost is another factor that restricts the communication between the requirements documenters and the operator community during elicitation. Cost also restricts communication between the other communities. For example, it may be useful for the operator community to know what can be implemented with current technology via the developer community. Without explicit funding to create prototypes or send the end users to developer sites to see examples of current technologies, this communication is not promoted and frequently does not take place.

Communication between a requirements documenter and a contractor is often very limited because of fears of cost overruns. If the documenters find a problem with the requirements which they judge to be trivial, e.g., an inconsistency without any known safety or reliability implications, they will not modify the requirements to fix the problem. A modification would involve a contract change, which would increase the project's costs. If the contractors find the requirements too general in some section, they will not rewrite the requirements because they do not receive financial incentives to update the requirements—this is the task of the requirements documenter community, not the developer community, and with military contracts the two communities are typically disjointed. Even if the contractor communicates to the requirements documenter that some of the requirements are too general, the documenter is prone to not update the requirements because of the subsequent increase in project costs due to "contract change." The working group's experience with Navy developments was that little or no communication occurred between the requirements documenters and developers. This point will be returned to when traceability issues are discussed.

There are also legal ramifications restricting communication between the developer community and the requirements documenter community. After a contract is posted but before it is awarded, the requirements documenter does not engage in discussion with developers. Such discussion could prove useful, e.g., it may be that sections of the requirements document are too restrictive and overspecify the system. The contractor could inform the requirements documenter that such a section inhibits the use of the best technology available.

Learning takes place among all of the communities involved in elicitation. For example, in the communication between the requirements documenters and the end users, the requirements documenters learn about desired features for the new system. In addition, these documenters learn about the domain, and the end users learn about the capabilities of delivery technologies. With this experience, the requirements documenters are better able to elicit information from other end users for future systems, e.g., they will be able to conclude more quickly which views expressed by different users refer to the same issue. The documenters have a better

understanding of the requirements process and a more in-depth understanding and appreciation of the domain.

The human resources problem is that the investment made in educating these requirements documenters is often lost. Given that the more experienced a requirements documenter is, the better he or she will be at eliciting requirements, it is prudent to keep documenters. However, many of the military sources experience a high turnover of personnel. Instead of capitalizing on past experience, new people continually have to be trained in the requirements process and problem domain.

A problem associated with human resources is that the requirements elicitation process is continually evolving. Documents explaining the process quickly become out of date. An "organizational memory" is necessary to ensure that both new and longstanding employees share the same referential background and are comfortable with the elicitation process.

The high turnover rate of parties involved with requirements elicitation enhances the importance of requirements traceability. Given a system that is continually evolving over the years, the knowledge about the domain and requirements elicitation work for earlier versions of the system would ideally be part of the requirements documenter's skill set. However, the personnel who elicited requirements for an early version of a system may not be available when elicitation is to be done for a new version. Therefore, this knowledge should be embedded in the requirements, rather than left with personnel who may not be accessible when a new version is being specified.

Likewise, because of the competitive bidding process in the military, it may be that the contractors for a given version of a system are a different set of contractors than those who have worked on a past version of that system. This update work would be greatly enhanced if the rationale behind the requirements were stored in the requirements document, rather than assumed to be part of the later contractor's background.

5.4.2 Traceability Issues

Due to a number of factors, including cost, schedules, and personnel resources, new systems frequently are updates of past systems. This is especially true in the military. For example, one of the working group members is involved with a project which traces back to 1962. Requirements generated for these earlier systems have a life span far beyond that of the initial delivery cycle.

These original requirements are used in whole or in part to specify an updated system. Unfortunately, these original requirements do not evolve cleanly as the system evolves. To cut costs, new needs for an updated system are added to the conglomerate of old requirements for that system, and when taken to the extreme, the old requirements are not changed at all. As a result, consistency problems often arise because the newly added requirements clash with points in the old requirements. This may not be noticed because the old requirements are not scrutinized. With each successive development of a system, the gap between the require-

ments and the implementation widens because each new version introduces an error term. Eventually the conglomerate of requirements for a system that has been through a number of versions becomes so unstructured that the requirements document retains little value.

Much of the current software development involves extending and evolving 10- to 15-year-old systems. Cost, schedule, and personnel constraints prohibit the building of new architectures or new requirements from scratch. An observation was made in the working group that it could take a year to create the paperwork for such a "ground up" proposal, without even considering the effort involved to do all this work from scratch.

As a result, old requirements documents will get reused in future development efforts. Old architectures are not thrown out; "we remodel the rooms, but not the building." Since change is inevitable, e.g., the hardware capabilities will advance over time and change, to motivate the new system development, the requirements document should also be able to evolve over time across different iterations of system development.

Requirements should also evolve over time within the same version, but linkages between the design, code, and the requirements are not maintained. Changes in the design are not maintained in the requirements. Changes in the code are not reflected with appropriate changes in the baseline requirements. Thus, the requirements are out of date even before project completion. Since the requirements do not even adequately mirror the system at the completion of the initial build, they are inadequate to use as a baseline when adding new requirements to specify an upgraded system. In future updates, the code is often consulted for the requirements, or at the very least to get at the *rationale* behind the requirements, because such information is not maintained over time in a requirements document.

There are no financial incentives to evolve the requirements, so this activity is not performed, even when there are known consistency and generality problems with the requirements document. One working group participant recalled a recent system which consisted of hardware and software components. The software components had more constrained safety requirements than the hardware. This inconsistency was discovered during the design phase. Rather than change the requirements to make them consistent, the more stringent software safety requirements were left in because the conflicting requirements were not hazardous. A change in requirements would have involved a contract change which costs money.

Likewise, there may be a requirement that is too broad, e.g., "Thou shalt be able to navigate globally." This requirement may get broken down into testable components in later phases, but the original requirements document will not be updated because of the costs involved.

Requirements are often prioritized implicitly. Explicit prioritization is avoided on some projects because the customer community would then be able to draw the line and indicate that features with higher priority should be implemented and those at lower priority should not. Such partitioning is deceptively simple when a list of requirements is ordered by priority. In actuality, the satisfaction of a higher priority requirement may satisfy very low priority items as well. If rationale were associated with the requirements, it could be a means for deciding which re-

quirements to implement if a system needs to be constrained due to cost, time, or other factors. Rationale could also help decide issues such as whether 47 medium priority items are more important as a whole than as a single high priority item.

Traceability is a very important issue with requirements elicitation because of the extended life span of the requirements documents. There are more general process issues concerning elicitation, which are discussed in the next section.

5.5 General Process Issues

There are many processes involved with requirements elicitation, but they are not well defined and not well linked. As a result, the quality of requirements produced through elicitation varies from project to project. There needs to be a well-defined, repeatable process for requirements elicitation. A single specific way to perform elicitation may not exist, and even if it does it may be too difficult to derive. Therefore, the process should consist of a set of conditions to strive toward, analogous to the Software Engineering Institute's software engineering process work and the definition of software maturity levels.

One effect of having an ill-defined elicitation process is that the level of requirements detail varies greatly from project to project. The A12 Project had a fair amount of documentation. Other Navy projects had very little specification in order to complete the requirements at low cost. Other projects suffered from overspecification. One system contained forty thousand requirements, built up over a series of years. There is a tradeoff involved. Do you give the developer community flexibility to find the optimal solution, or do you specify the system in more detail so you know precisely what to expect? An observation by one of the workshop members was that the Navy tends to underspecify and indicate what cannot be done, while the Air Force tends to overspecify and indicate what has to be done.

There have been project approaches that have created a good requirements document, but at the expense of lengthy review cycles and an excess of paper generated. In many cases these projects were terminated in favor of a process that would get the requirements done more cheaply and quickly. A successful elicitation process has to address the concerns of upper management and be able to justify resource and cost expenditures involved with gathering requirements. The existence of such a process will stabilize the influencing effects of individuals in the different elicitation communities, especially the customer community.

Before tool support can have an effect on requirements elicitation, there is a need to identify what constitutes good practice in this area. The working group cautioned that tools can anchor you to the past if the installed base of tools becomes large and the cost of upgrading the tools becomes prohibitive. Since the area of requirements elicitation is relatively new, any tools supporting elicitation should be evolutionary and extensible.

This section has concentrated on identifying some of the problems which plague requirements elicitation. The next section suggests solutions to some of these problems.

5.6 Suggestions for Improving the Elicitation Process

Elicitation is primarily a communicative process involving customers, requirements documenters, developers, and operators. Strategies for improving communications between these communities are suggested, which in turn will improve the elicitation process. The suggestions outlined in this section are presented in the same order as the problems were presented in the previous section, with communication issues first, followed by traceability issues, and then more general issues. It is important to recognize, however, that some suggestions are global in nature and may apply to more than one category.

5.6.1 Communication Issues

Many different communities, including customers, requirements documenters, developers, and operators contribute to requirements elicitation. Any process proposed for elicitation must consider these different groups, and elicitation tools have to support this general model. In the past, too much focus has been placed on the communication between the requirements documenter and the end user (operator), to the exclusion of the communication between the other groups.

In order to foster communication between these groups, managerial constraints have to be removed. This primarily involves educating management on the model and the usefulness of communication between groups early in a project life cycle. For example, management should be made aware of the long-term cost and quality improvements offered through the use of early prototypes to communicate between the developer, requirements documenter, and operator communities.

As another example, management should be made aware of the long-term cost and quality improvements of usability testing in communicating information between the operator community and requirements documenter community. When Florida Light and Power went to a TQM philosophy, they asked the users what they wanted in terms of service; they found out that receiving continuous power was of a much higher priority than decreasing service costs. This result was not anticipated by the company, but it led to improved requirements for future systems and greater consumer satisfaction. This example illustrates that the users do indeed know what they want, and that improved communication with the operator community can lead to better system requirements.

Communication between the different communities can also be improved by removing the contractual, legal, and financial barriers between these groups, especially between the requirements documenter and the developer communities. If a developer notices a problem with a requirement after the contract has been posted but before it has been awarded, then the developer should be free to tell the requirements documenter. If necessary, the acquisition procedures should be changed to provide such a "requirements review" period after the contract has been posted to promote communication between the developer and the requirements documenter.

If a problem is noticed by the requirements documenter after the contract has been awarded, the documenter should be free to communicate this problem to the developer community without having to suffer unwarranted cost increases. The developer community should not unjustly take advantage of this communication and increase costs.

The developer community should be receiving some financial incentive or contractual obligation to maintain the requirements document as well as create the design and implementation. Such maintenance would promote the traceability of the code back to the requirements and allow for requirements reuse. (This will be returned to when traceability is discussed.) When updates to the requirements document are necessary, such changes should be communicated back to the requirements documenter community (and the other communities if applicable), so that the updates can be verified to be in the best interests of all the communities involved.

A model which shows that the requirements can be frozen in a fixed document is unrealistic. The requirements documenter may find that a requirement is missing or inconsistent. The developer may find that a requirement is too general. The operator or customer communities may fill in a requirement that was missed because of a lack of communication earlier in the process. Whenever such an update occurs, however, it should be propagated through all the communities, i.e., elicitation is an iterative process.

5.6.2 Traceability Issues

Another key component to improving elicitation is to improve the organization of the information passing between the different communities. This point becomes more important because the amount of information to organize will increase as a result of improved communication. Because the personnel involved with requirements elicitation will likely change over the life of a single system development and over the tenure of the upgrade developments for that system, the information organization should not rest with individuals but should be integrated with the requirements document. Therefore it is important that the requirements document contain the rationale behind the requirements and, ideally, the domain knowledge used in gathering these requirements.

A system rarely undergoes just a single iteration, so it becomes vitally important to store the rationale associated with the requirements in the requirements document. The developer community working on an upgrade may not have worked on prior versions, and the personnel that documented the original requirements may no longer be available. If the rationale is not maintained with the requirements, two problems frequently occur: (1) The code is used to figure out the rationale behind the requirements, a time-intensive and error-prone process; and (2) new requirements are added for an upgrade which are in conflict with the rationale for some requirements in the existing system. Since this rationale is not maintained, this conflict is not noticed.

One technique for keeping track of this rationale is the Issue-Based Information System (IBIS) approach, described in detail in the CSCW October 1990 Proceedings. The IBIS method is used to capture dialogue information by keeping track of the issues being discussed, the po-

sitions taken on these issues, and the arguments supporting or objecting to positions. Such a technique is easy to learn and use, and organizes the information well. However, it does not support automated checking of attributes like consistency because there are not enough formalisms used in the method, and more comprehensive techniques may not get used because of their increased complexity.

In addition to tracking rationale, it would be advantageous to create domain models from the information gathered during elicitation so that all the involved communities understand what the inherently imperfect natural language description of the requirements actually mean. Such domain models could then be reused in future iterations of the system, promoting traceability and a quicker specification time.

5.6.3 General Process Issues

Although a wealth of information has to be stored during elicitation, it is often difficult to access information and to present the appropriate information to different communities. Therefore, improving the storage, organization, and retrieval of the information used in deriving system requirements is an important way to improve requirements elicitation. The previous sections highlighted the importance of improving communication between the communities involved in elicitation and the importance of keeping track of rationale and domain knowledge to enhance the traceability and modifiability of the requirements. This section concludes with some suggestions on what should be done to improve the elicitation process in general.

Rather than give the software development community a single specific elicitation method to follow, they should be encouraged to decide for themselves which approach is best for a given system in a given domain. They should be supported with a set of guidelines indicating the best practices possible for elicitation. This document should include the more specific points mentioned earlier concerning communications between different communities, keeping track of requirements rationale, and promoting requirements evolution. It also should include points which the working group did not detail, but which they recognized as being useful and important for requirements elicitation.

Given the existence of this document, tools which support the cited practices should be created and/or improved, and supplied to the elicitation communities. These tools should provide support for the model presented in this report, as opposed to just supporting communication between the requirements documenter and a user. Included in this tool set would be techniques to improve the capture and organization of information in meetings, help with the negotiation process, and support for the linking and tracing of requirements.

6 Requirements Engineering Techniques and Tools

6.1 Participants

John Gay	U.S. Army Computer Science School
William Gilmore	Software Engineering Institute
Fernando Naveda	University of Scranton
James Palmer	George Mason University
Colin Potts	MCC
James Sidoran	Rome Laboratory
Dennis Smith (Chair)	Software Engineering Institute
David Wood (Scribe)	Software Engineering Institute

6.2 Introduction

This report summarizes the activities of the Requirements Techniques and Tools working group (RTTWG) session of the Requirements Engineering and Analysis Workshop. The goal of the RTTWG was to examine the major functions of requirements engineering, identify techniques and tools appropriate for each function, and ground the techniques and tools in specific experiences of the participants with project development.

Although there is general agreement about the crucial importance of requirements engineering, there are a wide variety of techniques and tools available. The intent of the RTTWG was to make generalizations about requirements techniques and tools as appropriate, and to identify major gaps or untested areas in current techniques and tools. The goal of the discussions was not to validate or invalidate specific methods and tools, but rather to provide a context for the type of domain, organization, or project for which specific techniques and tools are appropriate or inappropriate. The hope was that the participants would bring a rich set of experiences to this activity.

A planned result of the RTTWG session was a matrix of requirements engineering sub-processes, techniques, tools, and project-specific experience summaries with the intent to provide a meaningful starting point for future research activities.

6.3 Discussion

As a point of departure, the session moderator introduced the findings of the 1989 workshop, "Requirements Engineering and Rapid Prototyping Workshop," sponsored by CECOM. The contention of the CECOM workshop was that the requirements engineering process consists of six generic subprocesses:

1. **Context Analysis**. The analysis of problem space and application domain.
2. **Objectives Analysis.** The analysis of the solution space and system objectives for lifetime use.
3. **Requirements Determination.** The specification of characteristics the system must meet to satisfy user needs.
4. **Requirements Analysis.** The analysis of expressed requirements, including related refinement, elaboration, and correction.
5. **Synthesis**. The formation of a cohesive specification from the detailed analysis, involving the integration of partitioned analyses occurring due to problem complexity and breadth.
6. **Validation**. The assurance that the expressed requirements match real user needs and constraints.

It was the original goal of the RTTWG to examine each of the subprocesses and their related techniques and tools as identified by CECOM. However, logistical constraints did not permit sufficient time to consider the full breadth of discussion topics. Early in the session it was noted that subprocesses 3 through 6 are more mature, while comparatively less is known about subprocesses 1 and 2. The group decided to focus on the early subprocesses in order to add more richness to those areas. In the end, most of the discussion centered on subprocess 1, context analysis, which deals with the analysis of the problem space and application domain and is concerned only with the description of problems, not the description of solutions.

The RTTWG decided to examine the meaning of context analysis, summarize examples of current approaches (or potential approaches) for carrying out context analysis, and attempt to identify gaps in existing technology. Time was allotted for a final brainstorming session in an attempt to capture as much knowledge and opinion as possible regarding each of the subprocesses.

6.4 Meaning of Context Analysis

To facilitate understanding among the RTTWG participants and to lend continuity to the discussions, the group considered the definition of the term context analysis before proceeding with other activities. The CECOM workshop identified context analysis as involving "analysis of the problem space and application domain of a potential system to be developed. It deals with description of problems only, not solutions." Four specific activities were identified as comprising context analysis: identification of problem space boundaries; needs identification; application modeling; and postulating solutions.

It should be emphasized that context analysis refers to setting the context of problems at an abstract level. The word "analysis" is meant in the sense of everyday examination of conditions and problems within the operational environment. It does not refer to a specific, formal analytical process. However, the working group suggested that context analysis be performed intentionally on an ongoing basis for its own purposes, independent of analyses carried out for particular systems. If an ongoing context analysis "mode of thinking" becomes institutionalized,

the recognition and identification of existing problem areas and shortfalls will occur more naturally.

6.5 Summary of Current or Potential Approaches

Many approaches to context analysis were discussed by the group, including:

- Causal Trees
- SWOT (Strength, Weakness, Opportunity, Threat) Analysis
- IBIS (Issue-Based Information Systems)
- CSCW (Computer Supported Cooperative Work)
- IRS (Information Requirements Study)
- Information Engineering
- Conceptual Modeling Using Expert System / Knowledge Base
- CAPS (Computer Aided Prototyping System)/ PSDL

It is notable that few of these approaches are intrinsically software-oriented. The group concluded that context analysis for software requirements can benefit greatly by borrowing techniques used in other fields. Several of these approaches were discussed in greater detail to capture the experiences of the working group participants. These detailed discussions are summarized in Table 1.

6.6 Gaps in Existing Technology

Following the discussion of existing and potential approaches to the problems of context analysis, the RTTWG considered gaps and weaknesses in existing technology that should be examined as a part of future research endeavors. Bill Gilmore, co-editor of the Requirements Engineering Methodology, Tools, and Languages working group proceedings of the CECOM workshop, joined the RTTWG briefly to discuss items that he considered to be unfinished business from the CECOM workshop. In particular, he pointed out the need to identify temporal and informational connections between the requirements engineering subprocesses to provide continuity and interconnection. A related issue is that language should provide the continuity. The ensuing discussion among the RTTWG participants confirmed a general consensus in this direction.

The unifying language need not, and probably should not be formal in its user interface. Although the closer one gets to the ultimate problem space, the more structured and formal the language becomes, the language should serve the process and not drive it. This is particularly important in the earlier phases of requirements engineering: while it is desirable to achieve a formalism, it is critical that the actual users define the requirements. It is not realistic to expect the users to learn complex formalisms.

The languages of the first two subprocesses must allow for communication among all stakeholders, but also must be sufficient to comprehensively express real needs. This language should be an underlying core grammar that captures the essence of the requirements. Multiple, less primitive languages can be built upon the core for use by different stakeholders. All of the languages would utilize the same information base. A suggested area for research is the applicability of an object-orientation to this underlying core language and/or its information base. In any case, a rich domain-oriented language is needed in order to capture real needs.

Several other gaps were identified in current technology. Detailed information for each of these gaps were captured in template formats, which are summarized in the following section.

6.7 Identified Research Topics

As part of a final brainstorming session, the RTTWG captured knowledge and opinions regarding potential topics for future research in requirements engineering. These topics cover the full breadth of requirements engineering and are not restricted to context analysis.

Identified research areas are listed below. Time constraints precluded detailed elaboration of these topic areas; however, they should provide fertile ground for future refinement and investigation.

- Examine group skills and their implications on requirements elicitation; perhaps develop a recommended composition of group skills needed for each phase of requirements engineering.
- Examine incremental formalization of requirements and develop a supportive conceptual model; do for the 1990s what the A7 approach did for the 1970s and 1980s.
- Examine meaning of validation via prototyping; consider how one actually validates a requirement using a prototype; develop an appropriate environment model/test case generation method.
- Develop better predictor for cost and risk than "lines of code", e.g., something along the lines of function points or complexity metrics, with a specific aim of estimating cost and risk.
- Examine qualitative or approximate simulation for validation to eliminate infeasible requirements.
- Develop prototyping techniques that are supportive of functional validation rather than just user-interface validation.
- Examine prototyping with multiple abstraction levels.
- Examine decision-making strategies such as JAD; involves getting decision makers together with a structured agenda, in an environment where they are able to define requirements and get buy-in through using prototypes and other mechanisms.

In addition, several topic areas were discussed in more depth as time allowed. Table 2 outlines information captured from detailed discussions of specific research concerns.

Table 1 Current and Potential Approaches

CAPS (Computer Aided Prototyping System)	**Statement.** Developer-oriented system with sophisticated graphic interface. Allows developer to construct a prototype based on libraries of reusable objects. Can browse, retrieve, and tailor objects. User interface includes syntax-driven input facility to create bridges between levels of abstraction from natural language to abstract language descriptions. Can show objects associated with a requirement or requirements associated with an object. Built around PSDL (prototype system description language), a procedural high-level description language designed for hard real-time systems. Includes expert system that generates natural language (structured English) from PSDL statements. **Actual Use.** Used in specifications of hypothermia treatment system for tumors. It is an ongoing development (CAPS), due for delivery in September1991. **Rationale.** Addresses feasibility of construction of a solution once a problem has been identified. **Positives.** Given a rich software base, it allows very quick creation of prototypes. It automatically generates Ada code from the prototype, and allows the developer to keep a highly structured history of the prototyping process. **Negatives.** Not object-oriented at all; entirely functionally-oriented viewpoint. **Next Steps.** n/a **Reference.** Luqi (Naval Post-Graduate School, Monterey, CA).
Conceptual Modeling (using expert system/knowledge base technology)	**Statement.** A method and tool that aids in the analysis of complex systems for the purpose of identifying, representing, documenting, and validating (understanding) system behaviors, during prerequirements phases (C.E., D.V.) of the system development life cycle. Expert system shell capabilities (rule definition, object-oriented representation, functions, procedures, simulation) are utilized to capture and execute system behaviors. Icon definition capability. **Actual Use.** Partial air defense model, in early prototype stage. Similar approach used for health care executive decision making. **Rational.** Address concepts feasibility issues, high-level requirements early in the development life cycle. Statement of need level. **Positives.** Supports animation, visualization of needs. **Negatives.** Difficult, ambitious problem. Mature technologies may not exist just yet. **Next Steps.** Further research **Reference.** James Sidoran, Rome Laboratory
CSCW (Computer Supported Cooperative Work)	**Statement.** Support environment for group decision meetings; can be used to facilitate requirements engineering. Uses trained facilitator, several possible group paradigms possible. **Actual Use.** Decision support, urban mass transportation administration, examination of existing requirements sets, e.g., Howitzer Improvement Program, couple others. **(Continued on next page)**

Table 1 Continued

CSCW (Computer Supported Cooperative Work)	**Rationale.** Group decision support system. Need for capture of multi media information. Ability to make non-sequential linkages (hyper-technology). Concept prototyping. Presentation. **Positives**. Supports *context* and *objectives* analysis as defined by CECOM. May use a facilitator. **Negatives.** Expensive **Next Steps**. n/a **Reference**. James Palmer
IBIS (Issue Based Information Systems)	**Statement**. Problem formulation method based on issues, positions, and arguments. **Actual Use**. Used extensively manually for architectural, urban planning, health care planning. Used for organizational decision-making within MCC. Used for product development at NCR (software/systems). **Rational**. Rhetorical model. Multi-stakeholder problem formulation/dialog supported. **Positives.** Simple model, easy to use and apply. **Negatives**. Not as simple as it looks to use properly. Doesn't help with decision making directly, other than clarifying issues. Networks are fixed and difficult to reconfigure. **Next Steps**. This type of approach can serve as example for future approaches. We need something like this that is specifically for requirements. Further "group technology" is important. **Reference**. K. C. Burgess-Yakemovic, NCR Human Interface Technology
Information Engineering	**Statement.** A process that identifies information requirements, organizational goals, and organization processes. Provides basis for logical architectures from these building blocks. **Actual Use**. Primarily used in business, but applicable to DoD organizations (has had some use, e.g., Gunter AFB). **Rationale**. Requires top management support; cannot be done from grassroots level. Requires involvement of primary proponents. A disciplined approach for identifying information system needs followed by systematic development discipline. **Positives**. Must be driven by proponent organization (not driven by computer/MIS people). Has the potential to provide an enterprise or organizational view which is problem independent. Has potential to eliminate redundancy in databases and applications. Drives planning from organizational goals, so focus is on "can be," not "what is." Integrates entire life cycle. **(Continued on next page)**

Table 1 Continued

Information Engineering	**Negatives**. Requires substantial amount of top management time; must be sold as being worth their time, not something that can be delegated to computer organization. Requires different mindset from traditional application development; difficult time focusing on abstract issues rather than becoming bogged down in minute detail. Tends to require proponents to think in terms of *data* rather than *information*. Time-consuming: there is a need to manage user expectations in that it can take a year or more for tangible results. People can be absorbed in the techniques rather than the substance of what is going on. **Next Step**. n/a **Reference**. James Martin books. Book by Clive Finklestein.
IRS (Information Requirements Study)	**Statement**. U.S. Army standard, derived from IBM's BSP (Business Systems Planning). A process that maps information requirements against the organization processes that make use of that information. **Actual Use.** Business systems development. In the Army, used at Ft. Sill for Standard Installation Support Modules (ISMs). Modeled information requirements of that installation with the goal of discovering opportunities for information sharing and consistency of information storage. **Rationale**. Requires top management support; cannot be done from grassroots level. Requires involvement of primary proponents. **Positives**. Must be driven by proponent organization (not driven by computer/MIS people). Has the potential to provide an enterprise or organizational view which is problem independent. Has potential to eliminate redundancy in databases and applications. **Negatives**. Requires substantial amount of top management time; must be sold as being worth their time, not something that can be delegated to computer organization. Requires different mindset from traditional application development; difficult time focusing on abstract issues rather than becoming bogged down in minute detail. Tends to require proponents to think in terms of data rather information. Treats the data as separate and distinct from the processes, rather than as unified objects. Result is "stovepipe" process-oriented systems. This isn't required by the approach, but tends to be the result if not used carefully. Focus is on "what is" rather than "what can be." **Next Steps**. n/a **Reference**. Army regulation 25 series (starting with AR 25-1). IBM's BSP manual.

Table 2 Research Topics

PROBLEM	CONTEXT	REQUIREMENTS / SUGGESTIONS
Support language for requirements engineering	There is a need to develop a language for supporting the requirements engineering process. The language should provide for both generic information capture and seamless integration with domain-specific information capture.	**Requirements.** The language must be subservient to the requirements process, and not drive it ("requirements by stealth"). Multiple view capability from a common information base. Must be targeted toward (usable by) both user and proponent. **Suggestions**. Consider object-oriented approach, as domain structure may be more stable than functional requirements. Address issues of transition between subprocesses. Examine domain-specific syntax requirements. A good candidate domain is C3I. Take a close look at work being done at George Mason University in this area.
Participative requirements development	There is a need to consider the sophistication and computer literacy of classes of users and the ramifications upon the ability and desire to use existing technology.	**Requirements.** Define mechanisms for getting stakeholders involved across phases. Particularly in defense systems, users are at "arm's length," and are not sufficiently involved in downstream process. **Suggestions**. Examine work by Clegg, et al, *People and Computers*, Ellis Horwood publishers; and Rouse, William, *Human Design*, Wiley and Sons Systems Engineering Series, 1991.
Examine interviewing techniques	There is a need to develop an organized approach to conducting elicitation interviews.	**Requirements.** Although a disciplined approach is desired, many or most users do not operate in a disciplined mode (in the engineering sense). **Suggestions**. Examine work by Gause and Wenberg, *Exploring Requirements: Quality Before Design*, Dorset House Publishers, 1989.
Prioritization of requirements	Determine the order of desirability of requirements in accordance with the needs of end-users.	**Requirements**. Although this problem needs to be recognized, it is probably not technologically addressable. There must be a viewpoint authority in place to make it work. Prioritization cannot occur in isolation (i.e., each requirement cannot be treated in isolation from other requirements). **Suggestions**. Examine work by Gilb, Tom, *Software Engineering Management*.

Table 2 Continued

PROBLEM	CONTEXT	REQUIREMENTS / SUGGESTIONS
Investigate object orientation for requirements engineering	There is some consensus among the RTTWG members that an object-orientated approach has promise of building a common information base that can be used throughout the entire requirements engineering and analysis process.	**Requirements.** Develop a real object-oriented requirements analysis approach. Existing approaches are not adequate. Domain structure is much more stable than functional requirements; allows people to think in real-world terms; support evolutionary requirements; favors reuse. **Suggestions**. Examine work by Wirfbrock, et al, *Object-oriented Software Construction*, Prentice Hall.

6.8 Summary

The CECOM Workshop identified six subprocesses of the Requirements Engineering Process. Due to time constraints, our working group focused on the least mature of the six subprocesses, Context Analysis, in an effort to add richness to that area. The group discussed the meaning of Context Analysis, summarized examples of current or possible approaches for Context Analysis, and identified gaps and research topics in this area.

Many approaches to Context Analysis were discussed, several of which were captured in template format, including rationale, strengths, weaknesses, and references for further information. It is notable that few of the identified approaches are intrinsically software-oriented. The group concluded that Context Analysis for software requirements can benefit greatly by borrowing from techniques used in other fields.

A number of research topics were identified by the working group to address gaps in the requirements engineering process. The identified topics included: examination of requirements engineering group skills, development of interviewing techniques, prioritization of requirements, incremental requirements formalization, development of prototyping techniques, and examination of object-orientation. Several of the identified research topics were captured in greater detail in template format, identifying needs, context, requirements, and suggested solutions.

A continuing theme which seemed to form the underlying essence of the problems addressed by the working group was a unification and interconnection between the requirements engineering subprocesses. This unification will provide a mechanism for requirements tracing and validation. It was generally agreed that some language should form the basis of continuity among the subprocesses, but that continuity should be established in the form of a common underlying grammar rather than a forced explicit representation. The grammar should support natural language expression at the Context Analysis stage, yet support increasing levels of formality as the requirements engineering process progresses. It is this mechanism of unification that will allow users and developers of diverse backgrounds to reconcile their concepts of the system under development in a disciplined and progressive specific fashion.

References

[BROOKS87] Brooks, F. "No Silver Bullet: Essence and Accidents of Software Engineering." *IEEE Computer* 20 (4), April 1987.

[CECOM89] *Proceedings of the Requirements Engineering and Rapid Prototyping Workshop.* Center for Software Engineering, U.S. Army Communications-Electronics Command, November 14-16, 1989.

[DSMC87] "Evolutionary Acquisition: An Alternative Strategy for Acquiring Command and Control (C2) Systems," The Defense Systems Management College, Fort Belvoir, VA, March 1987.

[SIO89] "Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation," Staff Study by the Subcommittee on Investigation and Oversight transmitted to the Committee on Science, Space, and Technology, U.S. House of Representatives, September 1989.

Appendix A Position Papers

The attached position papers were submitted by workshop participants:

- *CAPS as a Requirements Engineering Tool*
- *Requirements Specification of Hard Real-Time Systems: Experience with a Language and a Verifier*
- *Acquisition Model for the Capture and Management of Requirements for Battlefield Software Systems*
- *Requirements Process Analysis*
- *Unstable Requirements*
- *Requirements Engineering Processes and Products*
- *The Integrated Requirements Process*
- *Seven (Plus or Minus Two) Challenges for Requirements Analysis Research*
- *Requirements Techniques and Tools*
- *Requirements Elicitation Working Group*
- *A Computer Supported Cooperative Work Environment for Requirements Engineering and Analysis*
- *An Informal Approach to Developing an Environment for Requirements Capture and Refinement*

CAPS as a Requirements Engineering Tool

Luqi, R. Steigerwald, G. Hughes, F. Naveda, V. Berzins

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

ABSTRACT

The process of determining user requirements for software systems is often plagued with uncertainty, ambiguity, and inconsistency. Rapid prototyping offers an iterative approach to requirements engineering to alleviate the problems inherent in the process. CAPS (the Computer Aided Prototyping System) has been built to help software engineers rapidly construct software prototypes of proposed software systems. We describe how CAPS as a prototyping tool helps firm up software requirements through iterative negotiations between customers and designers via examination of executable prototypes.

1. Introduction

A major problem with the traditional waterfall lifecycle approach is the lack of any guarantee the resulting product will meet the customer's needs. In most cases the blame falls on the requirements phase of the lifecycle. Yourdon [Yourdon] cites studies that indicate 50% of errors or changes required in a delivered software product and 75% of the total cost of error removal are the results of inadequate, incorrect, or unstated requirements specifications. Often users will be able to indicate the true requirements only by observing the operation of the system. Unfortunately, the traditional life cycle yields executable programs too late in the software engineering process, at a point where major change is prohibitively expensive [Boar].

To alleviate the problems inherent in requirements determination for large, parallel, distributed, real-time, or knowledge-based systems, current research suggests a revised software development life cycle based on rapid prototyping [Berzins88, Berztiss, Tanik]. As a software methodology, rapid prototyping provides the user with increasingly refined systems to test and the designer with ever better user feedback between each refinement. The result is more user involvement and ownership throughout the development/specification process, and consequently better engineered software [Ng].

2. The Computer Aided Prototyping System (CAPS)

The problem with requirements engineering is amplified in the case of hard real-time systems, where the potential for inconsistencies is greater [Beam, Boyes, NGCR, Stankovic]. One of the major differences between a real-time system and a conventional system is required precision and accuracy of the application software. The response time of each individual operation may be a significant aspect of the associated requirements, especially for operations whose purpose is to maintain the state of some external system within a specified region. These response times, or deadlines, must be met or the system will fail to function, possibly with catastrophic consequences. These requirements are difficult for the user to provide and for the analysts to determine. Toward this end, an integrated set of software engineering tools, the

Computer Aided Prototyping System [Luqi88a], has been designed to support quick prototyping of such complex systems by using easy to understand visual graphics [PDW] mapped to a tight specification language, which in turn automatically generates executable Ada [Booch, Gonzalez] code. The main components of CAPS are the prototype system description language (PSDL), user interface, software database system, and execution support system (see Fig. 1).

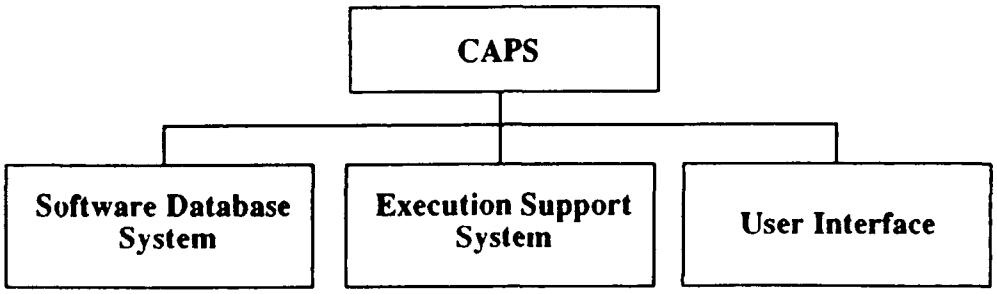


Fig. 1 High Level Structure of CAPS

2.1 Prototype System Description Language (PSDL)

The prototype system description language (PSDL) [Luqi88b] is the key component of CAPS. It serves as an executable prototyping language at a specification or design level and has special features for real-time system design. The PSDL model is based on data flow under real-time constraints and uses an enhanced data flow diagram that includes non-procedural control and timing constraints.

2.2 User Interface

The graphic editor, in the User Interface, is a tool which permits the user/software engineer to construct a prototype for the intended system using graphical objects to represent the system [Linton, TAE]. These objects include operators, inputs, outputs, data flows, and operator loops. The syntax directed editor is used by the user/software engineer to enter additional annotations to the graphics. A browser allows the analyst to view reusable components in the software base. An expert system provides the capability to generate English text descriptions of PSDL specifications. Together, these tools facilitate common understanding of PSDL components by users and software engineers alike, thereby reducing design errors.

2.3 Software Database System

The software database system provides reusable software components for realizing given functional (PSDL) specifications, and consists of a design database, software base, and software design management system.

The design database [Nestor] contains PSDL prototype descriptions for all software projects developed using CAPS. The software base contains PSDL descriptions and implementations for all reusable software components developed using CAPS. Prototyping with the software base speeds up evolution by providing many different versions of commonly used components [Steigerwald], making it easier to try out alternative designs. The software design management system manages and retrieves the versions, refinements and alternatives of the

prototypes in the design database and the reusable components in the software base.

2.4 Execution Support System

The execution support system [Borison] contains a translator, a static scheduler, a dynamic scheduler, and a debugger. The translator generates code that binds together the reusable components extracted from the software base. Its main functions are to implement data streams, control constraints, and timers. The static scheduler allocates time slots for operators with real time constraints before execution begins. If the allocator succeeds, all operators are guaranteed to meet their deadlines even with the worst case execution times. If the static scheduler fails to find a valid schedule, it provides diagnostic information useful for determining the cause of the difficulty and whether or not the difficulty can be solved by adding more processors. As execution proceeds, the dynamic scheduler invokes operators without real-time constraints in the time slots not used by operators with real-time constraints [Mok]. The debugger allows the designer to interact with the execution support system. The debugger has facilities for initiating the execution of a prototype, displaying execution results or tracing information of the execution, and gathering statistics about a prototype's behavior and performance.

3. CAPS as a Requirements Engineering Tool

3.1 Prototyping

The Computer Aided Prototyping System (CAPS) is used to create software prototypes, which are mechanically processable and executable descriptions of simplified models of proposed software systems. It is also used to modify these models frequently in an iterative prototype evolution process for the purpose of firming up the requirements. Fig. 2 illustrates the prototyping process which consists of two stages: prototype construction and code generation [Luqi89].

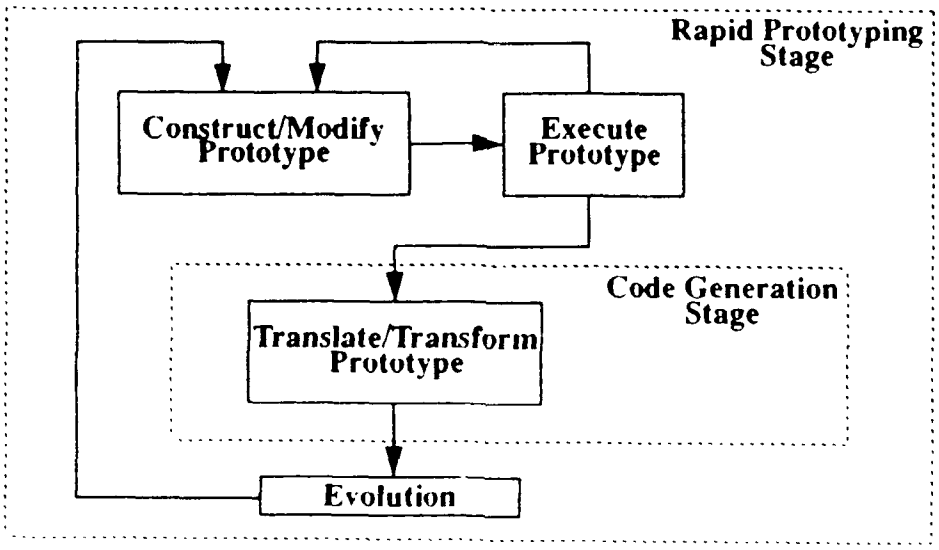


Fig. 2 Rapid Prototyping Process

Prototype construction is an iterative process that starts out with the user defining the requirements for the critical aspects of the envisioned system. Based on these requirements, the

designer then constructs a model or prototype of the system in a high-level, prototype description language and examines the execution of this prototype with the user. If the prototype fails to execute properly, the user then redefines the requirements and the prototype is modified accordingly. This process continues until the user determines that the prototype successfully meets the critical aspects of the envisioned system. Following this validation, the designer uses the validated requirements as a basis for the design of the production software.

The code generation stage focuses on transforming and augmenting the prototype to generate the production code. Prototypes are built to gain information to guide analysis and design, and support automatic generation of the production code.

To create production code from a prototype, it may be necessary to clean up the decomposition, add missing functions, and optimize performance. Prototypes go through many changes in the prototype construction stage, so that the structure of the final version may partially reflect past versions of the requirements that were proposed and rejected. Once the requirements and the desired behavior for the prototype have stabilized, it is useful to transform the structure of the prototype to simplify the decomposition and to remove features that are no longer supported by the final version of the requirements.

A prototype may not implement all of the functions of the proposed system, since the prototyping effort is focused in the aspects of the requirements that are unknown or uncertain. After the requirements have stabilized, the design and the structure of the prototype must be augmented to account for these additional functions. These augmentations can be expressed in the prototyping language to provide an early check on the adequacy of the final version of the system structure.

A prototype may not meet all of the performance requirements, or may operate in the same hardware and software environments as the proposed system. The structure of the prototype may have to be transformed to optimize its performance and to account for differences between the host environment for the prototype and the operating environment for the proposed system. It is desirable to record the desired transformations as annotations on the prototype, and to generate the transformed decomposition automatically based on the annotations. Such an approach preserves the structure of the prototype prior to optimization, so that a version of the prototype with this structure can help to evaluate system changes that are proposed after the system is placed in production use. The unoptimized version of the prototype is better suited for modification because the optimization transformations generally complicate the structure of the design and destroy the independence of its parts, thus making future modifications more difficult. This approach may provide the benefits of rapid prototyping in both the requirements analysis and system maintenance activities.

3.2 Domain Specificity and Requirements Traceability

Using CAPS to engineer requirements offers clear advantages over determining requirements manually. The prototype system description language is focussed on the domain of hard real-time systems and as such offers a common baseline from which users and software engineers describe requirements. Defining requirements in a *domain specific* language results in more efficiency and fewer errors because it constrains the way users and engineers can describe a particular requirement. In addition, the interpretations of requirements stated in a domain specific language such as PSDL are unambiguous, whereas requirements stated in English are often misunderstood.

In most software engineering efforts requirements are volatile, changing often over the course of the software development. Requirements traceability is essential to accurately map changed requirements into the implementation. CAPS offers basic requirements traceability through the "by requirements" statement in the PSDL grammar. This statement allows software engineers to associate actual requirements with the definitions of module interfaces and constraints by annotating the interface or constraint definition with an identifier. This method allows engineers using the design database to readily locate the portions of code that implement a particular requirement and make the appropriate changes. This feature offers substantial savings over manual methods of requirements tracing.

3.3 Requirements Engineering

The requirements for a software system are expressed at different levels of abstraction and with different degrees of formality. The highest level requirements are usually informal and imprecise, but they are understood best by the customers. The lower levels are more technical and more precise, are better suited for the needs of the system analysts and designers, but they are further removed from the users' experiences and less well understood by the customers. Because of the differences in the kinds of descriptions needed by customers and developers, it is not likely that any single representation for requirements can be the "best" one for supporting the entire prototyping process.

During the process of stabilizing the requirements via prototyping, it is necessary to repeatedly move from high-level requirements to details of system behavior, and from system behavior back to high-level requirements. The prototype designers must guess the intentions of the customers based on their informal statements, and embody their vision in a prototype design that can be demonstrated to the users. This process is imperfect, and the demonstrated behavior will help the customers identify differences between what they need and how the analysts interpreted their requests. When a bug in the system behavior is discovered, it must be traced back to the requirements to identify the specific guesses proposed by the analysts that are inaccurate. After the faulty decisions have been identified and new versions have been proposed, it is necessary to trace the effects of the change back down the refinement structure to find the parts of the prototype design that are affected, so that they can be adjusted and the next approximation to the requirements can be demonstrated.

In the context of prototyping, the requirements are used as a means for bridging between the informal terms in which users and customers communicate and the formal structures comprising a prototype. We believe that a useful representation for this information is a hierarchical goal structure, where informal customer goals are refined and defined by several levels of increasingly formal and precise subgoals, with different notations used at different levels. We expect natural language to be used at the highest levels, and the prototyping language to be used at the most detailed levels, with mixtures and possibly several additional notations appearing in the intermediate levels.

The subgoals of a goal in the hierarchy are proposed interpretations for the informal parent goals. We adopt the convention that a parent goal is met whenever all of its subgoals are met. The layers of the subgoal structure correspond to decisions about proposed system behavior and how it can be packaged and presented to users. The most specific subgoals at the leaf nodes of the hierarchy are tied directly to elements of the prototype design.

We are currently exploring guidelines for organizing such a subgoal hierarchy and design

database structures to provide automated support for maintaining and traversing this hierarchy, for recording past configurations of the requirements and prototype, for keeping track of the change history and the rationale for the requirements evolution that occurs during the prototyping process, and for finding the parts of this structure that are relevant for each of the tasks performed by the designers and analysts.

4. Conclusion

Rapid prototyping offers an iterative approach to requirements engineering to alleviate the problems of uncertainty, ambiguity, and inconsistency inherent in the process. CAPS (the Computer Aided Prototyping System) has been built to help software engineers rapidly construct software prototypes of proposed software systems. CAPS helps firm up software requirements through iterative negotiations between customers and designers via examination of executable prototypes. Using a prototype system description language enables engineers and users to quickly focus on the pertinent requirements of their system resulting in increased efficiency and fewer requirements errors.

REFERENCES

[Beam] Beam, W. R., Command, Control, and Communications Engineering, McGraw-Hill, 1989.

[Berzins88] Berzins, V., and Luqi, "Rapidly Prototyping Real-Time Systems", *IEEE Software*, September 1988.

[Berzins91] Berzins, V., and Luqi, Software Engineering with Abstractions, Addison-Wesley, 1991.

[Berztiss] Berztiss, A., "The Specification and Prototyping Language SF", Report 78, Systems Development and Artificial Intelligence Laboratory, Department of Computer and Systems Science, Stockholm University, 1990.

[Boar] Boar, B. H., Application Prototyping: A Requirements Definition Strategy for the 80's, John Wiley and Sons, Inc., 1984.

[Booch] Booch, G., Software Engineering With Ada, Benjamin/Cummings Publishing Company, Inc., 1987.

[Borison] Borison, E., "Program Changes and Cost of Selective Recompilation", Technical Report CMU-CS-89-205, Computer Science Department, Carnegie-Mellon University, July 1989.

[Boyes] Boyes, J. and Andriole, S., Principles of Command & Control, AFCEA International Press, 1987.

[Gonzalez] Gonzalez, D. W., Ada Programmer's Handbook and Language Reference Manual, Benjamin-Cummings, 1991.

[Linton] Linton, M. A., Vlissides, J. M., and Calder P. R., "Composing User Interfaces with InterViews", *IEEE Computer*, February 1989.

[Luqi88a] Luqi, and Ketabchi, M., "A Computer-Aided Prototyping System", *IEEE Transactions on Software Engineering*, October 1988.

[Luqi88b] Luqi, Berzins, V., and Yeh, R., "A Prototyping Language for Real-Time Software", *IEEE Transactions on Software Engineering*, October 1988.

[Luqi89] Luqi, "Software Evolution Through Rapid Prototyping", *IEEE Computer*, May 1989.

[Mok] Mok, A., "A Graph Based Computational Model for Real-Time Systems", Proceedings of the IEEE International Conference on Parallel Processing, Pennsylvania State University, 1985.

[Nestor] Nestor, J., "Toward a Persistent Object Base", in Advanced Programming Environments, vol. 244, Lecture Notes in Computer Science, Springer-Verlag, 1986, p.372-394.

[Ng] Ng, P. and Yeh, R., Modern Software Engineering Foundations and Current Perspectives, Van Nostrand Reinhold, 1990.

[NGCR] Naval Research Advisory Committee, Next Generation Computer Resources, Committee Report, February 1989.

[PDW] PDW 120-S-00533(Rev.B, Change 4), Over-the-Horizon Targeting (OTH-T) Gold Reporting Format, Naval Tactical Interoperability Support Activity, 30 June 1989.

[Stankovic] Stankovic, J. and Ramamritham, K., Hard Real-Time Systems Tutorial, Computer Society Press, 1988.

[Steigerwald] Steigerwald, R., Luqi, and McDowell, J., "Rapid Prototyping with Reusable Software Components: Methodologies for Component Storage and Retrieval", submitted to *Journal of Software Engineering and Knowledge Engineering* for publication.

[TAE] Transportable Applications Environment (TAE) Plus, National Aeronautics and Space Administration, Goddard Space Flight Center, January 1990.

[Tanik] Tanik, M. and Yeh, R., "The Role of Rapid Prototyping in Software

Development", *IEEE Computer*, v. 22, n. 5,pp. 9-10, May 1989.

[Vlissides] Vlissides, J. M., and Linton, M. A., "Applying Object-Oriented Design to Structured Graphics", Proceedings of the 1988 USENIX C++ Conference, October 1988.

[Tyszberowicz] Tyszberowicz, s, and Yehudai, A., "OBSERV Object-Oriented Specification, Execution, and Rapid Verification System", 3rd Israeli Conference on Computer Systems and Software Engineering, Tel-Aviv, Israel, June 1988.

[Yourdon] Yourdon, E., Modern Structured Analysis, YOURDON Press, 1989.

REQUIREMENTS SPECIFICATION OF HARD REAL-TIME SYSTEMS: EXPERIENCE WITH A LANGUAGE AND A VERIFIER

Constance L. Heitmeyer
Bruce Labaw

Introduction

Hard real-time (HRT) computer systems must deliver results within specified time intervals or face catastrophe. To detect timing problems in HRT systems, current development practice depends on exhaustive testing of the software code and extensive simulation. Unfortunately, this expensive and time-consuming process often fails to uncover subtle timing and other software errors. To improve this situation, important new research in real-time computing is now in progress, largely in scheduling theory (e.g., [Ramamritham89, Lehoczky89, Leung90]) and real-time programming languages and operating systems (e.g., [Donner89, Gerber89, Tokuda89]).

Although such research should significantly improve the quality of HRT software code, further research is needed in methods for specifying and verifying the requirements of HRT systems. Correct specifications of the requirements are critical: studies have shown that errors introduced during the requirements phase can cost as much as two hundred times more to correct than errors made later in the software life-cycle [Boehm81]. Unfortunately, most current software requirements documents for HRT systems are of poor quality, containing little precise guidance on the required timing behavior. Developers are usually forced to glean essential details from informal, natural language descriptions that are ambiguous, imprecise, and incomplete. Timing requirements are often missing. When they exist, they are usually embedded in premature design decisions.

To remedy this situation, we advocate a three-phased approach to developing HRT systems that emphasizes the requirements phase of the software life-cycle. With this approach,

- mathematically precise specifications of the timing and other system requirements are developed,
- machine-based verification tools are applied to the requirements specifications to improve self-consistency and to insure compliance with critical timing and other properties, and
- a semiautomated procedure is used to develop an implementation from the specifications. This implementation must meet the timing and functional constraints imposed by the requirements specifications.

Such an approach should dramatically decrease the number of timing and other errors in the system implementation. Formal requirements specifications should reduce errors by including rigorous definitions of the timing requirements and by removing ambiguous and unnecessary information. Computer-based verification should decrease errors by uncovering inconsistencies in the specifications and by demonstrating that the specifications satisfy critical timing properties. Finally, semiautomatic generation of an implementation should reduce the number of new errors introduced during the transition from requirements to software design and implementation

Present work at the Naval Research Laboratory (NRL) is focused on the first two phases of this approach. Our interest is in methods and tools that help software developers specify, analyze, and verify the functional and timing requirements of HRT systems. A major goal is to assemble a software requirements toolset containing tools developed at NRL (to be described in a future report) as well as promising tools developed elsewhere. Because most existing methods and tools for software specification focus on functional behavior, we especially seek tools for specifying and analyzing timing behavior. Also of special interest are methods and tools that *scale up*, i.e., tools that are useful in specifying and verifying requirements of real-world, practical HRT software.

The NRL toolset includes four classes of tools, namely, requirements generation tools, consistency checkers, verifiers of functional and timing properties, and tools that help build an executable version of the specification [Heitmeyer90]. The language supported by requirements generation tools should lead to formal, yet intuitive, specifications. The purpose of a consistency checker is to insure that the requirements

specification does not contradict itself; without executing the specification, such a tool detects those parts of the specification that are inconsistent. Verifier tools provide formal proof that given assertions about functional behavior and timing can be derived from the specification; of special interest in the development of real-time software are proofs that certain critical events occur within specified time intervals. A tool that helps translate a requirements specification into an executable form has two important benefits. First, by running the executable version, the specifier can determine whether the requirements specification accurately describes the intended external behavior. Second, running an executable version can provide modeling information useful in defining the timing constraints on certain crucial functions; such constraints are needed in reasoning about a system's timing behavior.

Recently, NRL studied existing commercial tools for requirements specification to determine whether any provide the four classes of support described above. Given our focus on HRT software, our study included only those tools whose vendors claim are designed for real-time software specification. Because the commercial CASE tools market is changing rapidly, installing a collection of tools and executing benchmark tests to compare them was not expected to be cost-effective: new tools and updated versions of existing tools were likely to be introduced before such tests could be completed. Consequently, the NRL study relied on reviews of vendor literature and discussions with tool vendors. Based on these, we concluded that no commercial tool provides all four classes of support. Although a few provide some simple checks of a specification's consistency and limited support for building executable specifications, no current commercial tool supports verification of critical functional and timing properties.

Although current commercial tools supporting HRT requirements specification are few and limited in capability, the SARTOR project at the University of Texas (UT) has developed two promising experimental tools. The first, a requirements generation tool, supports a graphical language, called Modechart [Jahanian88a, Jahanian91] that is designed to specify a system's timing requirements. The second, a verification tool, provides mechanical proof that a specification satisfies critical timing properties [Jahanian88b, Stuart90]. These prototype SARTOR tools are based on methods for specifying and analyzing timing properties that complement methods for specifying functional requirements [Heninger78, Heninger80] developed in NRL's Software Cost Reduction (SCR) project.

This paper describes NRL's experience with the Modechart language and a prototype version of the verification tool [Stuart90]. Recently, we developed Modechart specifications for several example systems and then used SARTOR's verifier to prove the consistency of a set of Modechart specifications with selected timing assertions. This paper introduces SARTOR and the Modechart language, presents two sets of Modechart specifications and associated timing assertions, and evaluates Modechart and the SARTOR verifier, identifying their contributions to real-time software technology and recommending improvements and enhancements. The paper concludes with a summary of significant issues in real-time specification and verification that are topics for future research.

1. Overview of SARTOR and Modechart

After describing how a specifier uses the SARTOR tools to build and prove properties about a requirements specification, this section provides a brief, informal summary of the Modechart language. For a more complete description of Modechart, see [Jahanian91].

SARTOR Overview. Figure 1 shows the relationship between three tools in the SARTOR toolset. These are **chart**, which supports the generation of Modechart specifications; **trans**, which translates Modechart specifications into a form of first-order predicate logic called Real-Time Logic (RTL) [Jahanian86]; and **verify**, which provides automated support for verifying timing assertions expressed in RTL. Each timing assertion is a *safety assertion*, i.e., a logical statement of the properties that must hold for the specifications to be considered correct. Such an assertion describes either the required temporal ordering of events (e.g., "No weapon can be released unless the Master Arm switch is on") or the required temporal distance between events (e.g., "The fire warning light is illuminated no more than 250 milliseconds after the software detects an engine temperature above the upper limit").

To develop, analyze, and verify a real-time requirements specification, the specifier first uses **chart** to express the requirements in the Modechart language and then applies **trans** to translate the Modechart specifications into RTL. Next, the specifier expresses the required timing properties in terms of RTL assertions. Finally, he applies **verify** to the specifications and each assertion to determine whether the assertion can be derived from the specifications. In particular, **verify** determines whether the assertion is valid, satisfiable, or not satisfiable. If the assertion is valid, then any legal implementation of the specifications is guaranteed to satisfy the assertion. If the assertion is satisfiable, then it is possible to find some implementations that satisfy the assertion, but certain implementations may not. If the assertion is not satisfiable, then the specifications are intrinsically incompatible with the assertion.

The specification and verification process outlined above is that ultimately envisioned by the SARTOR researchers. The current process, illustrated by the dashed line in Figure 1, is simpler. At present, the verifier operates directly on the Modechart specifications: no translation of the Modechart specifications to RTL is required. Further, the classes of timing properties that the current verifier can prove about Modechart specifications is limited.

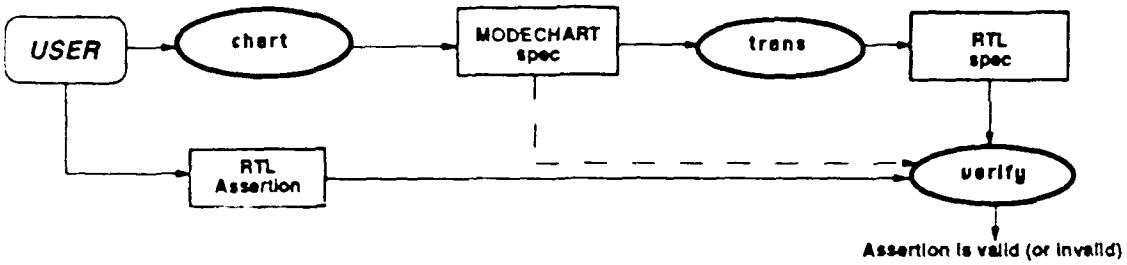


FIGURE 1. SPECIFICATION AND VERIFICATION WITH THE SARTOR TOOLSET

Modechart Specification Language. The historical roots of Modechart are Statechart [Harel90], a graphical language based on concurrent finite state diagrams, and the concept of modes invented by NRL in the SCR project [Heninger78]. The formal semantics of Modechart are defined by an action/event model containing five constructs. One of these, the *mode* construct, describes control information that imposes structure on a system's operation. Each mode is either primitive, parallel, or serial. A *primitive* mode contains no other modes, i.e., has no children. A *parallel* mode contains one or more children that are said to be *in parallel*; if the system is in parallel mode M, then the system is simultaneously in all of M's children. A *serial* mode contains one or more child modes that are said to be *in series*; if the system is in a serial mode M, then the system is in exactly one of M's children. Given a serial mode M, exactly one of M's children is the *initial mode*, the mode entered when mode M is entered.

The other four constructs of the model are actions, events, state variables, and timing constraints. An *action* is an operation that is executed when a mode becomes active; each mode has at most one associated action. A *state variable* describes the current state of a property or physical aspect of the system. An *event* is a point in time at which some state change occurs that is significant to the system's behavior. Several classes of events exist, including *external events*, e.g., the system operator sends an input to the system; *start* or *stop events*, which mark the starting and completion times of actions; *state variable transition events*, which mark a change in a state variable's value, and *mode entry* and *mode exit events*, which capture mode changes. In Modechart, a mode transition occurs when either a triggering condition or a timing constraint is satisfied. A *triggering condition* is an event occurrence and/or the truth of a predicate on modes or state variables (e.g., **DAVAIL** = *false*). Modechart uses deadlines and delays to specify timing constraints on mode transitions. A *deadline* is an upper bound on the time interval from mode entry to mode exit, a *delay* is a lower bound. Both Modechart and RTL assume a discrete model of time (i.e., represent time by the natural numbers).

In Modechart, state variables and modes have important semantic differences. Changing the value of a state variable requires the explicit invocation of an action. Because actions take nonzero time to complete, changing a state variable's value takes nonzero time. In contrast, a mode transition takes effect in zero time, i.e., the exit from the old mode and the entry into the new mode occur at the same time. Moreover, changing the value of a state variable requires the explicit invocation of an action. In contrast, a mode transition, e.g., mode entry, is implicit, occurring when a given triggering condition or upper/lower bound condition is satisfied.

Timing assertions about a set of Modechart specifications are expressed in RTL using a special *occurrence function* denoted by @ [Jahanian86]. This function, which has the form $@(\mathbf{E}, i) = j$, maps the ith occurrence of an event $\mathbf{E}$ to the time j that $\mathbf{E}$ occurred. Occurrence functions for mode entry and mode exit events are represented by the expression $@((M = b), i)$, where M is a mode and b is *true* (T) or *false* (F); $@((M = T), i)$ represents the time of the ith entry into mode M, $@((M = F), i)$ the time of the ith exit from mode M. Figure 2, which is based on [Stuart90b], describes some timing properties that the SARTOR verifier can deduce from a set of Modechart specifications. In Figure 2, M_k represents the kth mode; c, c', and c'' are non-negative integer constants; and $t_{k,i}$ and $\hat{t}_{k,i}$ represent the times $@((M_k = T), i)$ and $@((M_k = F), i)$. An asterisk (*) next to the name of a timing property indicates that any $\leq$ in the corresponding formula may be replaced by a $<$. Note that the verifier either determines whether a formula is consistent with the specifications (e.g., Inner Universal, Reachability) or deduces timing information from the specifications (Separation, Elapsed Time).

NAME	TIMING PROPERTY
Inner Universal*	Given modes M_1, M_2, M_3, M_4, $\forall i \exists j: \ t_{1,j} + c \leq t_{2,i} \ \wedge \ t_{2,i} + c' < t_{3,i} \ \wedge \ t_{3,i} + c'' \leq t_{4,j}$
Outer Universal*	Given modes M_1, M_2, M_3, M_4, $\forall j \exists i: \ t_{1,j} + c \leq t_{2,i} \ \wedge \ t_{2,i} + c' < t_{3,i} \ \wedge \ t_{3,i} + c'' \leq t_{4,j}$
All Universal	Given modes M_1, M_2, M_3, M_4, $\forall i: \ t_{1,i} + c \leq t_{2,i} \ \wedge \ t_{2,i} + c' \leq t_{3,i} \ \wedge \ t_{3,i} + c'' \leq t_{4,i}$
Separation	Given modes M_1, M_2, find c such that $\forall i: t_{1,i} + c < t_{2,i+1}$ ($t_{1,i} + c$ is a lower bound of $t_{2,i+1}$) $\forall i: t_{1,i} + c > t_{2,i+1}$ ($t_{1,i} + c$ is an upper bound of $t_{2,i+1}$) Alternatively, given modes M_1, M_2, find c, j such that $\forall i: t_{1,i} + c < t_{2,j}$ ($t_{1,i} + c$ is a lower bound of $t_{2,j}$) $\forall i: t_{1,i} + c > t_{2,j}$ ($t_{1,i} + c$ is an upper bound of $t_{2,j}$)
Reachability	Given modes $M_1, M_2, \ldots, M_K$ and modes $M'_1, M'_2, \ldots, M'_L$, determine if $\{M'_1, \ldots, M'_L\}$ is *reachable* from $\{M_1, \ldots, M_K\}$, i.e., if $\exists t, i_1, i_2, \ldots, i_K, l \in \{1, 2, \ldots, L\}, i_l: \ t \in (\cap_{k=1}^{K} [t_{k,i_k}, \hat{t}_{k,i_k}]) \ \wedge \ t \in [t_{l,i_l}, \hat{t}_{l,i_l}]$
Elapsed Time	Given mode M_k and the set of times $D = \{\hat{t}_{k,i} - t_{k,i} \mid i \in I^+\}$ that the system is in M_k, find $d_{max} \in D: \ \forall d \in D, d_{max} \geq d$ (maximum time in M_k) or find $d_{min} \in D: \ \forall d \in D, d_{min} \leq d$ (minimum time in M_k)

FIGURE 2. SOME TIMING PROPERTIES SUPPORTED BY THE VERIFIER

2. Modechart Examples

To evaluate Modechart and the verifier, we generated several sets of Modechart specifications, two of which are presented in this section. Example 1 describes the required system behavior at a railway crossing. Its purpose is to suggest an alternative, designed for ease of change, to the specification presented in [Jahanian88]. To evaluate Modechart and the current verifier on a more realistic example, we extracted the second example from the software requirements document of an existing avionics system, the Operational Flight Program (OFP) for the A-7E aircraft [Heninger78]. Example 2 has several features that make the specification nontrivial, i.e., a shared resource (the display) and several different environmental inputs and outputs. Prior to presenting the examples, we describe the top-level structure that we used to construct the Modechart specifications. The structure is designed to make the specifications easy to change.

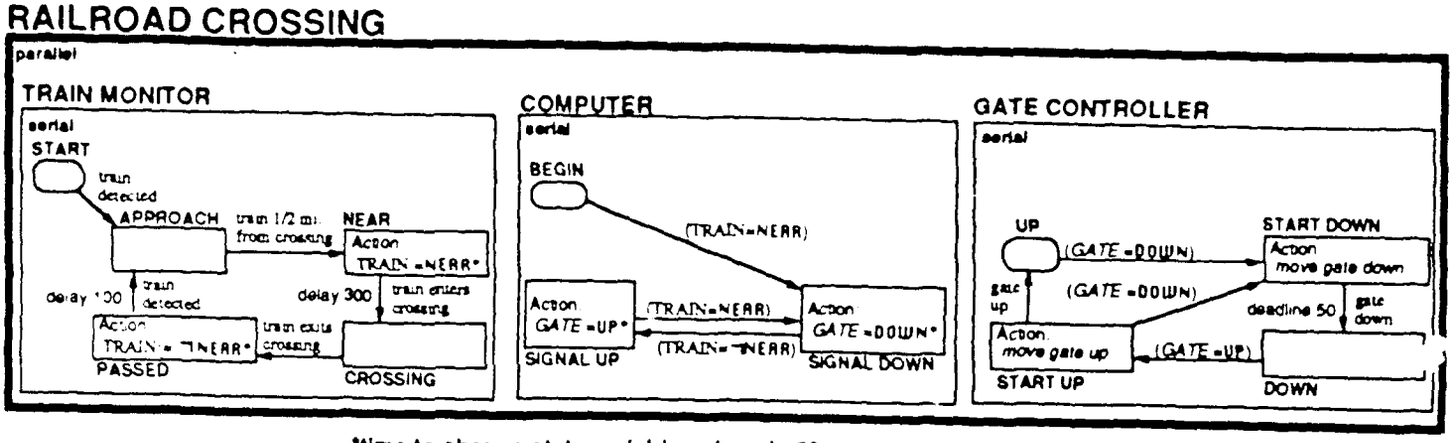
Organizing Modechart Specifications for Ease of Change. A critical aspect of a requirements document and one whose importance is often overlooked, is the document structure. Because the document specifying a system's requirements is likely to change, both during development and when subsequent versions of the system are built, the document should be organized for ease of change. Influenced by the structure of the A-7 requirements document [7] and Parnas' theory of software documentation [15], we have designed a methodology for organizing Modechart specifications based on ease of change. With this methodology, a HRT system is described as a top-level parallel mode with three children: an *input recognizer*, an *output generator*, and a *processor*. The input recognizer describes the required behavior of the input devices, i.e., translates the environmental variables of interest (such as characters typed by a human, continuous data from a sensor) into discrete input data items. The output generator describes the required behavior of the output devices, i.e., translates discrete output data items into the appropriate user-visible output (the display of sensor data, the firing of a missile, etc.). The processor component uses history (captured by the current set of modes) and input data items to initiate the appropriate output. In Modechart, external events represent environmental input variables, state variables represent input and output data items, and actions are used to produce user-visible output and to assign values to input and output data items.

Given this structure, we impose restrictions on how information is communicated among the three top-level modes. As in [Heninger78], the input recognizer in the sample Modechart specifications below uses only input data items to communicate with the processor component, while the processor component uses only output data items to communicate with the output generator. Future Modechart specifications could relax these restrictions somewhat. For example, a driver in the input recognizer might communicate directly with a driver in the output generator. Further, for ease of change, we limit the cases in which mode names can cross mode boundaries: only names of modes used in the restricted SCR manner (see [Heninger78]) can cross mode boundaries and only within the processor component.

Organizing the requirements in this manner is based on separation of concerns. Each of the three top-level modes captures some well-defined aspect of the requirements and isolates it from the other requirements. In particular, the input recognizer encapsulates information about the relevant environmental input variables (i.e., their values, data types, etc.) and their relation to the input data items; it makes no assumptions about how the variables will be used. Similarly, the output generator only contains information about the environmental output variables and their relation to the output data items; it makes no assumptions about the input. Only the processor mode describes the required relation between the inputs and the outputs.

Example 1: Railroad Crossing. In this example, the system's purpose is to lower a gate at a railroad crossing when a train approaches and to keep the gate down as long as the train is in the crossing. We assume that trains only move in one direction and that two trains are always some minimum distance apart. The original Modechart specification of this example (see [Jahanian88a]) consists of two parallel modes. In contrast, the specification shown in Figure 3, called **RAILROAD CROSSING**, uses the three top-level modes described above. The input recognizer, called **TRAIN MONITOR**, uses modes to represent the system state relative to four external events, the detection of an incoming train and the train's position at three points relative to the railroad crossing, namely, 1/2 mile from the crossing, at the entrance to the crossing, and at the crossing exit. An input data item TRAIN with value **NEAR** signals that the train is 1/2 mile from the crossing. The output generator, called **GATE CONTROLLER**, describes the behavior of the output device that raises and lowers the crossing gate. Finally, the processor, called **COMPUTER**, uses the train's position relative to the crossing and the current mode to signal via an output data item, *GATE*, that the gate is to be lowered or raised.

Figure 3 presents the Modechart specifications for this example. In the figure, parallel modes are represented by thick lines, serial modes by thin lines, initial modes by oblongs, and all modes but initial modes by rectangles. State variable transition events are represented by the notation (**state-variable**=*value*). In Figure 3 are several delays and deadlines that are a part of the Modechart specifications. The local timing constraint, 'delay 300', on the transition in the Train Monitor from mode **NEAR** to mode **CROSSING** indicates that a train will enter the crossing at least 300 time units after it is 1/2 mile in front of the crossing. In the Gate Controller, the local constraint, 'deadline 50', on the transition from mode **START DOWN** to mode **DOWN** indicates that the action lowering the gate takes at most 50 time units to complete.



*time to change state variable values is 50

FIGURE 3. MODECHART SPECIFICATIONS OF RAILROAD CROSSING EXAMPLE

A timing assertion we would like to prove about the specifications is that the gate is down while a train is in the crossing. To express this assertion in RTL, we write

$$\forall i \exists j \, @((\text{DOWN} := T), j) \leq @((\text{CROSSING} := T), i) \wedge @((\text{CROSSING} := F), i) \leq @((\text{DOWN} := F), j).$$

This means that every time interval during which the system is in CROSSING mode is contained in a time interval during which the system is in DOWN mode. Because mode transitions take zero time, the exit time for mode DOWN is equal to the entry time for mode START UP and the exit time for mode CROSSING is equal to the entry time for mode PASSED. In addition to proving the above assertion, we also proved a weaker assertion, namely, if the train is in the crossing, then the gate is down (in RTL, CROSSING $\rightarrow$ DOWN). Because the verifier cannot prove formulas in the latter form, we proved that the negation of this assertion, CROSSING $\wedge$ $\neg$DOWN, defines an unreachable state. To generate the proof, we augmented the Modechart specifications with an *unsafe* state, a state that violated the assertion, and executed the verifier on the augmented specifications to determine whether the unsafe state was reachable. Because it was not, the assertion is considered proven.

Example 2: Pilot Data Entry and Display. Figure 4a uses a set of Modechart specifications for a function performed by the OFP; Figure 4b shows two timing assertions we wished to prove about the specifications. The OFP function reads a character sequence (e.g., latitude or longitude) typed by the pilot and writes the sequence to a display panel. To initiate data entry, the pilot first presses the **DATA ENTRY** button. The software responds by turning on a keyboard light and clearing the display panel. Next, the pilot types a sequence of characters, which the software writes one character at a time to the display panel. Finally, the pilot presses the **ACCEPT** button to indicate that he has completed data entry. In response, the software turns off the keyboard light and clears the display panel.

To specify this example in Modechart, Figure 4a shows three top-level modes, called Pilot Input Recognizer, Data Entry and Display Function, and Output Generator, that correspond to the input recognizer, the processor, and the output generator described above. The Pilot Input Recognizer consists of three input drivers, one for each of the three hardware devices that the pilot uses to communicate with the software, namely, the **DATA ENTRY** button, the **ACCEPT** button, and the alphanumeric keyboard. The Data Entry and Display Function specifies how the system responds (i.e., what outputs it produces) to a sequence of inputs. The software response depends on both the mode that the software is in as well as the input. Like the processor component of the Train Crossing example, the Data Entry and Display Function receives input via changes to input data items and produces output by changing output data items. The Output Generator translates output data items into specific outputs (e.g., turn the keyboard light on). It consists of two drivers, one controlling the keyboard light, the other writing output to the display panel. In Figure 4a, conditions on state variables (i.e., predicates that remain true for some nonzero time interval) are represented by the notation **state-variable**=*value*, and all unlabeled mode transitions occur at the time of action completion.

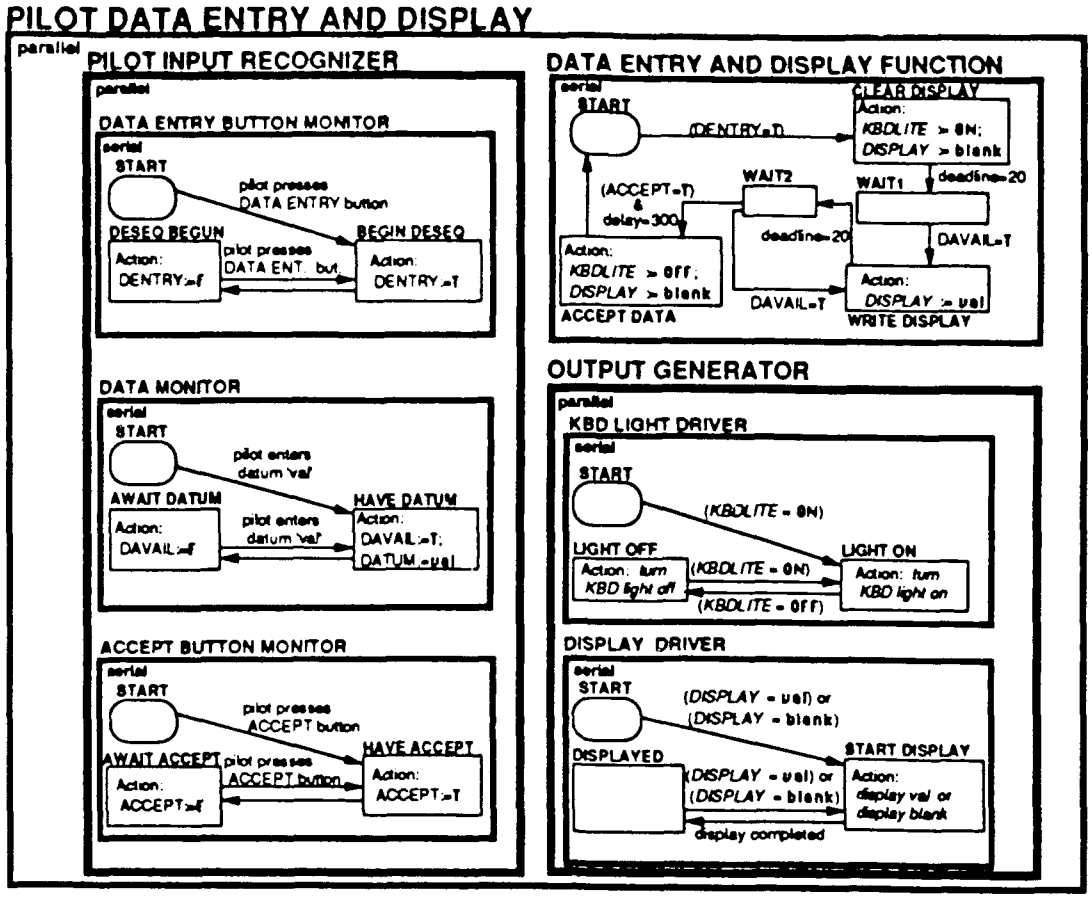


FIGURE 4a. MODECHART SPECIFICATIONS OF PILOT ENTRY AND DISPLAY EXAMPLE

	ENGLISH DESCRIPTION	DESCRIPTION IN RTL
(1)	Datum displayed within 200 t.u. after pilot entered datum	$\forall i @((\text{HAVEDATUM} := T), i) \leq @((\text{DISPLAYED} := T), i) \wedge$ $@((\text{DISPLAYED} := T), i) \leq @((\text{HAVEDATUM} := T), i) + 200$
(2)	Last datum displayed at least 225 t.u. before pilot can enter ACCEPT	$\forall i \exists j @((\text{HAVEDATUM} := T), j) \leq @((\text{HAVEACCEPT} := T), i) \wedge$ $@((\text{HAVEACCEPT} := T), i) \leq @((\text{HAVEDATUM} := T), j+1) \wedge$ $@((\text{STARTDISPLAY} := F), 2i + j - 1) + 225 \leq @((\text{HAVEACCEPT} := T), i)$

FIGURE 4b. TIMING ASSERTIONS FOR PILOT ENTRY AND DISPLAY EXAMPLE

Using the current verifier, we proved two timing assertions about this set of Modechart specifications. One assertion is that a data character is displayed within some fixed time interval after it is entered. Specifically, if t is the time that the pilot entered the ith character, then the time at which the ith character appears on the display panel is less than or equal to $t + 200$. To express this assertion in RTL, we write

$$\forall i @((\text{HAVEDATUM} := T), i) \leq @((\text{DISPLAYED} := T), i) \wedge$$
$$@((\text{DISPLAYED} := T), i) \leq @((\text{HAVEDATUM} := T), i) + 200. \quad (1)$$

Unlike the proof in the railroad crossing example, where the timing assertion could be derived from the original Modechart specifications, proving this assertion required the addition of two constraints to the Modechart specifications. First, we needed to bound the pilot's input rate. Based on human performance limitations (humans can only type so fast), we defined a lower bound on the time interval between any two successive pilot key presses. Second, we needed to impose an order on the sequence of pilot inputs, since this assertion is only valid for some pilot input sequences. The specification of the Input Recognizer in Figure 4a permits all possible pilot input sequences, even illegal ones. The assertion in (1) is true only when the pilot enters a legal sequence: a DATA ENTRY followed by one or more data characters followed by an ACCEPT.[1]

[1] In a complete specification of Example 2, the Data Display Function would also recognize illegal input sequences and

Although we used the verifier to prove (1), the proof only shows that the display is updated within some time period after the ith character is entered; both entry of a data character and entry of an **ACCEPT** could cause the display to be updated. One solution is to replace the START DISPLAY mode in the display driver specification with two modes: START DISPLAY VAL (displays a data value) and START DISPLAY BLANK (displays one or more blanks). Then, the timing assertion to be proven is

$$\forall i @((\text{HAVEDATUM} := T), i) \leq @((\text{STARTDISPLAYVAL} := F), i) \wedge$$
$$@((\text{STARTDISPLAYVAL} := F), i) \leq @((\text{HAVEDATUM} := T), i) + 200.$$

Because the verifier cannot prove formulas of this form, the assertion was rewritten in Modechart and proved using a reachability argument. (We note that a solution that replaces the START DISPLAY mode with two new modes is undesirable for ease of change reasons. The output driver specifications should not be influenced by the requirements of the verification process.)

The second timing assertion states that a minimum delay exists between the time that the last character of the character string is displayed and the time that a pilot press of the **ACCEPT** key is allowed. The rationale is that, before the pilot presses **ACCEPT**, he needs a minimum time to read and validate the string of characters that appear on the display. This assertion is expressed in RTL as[2]

$$\forall i \exists j @((\text{HAVEDATUM} := T), j) \leq @((\text{HAVEACCEPT} := T), i) \wedge$$
$$@((\text{HAVEACCEPT} := T), i) \leq @((\text{HAVEDATUM} := T), j + 1) \wedge$$
$$@((\text{STARTDISPLAY} := F), 2i + j - 1) + 225 \leq @((\text{HAVEACCEPT} := T), i). \quad (2)$$

If, in (2), complete pilot input sequences are required (i.e., every **DATA ENTRY** is paired with a unique **ACCEPT**) and if i is the number of character strings and j the total number of characters entered by the pilot, then $2i + j - 1$ is the total number of pilot key presses. To prove (2), we needed to augment the two constraints above with a third constraint that defines an upper bound on the length of each character string and requires that an **ACCEPT** key press terminate each character string.

To prove the timing assertions in (1) and (2), we needed to extend the original set of Modechart specifications shown in Figure 4a. Because the current verifier can only prove consistency between a set of Modechart specifications and a single RTL assertion, we generated Modechart specifications for all three constraints, adding them to the original Modechart specifications. (A technical issue about this approach is how to show consistency between the original and the extended Modechart specifications.) In developing the proofs of the assertions, an important consideration was whether the three constraints were requirements missing from the original specifications or whether they were simply logical statements needed to complete the verification process. We decided that two of the constraints should be added to the original specifications, in particular, the constraint describing human performance limitations and the constraint limiting the character string length and requiring termination of a character string by an **ACCEPT**. In contrast, the remaining constraint, which defines the legal pilot input sequences, was simply needed to complete the verification process: the statement that we wanted to prove concerns the system's response given legal pilot input. Hence, the assertion in (2) is incomplete. A complete statement of the assertion includes (2) as a consequent and a description of legal pilot input as an antecedent.

3. Contributions of Modechart and the SARTOR Verifier

Modechart. The SARTOR research effort has contributed to real-time software technology by providing an integrated approach to the specification and verification of critical timing properties. A crucial aspect of SARTOR is the Modechart language. While specifications in logic-based languages, such as RTL, other first-order languages (e.g., [Heitmeyer83, Auernheimer86]), and temporal logics (e.g., CTL [Clarke87] and RTTL [Ostroff89]), facilitate machine-based analysis and verification, humans find such specifications hard to produce and hard to understand (e.g., see [Jaffe89]). In contrast, we found the graphical Modechart specifications highly readable and relatively easy to generate. Although complete graphical specifications of the requirements may be impractical for large systems, the

generate appropriate responses (e.g., error messages).

[2] The second clause of the formula is only checked if $@((\text{HAVEDATUM} := T), j + 1)$ is defined, that is, if the pilot has entered a character following the **ACCEPT**.

readability of the Modechart specifications make them very useful during the process of constructing the requirements specifications.

A fundamental contribution of Modechart is the ease with which specifiers can use the language to understand and reason about a system's timing behavior. Specifiers can first use modes, actions, events, and state variables to define the parallelism and sequential behavior inherent in the application domain. We found that Modechart's hierarchical structure facilitated the construction of our specifications by allowing us to combine top-down and bottom-up approaches. Once the system's *functional behavior* is defined, then timing behavior can be added in terms of deadlines and delays.

Unlike temporal logics, such as CTL and RTTL, which are designed to specify the temporal ordering of events, Modechart and RTL are designed to specify both the temporal ordering of events and the temporal distance between events. In real-time systems, constraints on the temporal ordering of events are insufficient. In such systems, certain critical events (e.g., the firing of a weapon, an alert signaling the spill of a hazardous substance) need to occur within specified time intervals. Unlike languages based on temporal logic, Modechart and RTL provide a compact notation for defining the timing constraints imposed on critical events. These constraints are described in Modechart by delays and deadlines, in RTL by the occurrence function.

In addition to producing highly readable specifications that compactly express both temporal ordering and temporal distance, Modechart has additional benefits lacking in other specification languages. Unlike [Heninger78], which describes only the software requirements (represented in our specifications by the processor component), Modechart can describe the complete *system requirements*. The inclusion in Modechart of external events as well as action completions makes the specification of the complete system requirements possible. An additional benefit is Modechart's support for concurrency. In a HRT system, input devices, output devices, and computers need to operate concurrently, and their behavior needs to be synchronized. The parallel modes included in Modechart make the description of concurrency possible, while Modechart's state variables enable synchronization and communication among parallel modes. A third benefit is Modechart's support of nondeterminism. As Gabrielian has noted [Gabrielian90], in some parts of a specification, an event or condition may trigger a transition to more than one mode transition. If the actual requirements permit any one of the possible transitions, forcing a transition to exactly one mode is a premature design decision. Modechart's semantics allow the specifications to express such nondeterminism.

Verifier. We found the prototype verifier useful in improving the correctness and the completeness of our sample specifications. While human proofs of timing assertions are feasible, such proofs often contain errors, first, because proofs involving inequalities and substitutions are tedious, and, second, because humans may fail to provide complete proofs, especially for boundary cases. The verifier not only allowed us to detect such errors but also increased our overall understanding of the specifications, especially the interactions of individual components. However, while helpful, verification tools do not free the human from thinking about the logic of the specifications. They provide mechanical assistance for checking the logic. Our experience suggests that humans working with a mechanical verifier are more likely to find errors in the specifications' logic than humans doing manual verification alone.

Although it was designed to prove timing requirements, we discovered that the SARTOR verifier can also prove a class of functional properties. For example, to prove the assertion, "If the navigation mode is **AFLYUPDATE**, then the weapon mode is **BOC**," we can augment the Modechart specifications with an unsafe state, **AFLYUPDATE** $\wedge \neg$ **BOC**, and prove that this state is unreachable. However, the functional properties that the current verifier can prove are limited. It cannot, for example, prove functional properties that rely on data type definitions, since the tool does no type checking. (Moreover, Modechart lacks constructs for defining data types.)

4. Further Development of Modechart and the Verifier

As noted above, the Modechart language and the SARTOR verifier are prototypes. In this section, we recommend some ways in which the SARTOR toolset could be more fully developed. One general comment about both Modechart and RTL concerns expressiveness. In some cases, a constraint was more easily expressed in one language than the other. For example, a Modechart specification of the legal pilot input sequences is straightforward, whereas a specification of the sequences in first-order predicate logic (such as RTL) is tedious and less intuitive, requiring considerable notation for bookkeeping purposes. In contrast, sometimes an assertion is more easily expressed in RTL. For example, given a set of Modechart specifications, defining timing constraints involving nonadjacent modes is easier and more natural in RTL than in Modechart. Further analysis is needed to identify other classes of constraints that are more easily expressed in one language than in the other.

Modechart.

Passing Values. In Modechart, we found no formal way to capture a value which accompanies an external event. In the Pilot Data Entry and Display specifications, for example, suppose the pilot types the letter 'N'. The information to be communicated consists of two parts: the event E, where E represents the event 'new data available', and the value 'N'. Although Modechart provides notation for describing the event E, namely, ΩE, no notation exists for describing the value 'N'. One existing formal notation that can describe both an external event and the value accompanying the event is proposed in [Jacob86].

Shared Resources. We found the statement of certain timing assertions impossible if a given resource (e.g., a display device) was shared rather than dedicated. This problem arises from a lack of expressiveness in Modechart. To illustrate this, we consider Example 2. If the display driver is dedicated to the display of the character data, then the ith entry into the HAVEDATUM mode of the DATA MONITOR represents receipt of the ith datum and the ith entry into the DISPLAYED mode of the DISPLAY DRIVER corresponds to the user-visible display of the ith datum. However, if the display driver is shared, this correspondence no longer exists: the current version of Modechart provides no way to associate the ith entry into a mode with the jth time an associated action is completed by a shared resource.

Continuous Environmental State Variables. Currently, Modechart cannot describe continuous variables. While software can only handle discrete-valued variables, environmental variables that represent system inputs and outputs may be either continuous or discrete. An example of a continuous environmental variable is air pressure. Describing such a variable over time as a sequence of discrete samples, rather than as a continuous function of time, is a premature design decision. One possible solution that merits investigation is Parnas' concept of monitored and controlled variables [Parnas90].

Functionality. In generating sample Modechart specifications, we identified some cases in which Modechart's functionality was overly restricted. For example, Modechart prohibits the specifier from assigning timing constraints and triggering conditions to the same mode transition. In the Gate Controller specification in Figure 3, for example, both a deadline of 50 and a triggering condition ('gate down') are assigned to a single mode transition. The current Modechart semantics force the specifier to decompose this mode transition into two separate transitions, one governed by the deadline (or delay), the other by the triggering condition. In our view, assigning both timing constraints and triggering conditions to a single mode transition is more convenient (and less confusing). A second example of limited functionality is Modechart's treatment of *self-looping*, i.e., a transition from a mode back to itself. Even though specifiers find it very useful, Modechart prohibits self-looping for theoretical reasons: self-looping in modes with no associated action results in an infinite loop. In our view, self-looping should be allowed in Modechart as long as time in mode is nonzero.

Given such limitations on Modechart functionality, an issue is how to obtain the needed functions: should the Modechart semantics be changed or should 'syntactic sugar' be added? The solution is not always obvious. A change in the Modechart semantics needs to be carefully considered since the semantic model loses elegance as more and more features are added. At the same time, frequent use of syntactic sugar is also ill-advised, since syntactic sugar hides the semantic model from the user

Timing Constraints Involving Non-Adjacent Modes. Another problem that we experienced in Modechart was an inability to define timing constraints on non-adjacent modes, modes between which no mode transition exists. To express such timing constraints in Modechart, we needed to create dummy modes. (We regard dummy modes as undesirable artifacts that add to the specifications' complexity.) To illustrate this problem, we consider the timing constraints imposed on a HRT system by two sources, the system's physical environment and the performance limitations of the system's users. (Both sets of constraints should be included in the requirements specifications.) In the railroad crossing system, for example, suppose t represents the minimum time that a train requires to travel from a point 1/2 mile before the crossing to the crossing exit. Unless a dummy mode is created, Modechart provides no means of specifying the delay t. A similar problem arises in the Pilot Input and Display system. Because the pilot can only type so fast, we defined a lower bound on the time interval t between two consecutive key presses, say, the press of the **DATA ENTRY** button and entry of the first character. Because, in the Modechart specifications, two consecutive key presses may involve nonadjacent modes, expressing the delay t in Modechart is impossible without the use of dummy modes. (As stated above, RTL expresses such timing constraints easily.)

Resource Contention. In any real-time system, there will be contention for shared resources, such as I/O devices and processors. A problem is how to use the known timing information to determine that a schedule for assigning

the resources is feasible. Because Modechart is not equipped to handle resource contention, another tool is needed to analyze the specifications for scheduling feasibility.

Verifier.

User-Friendly Feedback. As [Rushby89] has stated, determining whether an assertion is valid is only one of the useful functions that a verifier performs. In addition, a verifier should support an interactive human-computer dialogue that enhances human understanding of the specifications and that facilitates reasoning about them. The computer's side of the dialogue should provide feedback that is easy for the human to understand; the human's side should facilitate communication to the computer of the human's intentions. Because the goal of the SARTOR verification effort was to find an efficient decision procedure for verifying timing assertions, little attention has yet been paid to the verifier's user interface. Although the human's input to the verifier is a set of Modechart specifications, the verifier's feedback takes the form of the computation graphs used in the verifier's implementation. The result is an unfriendly user interface. To understand the verifier's feedback, the human is forced to translate his Modechart specification into the appropriate computational graph. An improved user interface is needed before the tool can be used in a production environment.

Bounds on Timing Variables. During the requirements phase of software development, complete knowledge about the system timing will be unavailable. However, in most cases, specifiers will have some limited information about timing (e.g., the time required by a given output device to complete an action, the interval between successive inputs, human performance times, etc.). In such cases, specifiers should be able to represent as a variable the time that the processor needs to perform an action (such as a computation). Then, using the known timing information and global timing assertions, the verifier should be able to derive bounds on such variables. These upper and lower bounds on processor times would have high utility for system designers, since they could supply values for the parameters of pre-runtime schedulers.

Points in Time Versus Nonzero Time Intervals. In most cases, we wished to prove that an assertion was true for a nonzero time interval rather than simply a point in time. As an example, consider the timing condition in Figure 2 called Reachability. Given modes M_1 and M_2, suppose that the system is in mode M_1 for some nonzero time interval $d = [t_1, t_2]$ and that it enters another mode M_2 at time t_2. Currently, the verifier concludes that M_2 is reachable from M_1, even though the system spends zero time in M_2. We recommend that the current timing properties supported by the verifier be reviewed to determine which, if any, should hold for zero length time intervals.

Overspecification of the Constraints. To prove the RTL assertion in (2) using the verifier, we needed to overspecify the requirement on legal pilot input and on the maximum length of the input character string. The actual requirement for legal pilot input sequences is that, if the final character string in the sequence is terminated by a key press, that key press must be an ACCEPT. In other words, the specification allows incomplete sequences. However, to prove (2) for legal input sequences, the verifier requires the final key press (i.e., the ACCEPT) to be present. Also, the verifier requires the limit on the character string length to be a constant. Such constraints force overspecification of the requirements and are thus artifacts required by the current verification process. We note that overspecification of the constraint was the result of specifying the constraint *in* Modechart. Specifying the constraint in RTL would have avoided the problem but was not an alternative due to limitations that the present verifier imposes on input formats (see below).

Limited Formula Repertoire. As noted above, the present verifier only supports a small number of RTL formulas. To prove the assertions presented in Section 2, we needed several additional formulas, such as (1) $A \rightarrow B$, where A and B are logical statements, and (2) $\forall i A \leq B \wedge B \leq A + n$, where n is a positive integer and $A = \mathcal{O}((M_1 = T), i)$ and $B = \mathcal{O}((M_2 = T), i)$ are occurrence functions with $M_1 \neq M_2$. The current verifier also restricts the form of the occurrence function, accepting only functions of the form $\mathcal{O}((M = true), N)$, where N is a simple variable, not an expression (e.g., $2i + j - 1$). To handle the preceding formulas, we replaced each with an equivalent formula that the verifier supports. A future version of the verifier would be useful that proves such formulas in their original form.

Limitations on Input Format. In some cases, we wanted to prove an assertion about a combination of Modechart specifications and one or more RTL assertions. For example, it is easier to describe certain timing constraints, for example, those that refer to nonadjacent modes, in RTL than in Modechart. (Note that such timing constraints, while defined in RTL, are part of the specifications rather than assertions to be proven about the specifications.) Currently, the verifier can only prove an assertion about a set of Modechart specifications. Enhancing the verifier to prove properties about a combination of Modechart specifications and RTL assertions would be useful.

5. Future Research Topics in Real-Time Specification and Verification

In our view, the Modechart language and the SARTOR verifier represent a significant advance in the state-of-the-art of specification and verification of HRT systems. A major advantage of Modechart specifications is their readability and the ease with which specifiers can use the language to reason about timing. Moreover, unlike temporal logics, Modechart specifications can compactly express both temporal distance and temporal ordering. The SARTOR verifier demonstrates the feasibility of machine-based proofs that Modechart specifications have certain specified timing properties.

Based on our experiments with the SARTOR tools, we have identified a number of high-level technical issues that are beyond the scope of the current SARTOR project. Below, we summarize four major topics for future research.

- **Uniform Approach to Specification and Verification of Functional and Timing Properties.** The SARTOR toolset is designed to specify and verify only one aspect of the system requirements, namely, the timing requirements. Still needed is an approach to specification and verification that handles both functional AND timing requirements. As noted above, the SARTOR verifier can already prove a class of functional properties, e.g., that certain 'unsafe' states are unreachable [Stuart90]. However, a general-purpose verification tool is needed that can prove claims about both timing properties and a rich set of functional properties. Such a general-purpose tool could use, for example, data type definitions to determine whether certain assertions about functional behavior are valid.
- **A More General Timing Model.** The current discrete-time model used in defining Modechart and RTL is appropriate for describing the processor component (i.e., the software model) of the system requirements, since the software runs on a digital computer. However, to describe the environmental inputs and outputs, a continuous time model is more appropriate. Further, the timing model that underlies Modechart and RTL is an idealization. A single master clock is assumed that makes no errors. In real software systems, more than one clock may be used, and each clock is imperfect. Defining a more general timing model for SARTOR would be useful.
- **SARTOR Methodology for Requirements Specification and Verification.** At each phase of the software life-cycle, specifications have different purposes and different properties. Because of its generality, Modechart can be used to construct specifications at many different levels of abstraction. Due to our focus on requirements, we seek a methodology for building and verifying *requirements* specifications in Modechart. Such specifications must have properties needed in a requirements document, e.g., design for ease of change, avoidance of premature design decisions, etc. This requirements methodology should also cover verification, providing guidance on the handling of various aspects of verification (e.g., should timing constraints be specified in Modechart or RTL, should given timing constraints augment the original specifications or are they part of the formulas to be verified, etc.). The top-level structure sketched in Section 2 is one step in the direction of a complete, comprehensive methodology for building and verifying Modechart requirements specifications. The principles and guidelines initiated in the SCR project (see [Parnas86], [Parnas90]) are a good foundation for the methodology.
- **Methodology for Deriving an Implementation from a Modechart Requirements Specification.** Although the current SARTOR methods allow us to specify, and prove properties about, required timing behavior, no guidance exists on how to derive an implementation from the specifications and how to prove that the implementation and critical timing and functional properties are consistent. Without such a methodology, the current methods and tools are incomplete. Such a methodology would extend the requirements methodology described above to later phases of software development.
- **An Effective, User-Friendly Toolset Interface.** As noted above, tools, such as mechanical verifiers, can provide significant help in generating and improving the correctness of HRT specifications. Yet, tools that provide a powerful set of capabilities have significantly less utility if they have poor quality user interfaces. Unfortunately, the user interfaces of many existing CASE tools are ill thought-out and inadequately tested. Needed is research in interface design principles for tools supporting software specification and verification.

A final comment concerns the scaleability of Modechart. Little is known about the utility of Modechart, RTL, and the SARTOR verifier for building real-world systems. Our experiments suggest that Modechart's scaleability is limited: specifying large quantities of requirements data in graphical form is probably impractical. But this doesn't mean that Modechart isn't useful. In our view, more than a single approach to real-time requirements specification is needed. Because it produces highly readable, intuitive specifications of the required behavior, Modechart may be most appropriate during the process of building the requirements specification. In contrast, the tabular formats for requirements specification introduced in SCR (see [Heninger78, vanSchouwen90] for examples) are more appropriate in a reference document. These formats provide the reader with less intuition about the requirements than the

graphical notation but do concisely and formally describe the large volume of requirements data associated with real-world, practical software. For these reasons, the toolset we are constructing supports both the Modechart and the SCR 'views' of the requirements data. Our future goal is to develop a single conceptual model that supports these two 'views'.

Acknowledgments. The authors are especially grateful to Paul Clements of UT and NRL for many valuable discussions and, in particular, for clarifying the Modechart semantics and helping us organize the Modechart specifications for ease of change. We also thank John Gannon of the University of Maryland for valuable discussions and for helping articulate the different roles that Modechart and the SCR tables might play in the requirements specification process and both Al Mok and Doug Stuart of UT for allowing us to experiment with the SARTOR tools and for their openness to our suggestions concerning further development. We also acknowledge Paul Clements and John Gannon for comments on an earlier draft. Finally, we thank the other members of our project, Carolyn Brophy and Anne Rose, and our sponsor, CDR J. Van Fossen.

REFERENCES

[Auernheimer86] B. Auernheimer and R. Kemmerer, "RT-ASLAN: A Specification Language for Real-Time Systems," *IEEE Trans. Softw. Eng. SE-12*, 9, Sep. 1986.

[Boehm81] B Boehm, *Software Engineering Economics*, Englewood Cliffs, NJ, Prentice-Hall, 1981.

[Clarke87] E.M. Clarke and O. Grumberg, "Research on Automatic Verification of Finite-State Concurrent Systems," *Ann. Rev. Comput. Sci. 2*, 269-90, 1987.

[Donner89] M. Donner et al., "A Structuring Mechanism for a Real-Time Runtime System," *Proceedings, Real-Time Systems Symposium*, Santa Monica, CA, Dec. 5-7, 1989, 22-30.

[Gabrielian90] A. Gabrielian et al., "Specifying Real-Time Systems with *Extended* Hierarchical Multi-State (HMS) Machines," Thomson-CSF, Inc., report 90-21, Jan. 1990.

[Gerber89] R. Gerber and I. Lee, "Communicating Shared Resources," A Model for Distributed Real-Time Systems," *Proceedings, Real-Time Systems Symposium*, Santa Monica, CA, Dec. 5-7, 1989, 68-78.

[Harel90] D. Harel et al.,"STATEMATE: A Working Environment for the Development of Complex Reactive Systems," *IEEE Trans. Softw. Eng. SE-16*, 4, Apr. 1990.

[Heitmeyer83] C. Heitmeyer and J. McLean, "Abstract Requirements Specifications: A New Approach and Its Application, *IEEE Trans. Softw. Eng. SE-9*, 5, Sep. 1983, 580-589.

[Heitmeyer90] C. Heitmeyer and B. Labaw, "Software Development for Hard Real-Time Systems," *Proceedings, Seventh IEEE Workshop on Real-Time Operating Systems and Software*, Charlottesville, VA, 10-11 May 1990.

[Heninger80] K.L. Heninger, "Specifying software requirements for complex systems: New techniques and their application," *IEEE Trans. Softw. Eng. SE-6*, 1, Jan. 1980.

[Heninger78] K.L. Heninger et al., "Software requirements for the A-7E aircraft," NRL Rep. 3876, Nov., 1978.

[Jacob86] R. Jacob, "A Specification Language for Direct Manipulation User Interfaces," *ACM Trans. on Graphics 5*, 4, 283-317, 1986.

[Jaffe89] M.S. Jaffe and N.G. Leveson, "Completeness, Robustness, and Safety in Real-Time Software Requirements," Univ. of Calif., Irvine, TR 89-01.

[Jahanian86] F. Jahanian and A. K. Mok, "Safety Analysis of Timing Properties in Real-Time Systems," *IEEE Trans. Softw. Eng. SE-12*, 9, Sep. 1986, 890-904.

[Jahanian88a] F. Jahanian et al., "Semantics of MODECHART in Real Time Logic," *Proceedings, 21st Hawaii Intern. Conf. on System Sciences*, Jan. 5-8, 1988.

[Jahanian88b] F. Jahanian and D.A. Stuart, "A Method for Verifying Properties of MODECHART Specifications," *Proceedings, Real-Time Systems Symposium*, Huntsville, AL, Dec., 1988.

[Jahanian91] F. Jahanian and A. K. Mok, "Modechart: A Specification Language for Real-Time Systems," *IEEE Trans. Softw. Eng.* (to appear).

[Lehoczky89] J. Lehoczky et al., "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proceedings, Real-Time Systems Symposium*, Santa Monica, CA, Dec. 5-7, 1989, 166-171.

[Leung90] J.Y.-T. Leung and C.S. Wong, "Limiting the Number of Late Tasks with Error Constraint," *Proceedings, Real-Time Systems Symposium*, Orlando, FL, Dec. 5-7, 1990, 32-40.

[Ostroff89] J.S. Ostroff, "Real-Time Temporal Logic Decision Procedures," *Proceedings, Real-Time Systems Symposium*, Santa Monica, CA, Dec. 5-7, 1989, 92-101.

[Parnas86] D.L. Parnas and P.C. Clements, "A Rational Design Process: How and Why to Fake It," *IEEE Trans. on Software Eng. SE-12*,, Feb. 1986, 251-257.

[Parnas90] D.L. Parnas and J. Madey, "Functional Documentation for Computer Systems Engineering," TR 90-287, Queens Univ., Kingston, Ontario, Sept. 1990.

[Ramamritham89] K. Ramamritham et al., "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Trans. on Parallel and Distributed Systems 1*, 2, April 1990.

[Rushby89] J. Rushby and F. von Henke, "Formal Verification of the Interactive Convergence Clock Synchronization Algorithm using EHDM," SRI-CSL 89-3, SRI International, Menlo Park, CA, Feb. 1989.

[Stuart90] D.A. Stuart, "Implementing a Verifier for Real-Time Systems," *Proceedings, Real-Time Systems Symposium*, Orlando, FL, Dec. 5-7, 1990, 62-71.

[Tokuda89] H. Tokuda and N. Kimura, "ARTS: A Distributed Real-Time Kernel," *ACM Operating Systems Review, 23*, 3, July, 1989.

[vanSchouwen90] A.J. van Schouwen,"The A-7 Requirements Model: Re-examination for Real-Time Systems and an Application for Monitoring Systems," Queen's Univ., Kingston, Ontario, TR 90-276, May 1990.

RESEARCH AND DEVELOPMENT TECHNICAL REPORT

CECOM-TR-90-2 (CECOM)
C-05043-NY-000100 (CECOM-CSE)

ACQUISITION MODEL FOR THE CAPTURE AND MANAGEMENT OF REQUIREMENTS FOR BATTLEFIELD SOFTWARE SYSTEMS

January 1991

DISTRIBUTION STATEMENT
Approved for public release; distribution is unlimited.

CECOM CENTER FOR SOFTWARE ENGINEERING
US ARMY COMMUNICATIONS-ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703-5000

UNCLASSIFIED

ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION
Unclassified

1b RESTRICTIVE MARKINGS

2a SECURITY CLASSIFICATION AUTHORITY

2b DECLASSIFICATION/DOWNGRADING SCHEDULE

3 DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution is unlimited

4 PERFORMING ORGANIZATION REPORT NUMBER(S)
CECOM-TR-90-2 (CECOM); C-05043-NY-000100 (CECOM CSE)

5 MONITORING ORGANIZATION REPORT NUMBER(S)

6a NAME OF PERFORMING ORGANIZATION
US Army Communications/Electronics Command Center for Software Engineering

6b OFFICE SYMBOL (If Applicable)
See 6c

7a NAME OF MONITORING ORGANIZATION

6c ADDRESS (City, State and Zip Code)
Cdr, US Army Communications/Electronics Command,
ATTN: AMSEL-RD-SE-AST-SE
Fort Monmouth NJ 07703-5000

7b ADDRESS (City, State, and ZIP Code)

8a NAME OF FUNDING/SPONSORING ORGANIZATION
SAME

8b OFFICE SYMBOL (If Applicable)

9 PROCUREMENT INSTRUMENT INDENTIFICATION NUMBER

8c ADDRESS (City, State and Zip Code)

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO	TASK NO.	WORK UNIT ACCESSION NO
63783A/ A094			

11 TITLE (Include Security Classification)
Acquisition Model For the Capture and Management Of Requirements For Battlefield Software Systems (U)

12 PERSONAL AUTHOR(S)
Harlan Black, Editor, with David Leciston, Rinetta McGhee, and John Zimmerlich.

13a TYPE OF REPORT
Technical Report

13b TIME COVERED
FROM Sept. 89 TO January 91

14 DATE OF REPORT (Year, Month, Day)
91-01-15

15 PAGE COUNT
84

16 SUPPLEMENTARY NOTATION

17 COSATI CODES

FIELD	GROUP	SUB-GROUP
12	05	

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Software Requirements; System Requirements; Software Methodology; Requirements Engineering; Rapid Prototyping

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This report presents an acquisition model that meets the needs of new and unprecedented systems that are software intensive, large, complex, and have extensive man-machine interface requirements. When properly applied, it should reduce the cost, schedule, and quality risks that have been associated with these types of procurements. This model is proposed within the context of DOD-STD-2167A and can be tailored to apply to a wide range of acquisitions.

This model acknowledges that requirements have not and perhaps can not be fully and adequately specified up front, prior to acquisition, especially for large and complex systems. Rather, they evolve throughout the system life cycle. It stresses that requirements must be **engineered** and **managed**, not merely written.

The model proposes six risk reduction strategies, which have been previously recommended by numerous DoD studies. This report provides guidance for the Project Manager on their implementation.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT
☑ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL
Harlan H. Black

22b TELEPHONE (Include Area Code)
(908) 532-2238

22c OFFICE SYMBOL
AMSEL-RD-SE-AST-SE

DD Form 1473, JUN 86 Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

TABLE OF CONTENTS

APPENDICES

1.0 EXECUTIVE SUMMARY

This report presents an acquisition model that meets the needs of new and unprecedented systems that are software intensive, large, complex, and have extensive man-machine interface requirements. When applied properly, it should reduce the cost, schedule, and quality risks that have been associated with these types of procurements. This model is proposed within the context of DOD-STD-2167A and can be tailored to apply to a wide range of acquisitions.

Although the immediate audience of this report is the Project Manager, all defense acquisition personnel can benefit from its contents. The intent of this report is to characterize a process model for Requirements Engineering and not to fully specify every detail for its implementation.

The following problems have adversely affected acquisitions: Solicitation and award of a full-scale development contract with incomplete and/or ambiguous requirements; delayed requirements definition and documentation; the appearance of contractual relationships that encourage requirements to increase; and dynamic operational environments where requirements continue to change.

It is acknowledged that requirements have not and perhaps can not be fully and adequately specified up front, prior to acquisition, especially for large and complex systems. Rather, they evolve throughout the system life cycle.

This model stresses that requirements must be **engineered** and **managed**, not merely written. It proposes the following six risk reduction strategies: Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on; contractually decouple requirements definition from the full-scale development effort; establish a functional baseline with an approved System/Segment Specification prior to the solicitation and make the System/Segment Specification a part of the solicitation package; document the user interface and interaction in the System/Segment Specification, together with system testing information; provide structure for the relationship and interaction between the user and the full-scale development contractor for all requirements related matters; and plan to develop systems in an evolutionary manner.

These strategies have been previously recommended by numerous DoD studies. This report consolidates many of their recommendations. It also provides general guidance for the Project Manager on their implementation, from 'Milestone Zero' through the fielding of the last incremental release, presenting the responsibilities and relationships of the primary participants. Appendices provide additional detail, providing information for the acquisition of a Requirements Engineering effort, a proposed format and content of a Requirements Engineering Plan for a typical project, guidelines for the specification of the user interface, a glossary, a bibliography, and a graphical overview of this model, suitable for presentation.

2.0 INTRODUCTION AND BACKGROUND - "WHY"

2.1 Introduction

This document presents a model for the acquisition of software intensive battlefield systems. The model is intended to reduce software life cycle cost, schedule and risks by concentrating on improving the capturing and managing of requirements.

This section contains background information on the need for a new acquisition model.

Section 3 describes the model and provides general guidelines.

Section 4 provides criteria for determining if and when to apply this model.

Section 5 presents specific guidelines for the Project Manager to implement the model.

Section 6 delineates our efforts to insert the model.

The appendices provide additional information for acquisition personnel to implement this model.

2.2 Background

Modern weapon systems are software intensive. That is, they rely heavily on software to provide functionality. These systems are characterized by having extensive user interfaces and interdependence with other systems. They are typically large and complex and they operate in a dynamic environment.

The delineation of requirements for such systems is often incomplete, inconsistent, and specified at varying degrees of detail, all of which significantly contribute to the risk of the development. Some full-Scale Development (FSD) contracts for such systems are awarded with incomplete and ambiguous requirements, as the time and effort needed to improve upon requirements definition is frequently underestimated. Requirement errors are frequently not being discovered until much later in the development and acquisition process, resulting in cost and schedule growth. In addition, there have been systems for which the specification of user interface and interaction detail was delayed until the critical design review, making changes and improvements very costly in dollars and schedule.

Currently, FSD contractors, in their role of requirements capture, keep the government apprised of new capabilities that can enhance the system being developed. The identification of these capabilities may arise either from new technology or from knowledge of the limitations and potential of the system as it matures. Users and their representatives are typically receptive and supportive of additional requirements which they perceive as providing them with more options and functionality. There have been cases where the FSD contractor was in the awkward position of appearing to drive up the system requirements as a result of this relationship.

Finally, some acquisitions plan to develop and field the system in a single step, not allowing new and unforeseen requirements that materialize as the system matures to be easily incorporated, or uncertainties of risks in implementation to be timely dealt with.

3.0 DESCRIPTION OF THE ACQUISITION MODEL - "WHAT"

This acquisition model stresses Requirements Engineering, emphasizing techniques for requirements definition and change management.

The model recommends the following six strategies for risk reduction:

- Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.
- Contractually decouple requirements definition from the FSD effort.
- Establish a Functional Baseline (FBL) with an approved System/Segment Specification (SSS) prior to the solicitation and make the SSS a part of the solicitation package.
- Document the user interface and interaction in the SSS, together with system testing information.
- Provide structure for the relationship and interaction between the user and the full-scale development contractor for all requirements related matters.
- Plan to develop systems in an incremental, evolutionary manner.

These strategies have been recommended by numerous studies and workshops (refer to [1]through [4]). This model consolidates those recommendations that are most applicable to our software intensive battlefield systems.

While this model should reduce the quantity and severity of requirements related problems, it is not envisioned that they will or can be eliminated. We will always have valid needs to change requirements, from such reasons as advances in technology, changes in enemy tactics and capabilities, changes to external systems which must be interfaced with, and insight gained during the system implementation.

The following subsections present these strategies.

3.1 Designate a Requirements Engineering Effort Which Applies Requirements Engineering Techniques from the Early Project Phases and On.

Requirements Engineering is the process of applying engineering disciplines to requirements definition and management.

It is not sufficient to write requirements. Requirements must be **engineered** and **managed**. This model strongly suggests the early designation of a team or effort that is responsible for engineering the system's requirements, the Requirements Engineer (RE).

RE must wear many hats. To the user, the RE is a developer, exploring the feasibility and impact of their requirements and then validating them. To the Project Manager (PM), the RE is a

consultant on requirements and their impact. To the FSD contractor, the RE is the user, who seeks clarification for requirements related questions. The latter is bound to occur, as a significant amount of requirements refinement and clarification occurs during the software design phase

As this function is highly technical and system oriented, it may be appropriate for the project's system engineer to be assigned the lead responsibility.

Staffing the RE team will be a non-trivial and critical task. It is most advantageous for the Government to have its own personnel perform this function directly, and not through a contractor. The Government, itself, must be the one who is the most aware of what it needs, the system requirements. However, this may not always be feasible, due to personnel constraints and it may therefore require contractual support.

One can view the RE as having a role that is similar to an architect of a building project [5].When constructing a building, we prefer to consult with the expert and independent architect regarding our needs and desires, not construction contractors, who may have expertise, but who also benefit from the new work that requirements generate.

The RE should be under the control and direction of the PM and the effort should be initiated prior to the FSD solicitation with the immediate goal of enhancing the FSD procurement package.

From the earliest acquisition planning phases, adequate time must be allocated for the RE effort. The effort, itself, must begin no later than the initial drafts of the Operational and Organizational Plan and Required Operational Concept documents, early on in the project and before commitments by Government and contractors are made.

Although this report is not a tutorial on Requirements Engineering, it should be noted that the RE has a host of tools and techniques at his disposal to symbolically construct aspects of a system and effectively derive and validate the requirements. Technology provides the capability to quickly generate sample screens, interactions, and representative usage of a proposed system. We refer to this technique as prototyping. We also include simulation and modelling in our definition of prototyping.

Prototyping can be effectively used to test the feasibility of both user requirements and possible implementations. Prototypes can be generated to capture and examine user interface and other external interface requirements, communication protocols, functional operations, conditions, constraints, and performance. Prototyping can be used to perform trade-off studies. Prototyping may also shed light on total system acquisition costs. Finally, symbolic construction typically involves designing the symbolic system, where invaluable insight on the requirements and their allocation for the real system is derived.

There is however, a major pitfall with rapid prototypes. Although quick, they are also 'dirty.' That is, they are not always engineered in a way that is efficient or easily maintainable (fixable and changeable). The FSD effort is engineered properly, but the user has to wait for it. A common occurrence when a prototype provides or appears to provide needed capabilities is that the user wants to field it immediately. The PM must make it clear to all who have a stake in the

system, the stakeholders, that the use of a poorly engineered prototype in actual fielded applications is not recommended, nor can such a system be supported.

Reference is provided [6] for additional information on prototyping and other Requirements Engineering techniques.

This model recommends that the RE be involved with requirements related issues throughout the lifetime of the project, not just during its early stages. During system development, the RE should interact with the Combat Developer (CD) regarding proposed changes to the baseline. The RE should interact with end users after initial system fielding to gain their feedback. Relevant activities include prototyping to define or refine requirements for future blocks, risk and feasibility analysis, trade-off studies, requirements change impact analysis, tracing requirements between documents, maintaining the consistency of requirements documents, verification that requirements are being met by the developer, and supporting the PM during reviews and audits.

The PM must carefully assess the requirements for the Requirements Engineering effort and then monitor it carefully. Just as with the FSD effort, the risk of requirements proliferation exists. Unlike the FSD effort though, this effort is on a much smaller scale, reducing risk impact.

3.2 Contractually Decouple Requirements Definition from the Full-Scale Development Effort.

Requirements are a major driving force in acquisition cost and schedule. They should, therefore, be engineered by an independent agent, the RE, and not by the FSD contractor who stands to gain additional work from additional requirements.

A RE contractor should therefore be precluded from the FSD competition and subcontracting.

The FSD contractor should only be responsible for activities beginning with software requirements analysis. This strategy would insure that the design effort commences with a well stated set of requirements.

To minimize the learning curve for the FSD contractor to become familiar with the system's requirements, industry should be kept informed of the acquisition potentials of the system at the earliest possible time. They should also be provided with drafts of all releasable requirements documents, as they become available, as well as prototypes, if appropriate. In the past, comments received from industry during this stage have proven to be invaluable for many projects.

3.3 Establish a Functional Baseline with an Approved System/Segment Specification Prior to Solicitation and Make the Specification a Part of the Solicitation Package.

Solicitation and award of the FSD contract without a firm understanding and agreement with the CD and all stakeholders on the requirements will lead to a contract that lacks firm (or any) cost and schedule commitments.

The recommended strategy is to have the RE write the SSS and conduct the System Requirements Review prior to the solicitation. The approved and validated SSS would then come under Government configuration control and become part of the FBL. The SSS should also become

a part of the solicitation package. In doing so, we will know what we are buying and bidders will know what we really want.

This approach does not eliminate the possibility of changing the requirements during the solicitation period and during the development, with controlled revisions of the SSS. However, it does reduce some of the opportunities for changes with serious impact to occur.

3.4 Document the User Interface and Interaction in the System/Segment Specification, together with system testing information.

As mentioned previously, user interface and interaction details are rarely agreed upon in a timely manner, which greatly impacts cost and schedule. Section 3.2.3 of the SSS format describes the interfaces with external systems. This is an ideal place to provide detail on the man-machine interface and interaction from the user-perspective of the system. A detailed breakdown of the information that is needed for this section is provided in Appendix D.

It should be noted that section 4.0 of the SSS deals with provisions for quality assurance. Test case requirement coverage and general system test philosophy should be specified by the RE in this section. Additionally, the RE may be asked to specify the system requirements test plan and cases in separate documents. For some developments, it may be appropriate for the RE to support or actually perform the testing..

3.5 Provide structure for the relationship and interaction between the user and the full-scale development contractor for all requirements related matters.

As mentioned previously, the relationship between the FSD contractor and the user can unknowingly contribute to requirements growth. This model recommends that the user/ FSD interaction be restricted to the point where there is no appearance of a conflict of interest. For example, the contractor should be restricted from picking up the phone and suggesting new requirements directly with the user. Rather, the user and the contractor should interact with the PM and RE.

As the expert on the system requirements, the RE is a competent representative and advocate for user needs to the developer. As an expert on system development, the RE is able to evaluate feasibility and discuss technical concerns with the user.

The CD should be an active participant in the system's formal reviews. These reviews provide a formal and controlled environment for user-developer interaction. Understandably, interactions such as end user evaluation at the contractor site should not be precluded.

3.6 Plan to Develop Systems in an Incremental, Evolutionary Manner.

Our battlefield systems are often too dynamic and/or complex to field successfully in a single release. It is, therefore very difficult to plan for a system's development in one release, or block.

Plans for the development should call for incremental releases of the system. It is recommended that users prioritize their requirements, listing and rating them by need and by certainty. Requirements that are certain, well understood and that are critical to user/system

functionality should be met in the initial release. These requirements should be specified in detail in the body of the SSS. This initial system must be useful to the user, providing essential capabilities, albeit it is not everything that is needed.

Requirements for subsequent releases must also be documented in the SSS. They can be stated in separate appendices and at this point, do not need the great detail of the initial release's requirements.

Requirements for subsequent releases can become separate options on the FSD contract or they can be separate procurements, depending on the system.

As the RE completes work on a block, the RE should continue to interact with the CD and all system stakeholders to refine and document requirements for subsequent blocks. The end user must provide the RE with feedback from his experience with blocks already in the field.

Just as with the initial release, subsequent releases must be completely defined, validated by all stakeholders, and baselined before commitments are made to implement them.

4.0 MODEL APPLICABILITY -"WHEN"

This section presents the characteristics of systems that would most benefit from applying the strategies of this model. Any one of these characteristics can be sufficient to warrant the use of this model. The model can also be tailored, applying some of its six strategies to broaden its relevance.

As an example, a new and complex Command and Control system would benefit significantly from the application of this model to its acquisition.

4.1 Medium to Large Size.

Systems that are expected to exceed 50,000 source lines of code usually require lengthy development schedules, significant investment in resources (funding, management attention, and manpower), extensive design efforts, and prolonged test and evaluation programs. The potential for overruns due to faulty or deficient system requirements in these programs warrants the use of this model to reduce the technical risk and shorten the development schedule through advanced requirements definition techniques prior to FSD.

4.2 Complex Functionality.

When a system has complex functions, there is a very high risk of cost and schedule growth unless the requirements are defined prior to FSD to the fullest possible degree. Complexity can come from having a large number of user options or having a large number of external interfaces. It can also come from the internal complexity of the software needed to satisfy the functional requirements. This model recommends the application of Requirements Engineering technologies to better understand and specify system requirements.

4.3 Intensive Man-Machine Interface.

Without hands-on user involvement, it is difficult to specify and validate the requirements for systems that have complex, user-dependent, man-machine interfaces. This model recommends rapid prototyping and the early documentation of the user interface and interaction.

4.4 Unprecedented Systems.

Systems which are being developed to provide capabilities that have not been previously available would benefit greatly from this model. As the delivery of the system will probably change the operational environment, users need to work with prototypes early on, to understand potential applications and impacts.

5.0 PROJECT LEVEL MODEL IMPLEMENTATION - "HOW"

This section provides the PM with additional guidance on applying the acquisition model. This is divided into four time frames:

- Milestone 0 to Requirements Engineering Task Initiation.
- Requirements Engineering Task Initiation to Release of the FSD Request For Proposal (RFP).
- FSD RFP Release to Contract Award.
- FSD Contract Award to Final Block Deployment.

This model proposes no changes to procurement strategies before Milestone 0.

Each of the following sections identifies the model activities during these phases and presents responsibilities and relationships of the primary participants.

5.1 Milestone 0 to Requirements Engineering Task Initiation

After Milestone 0, the PM prepares an Acquisition Plan, based on risk analysis, which addresses the degree that the model will be utilized and the needs for Requirements Engineering. The plan should include:

- The block release approach.
- A scope and strategy for Requirements Engineering.
- A proposed source for the Requirements Engineering expertise, either in-house or contract.

If a Requirements Engineering contractor is needed, a cost reimbursable type contract is recommended because the tasks for this effort are difficult to predict. Appendix A contains guidance for acquiring a Requirements Engineering contractor. Appendix B contains technical content for a Requirements Engineering Statement of Work. The latter can also be used when the Requirements Engineering is being done by Government personnel.

Prospective bidders should be requested to document their approach in a Requirements Engineering Plan. After contract award, the Requirements Engineering Plan should become part of the contract. A proposed format and content of this plan is provided in Appendix C.

Responses to a Requirements Engineering RFP should be evaluated based on the bidder's understanding of the technical and operational characteristics of the objective system, the candidate's expertise in Requirements Engineering, and his relevant experience in system development.

5.2 Requirements Engineering Task Initiation to Full -Scale Development Request for Proposal Release

The goal of this phase is to produce a high quality FSD procurement package, with clearly specified requirements and a FBL, documented by a SSS.

Under the direction and control of the PM, the RE develops the FBL. The PM, together with the RE, must identify the areas of requirements related risk. From this analysis, the Requirements Engineering Plan may need to be revised in order to identify the portions of the system that need to be studied, the techniques that should be used, and who should review the products. The PM must insure a disciplined flow of information between the program participants and that all requirements related information is well documented and transferrable. The PM must evaluate the evolving requirements, providing guidance to the RE through frequent interaction and in-progress reviews.

It is recommended that the PM keep industry apprised of the developing RFP, providing them drafts of requirements related documents and products, as appropriate, as well as a draft RFP. Care must be taken to give equal access and opportunities to all prospective bidders.

The RE must engineer the requirements, refining and transforming the requirements from a broad Mission Needs Statement to a validated FBL. In addition, the RE must insure that the requirements are feasible, consistent and testable. Using the best available Requirements Engineering technology, the RE interacts with the CD and/or end user in an iterative fashion, until the requirements are clarified, validated, and refined for a quality SSS.

In all likelihood, the first release will have some, but not all, of the functionality that the end users requested. The PM must work carefully with the RE and all stakeholders as they prioritize requirements for the initial release. The following factors relating to a requirement should be considered:

- Criticality.
- Desirability to the user.
- Implementation risk.

The block release strategy must be reflected in the SSS. The body of the SSS should focus on the initial release. All subsequent block releases must be planned, specified in as detailed a manner as is possible, and incorporated into appendices of the SSS. All requirements in the SSS, as well as proposed incremental versions, must be documented and clearly traceable to source documents.

The SSS is formally validated through the Systems Requirements Review, per DOD-STD-2167A. This review should be hosted by the RE, who authored the SSS. Once accepted, the SSS is placed under configuration control.

The validated SSS becomes part of the FSD RFP, enhancing the procurement package and providing confidence for the commitment of resources needed to build the system.

The CD supports both the PM and the RE by providing expertise or actual end users to meet the Requirements Engineering needs. The CD must review the FSD RFP.

5.3 Full-Scale Development Request for Proposal to Contract Award

To maintain procurement integrity, the SSS must be frozen before the FSD RFP is issued. In a dynamically changing world, this is not always possible and user needs may dictate a pre-award revision. Also, the Government may receive valuable insight from comments from the bidders. If possible, changes should be relegated for inclusion in later incremental blocks. If changes are mandated, all bidders must be given sufficient time after receipt of the updated specification to submit their 'Best and Final Offer.'

Once Block One requirements have been baselined, the RE can begin defining and refining requirements for subsequent incremental blocks, repeating the process used in the identification of the Block One requirements.

5.4 Contract Award to Final Block Deployment

During this phase, the RE tasks during this period include interacting with system stakeholders and the FSD contractor, supporting the initial block's development, and supporting the development of future blocks.

When discussing requirements with a stakeholder, the RE must play the role of the FSD contractor and evaluate requirements feasibility and impact, consulting and/or involving the FSD contractor as necessary. The RE should then provide the PM with a recommendation and an implementation approach. If this new or changed requirement is approved, the RE should document it as an Engineering Change Proposal.

To the developing contractor, the RE is the surrogate user, answering requirements related questions and seeking clarification from the CD when necessary.

If the FSD contractor differs with the RE, then the PM will make the final decision.

The RE can support the PM in the same areas that were suggested for Block One. Because the RE has an in-depth understanding of the system specifications and the development rationale, the PM can use this knowledge to aid in overseeing the FSD effort. The PM can use the RE to assist in ensuring that the system's requirements are being met. The RE may provide support during reviews and may review draft contract data requirements list items. The RE can also be tasked to trace the evolving requirements to source documents. The RE may provide technical support to the project's Configuration Control Board to review proposed changes. The RE may prepare the plans and test cases for acceptance testing and perform the tests.

Future releases incorporate specific requirements which have been identified in the initial SSS and deferred to a later release. They also include requirements that are changes to the initial baseline, resulting from lessons learned from fielded releases. The RE should, therefore, solicit and record user feedback on their experience with the fielded releases.

Engineering, specification, and validation of the requirements for later block releases

proceeds in the same fashion as the first release. However, later blocks have the added constraint that the change must be compatible with fielded blocks.

When a proposed block FBL has been sufficiently defined, the RE should host a new SRR for it. This review should address new requirements, changes to old requirements, and compatibility issues between blocks.

The RE should be responsible for reflecting the evolution of the system by revising the SSS, using Engineering Change Proposals, providing additions and modifications to the sections that refer to the future blocks.

The PM must decide how new incremental blocks should be acquired. The existing FSD contract may make provisions for a block approach. Or, the contract may be modified, based upon new costs and/or schedules. Alternatively, a new contract could be awarded competitively. The existence of a RE gives the PM a resource to help him proceed with any of these strategies. Further, the RE can assist in the development of the necessary procurement documents.

6.0 EFFORTS TO PROMOTE THE MODEL AND GAIN ITS ACCEPTANCE

The acquisition model presented in this report proposes a small change to the current acquisition process. Current policies, regulations, and standards do not preclude the implementation of this model, but they do not encourage it, either.

A step-by-step approach is planned to gain recognition and acceptance of the strategies in this model. Our near term goal is for the model to be implemented and validated on a pilot project at the US Army Communications/Electronics Command (CECOM), monitoring and reporting the technical and financial benefits provided by it on the program.

In addition, we have prepared a Requirements Engineering tutorial to brief CECOM and PEO/PM organizations on the advantages and features of this model.

We also plan to use existing Defense Industry associations and trade journals to disseminate the model and its effect on the system development process. These associations provide an effective medium for educating industry and also provide a forum to obtain industry review and comment on this model.

Finally, CECOM Center for Software Engineering (CSE) is serving as CECOM's focal point for Requirements Engineering related research, development, and implementation. As such, the CECOM CSE would provide the necessary support in implementing this model.

Appendix A

TECHNICAL GUIDANCE FOR REQUIREMENTS ENGINEERING CONTRACT RFP PREPARATION AND EVALUATION

This appendix provides information for a Project Manager for the preparation and evaluation of a Requirements Engineering contract's Request For Proposal. This appendix was not written to be a sample RFP, as contractual guidance, content, formats, and legal interpretation vary between acquisition agencies.

A.1 RFP Preparation

A.1.1 Instructions to Offerors

In their technical proposal, bidders must demonstrate competence and a detailed understanding of the mission environment of the objective system, software engineering, and Requirements Engineering.

The contractors must identify their experience in working with other systems in the same general mission area as the objective system. This experience need not specifically involve requirements derivation. The purpose of the mission area experience is to guarantee a familiarity with the environment, terminology, and philosophy of the intended system.

Prospective contractors should cite specific experience in using Requirements Engineering tools, techniques, and methodologies for the development of the requirements of tactical software systems. They must identify specific projects in which this experience was used. They must identify those tools with which they have specific experience and that are available to support the Requirements Engineering process.

They should demonstrate their understanding of Requirements Engineering techniques by discussing perceived advantages and trade-offs associated with different Requirements Engineering approaches and techniques, as related to the objective system.

The contractors should outline, categorize, and discuss potential aspects of risk of the objective system development and present their approach to deal with them.

Contractors should cite specific experience relating to the development of tactical software for systems that are as complex as the objective system.

A.1.2 Draft Deliverables

Each respondent should submit draft CDRL items with their proposal. These sample deliverables provide insight into the contractor's understanding of the requirements of this effort. They also provide an insight regarding the contractor's capabilities as a RE.

A.1.2.1 Requirements Engineering Plan.

The Requirements Engineering Plan documents the approach that the contractor will take to identify, document, and validate the requirements of the objective system, identifying all activities, schedules, tools, and resources that will be used in the requirements engineering process.

The format and content of a Requirements Engineering Plan is provided in Appendix C of this report. This document should become a part of the Requirements Engineering contract upon its award.

A.1.2.2 Requirements Engineering Notebook Format.

Requirements Engineering Notebooks document the underlying source and rationale of the system requirements, providing the PM with an audit trail from the requirements sources to the Requirements Engineering products and activities. The contractors must propose a structure and format for these notebooks.

A.1.2.3 Configuration Management Plan.

The Configuration Management Plan specifies how the contractor will maintain his developmental baseline control of requirements and prototypes until such time as the specifications are turned over to the Army for formal configuration control. The contractors must provide a draft of this plan.

A.1.3 Corporate Experience

The bidders should provide project profiles which demonstrate their experience in the mission area of the objective system, in software engineering, and in Requirements Engineering.

A.1.4 Personnel/MANPRINT

Bidders should provide a plan for staffing the Requirements Engineering effort. This plan should identify key technical and management personnel. Key individuals should be committed to participation in the effort more than 50 percent of their time and should not be replaced before 90 days after award. Resumes should be included for all key personnel. These resumes should fully document the individual's education and specific experience which is relevant to the effort.

The bidders should provide a staffing plan for obtaining and assigning non-key personnel.

A.1.5 Management

The bidder's management proposal should cover the corporate and project management structure that will be in effect during the effort, including responsibilities, access to resources, quality assurance procedures and subcontract management.

A.2 RFP Evaluation

A.2.1 Government RFP Evaluation Team

The Government must ensure that a qualified team is available to support the evaluation of the Requirements Engineering contractor. Individuals on this team should have experience in the mission area of the objective system, Requirements Engineering, and software engineering. If such individuals are not available within the PM's organization, then the PM must draw support from other organizations for the procurement process.

A.2.2 Evaluation Factors

The following evaluation factors should be considered when planning for the evaluation:

Technical:

- Demonstration of the contractor's understanding and experience in the mission area of the objective system.
- Demonstration of the contractor's understanding of the risk areas associated with the objective system and an appropriate plan for mitigating those risks.
- Demonstration of the contractor's understanding and experience in Requirements Engineering.
- Adequacy and appropriateness of the proposed Requirements Engineering methodology for the objective system.
- Adequacy of the proposed Requirements Engineering Plan.
- Demonstration of the contractor's understanding and experience in software engineering.
- Hardware and software resources availability to perform the work.
- Demonstration of the contractor's capability to write clearly, unambiguously, and concisely.

Personnel/MANPRINT:

- Key and supporting personnel availability who are qualified in the objective system mission area, Requirements Engineering, and software engineering.
- Realism and adequacy of staffing plan for non-key personnel.

Management:

- Effectiveness of project organizational structure and management approach in controlling cost and schedule, and insuring quality.

- Adequacy and feasibility of plan for acquiring supplementary resources, such as subcontractors, if subcontractors are proposed

- Adequacy of the quality assurance approach.

Appendix B

TECHNICAL CONTENT FOR A STATEMENT OF WORK FOR REQUIREMENTS ENGINEERING SUPPORT

This appendix provides technical content for a Statement of Work for a Requirements Engineering contractual effort. This appendix was not written to be a sample RFP, as contractual guidance, content, formats, and legal interpretation vary between acquisition agencies.

The developing agency is referred to as [Agency]. The name of the objective system is referred to as [System].

This appendix was written as broadly as possible, including as many Requirements Engineering functions as possible. It may be tailored to apply to a wide range of acquisitions.

B.1 General Requirements

The RE shall be responsible for the development, refinement, validation, documentation, traceability, and management support of the [System] system requirements and their evolution. He shall interact with the Program Manager (PM), the Combat Developer (CD), the Full-Scale Development Contractor (FSDC), and all system stakeholders. The RE shall be responsible for the preparation of the Systems/Segment Specification (SSS) and the System Requirements Review. The SSS shall fully describe the user interface and interaction needs of [System]. This effort shall continue through the fielding of the final version of the last block release.

The above shall be accomplished in three distinct phases:

- Phase 1 -- Requirements Engineering task initiation through FSD RFP release
- Phase 2 -- FSD RFP release through contract award
- Phase 3 -- Contract award through final fielding

B.1.1 Phase 1 -- Requirements Engineering Task Initiation Through FSD RFP Release

During this phase, the RE shall establish a Requirements Engineering facility and environment. The RE shall structure/revise a cohesive plan for the Requirements Engineering needs for the acquisition and ultimate fielding of [System], performing whatever risk assessments that are necessary. The RE shall implement this plan. The RE shall verify that all requirements are feasible, consistent, testable, and complete. The RE shall interact with the PM, CD and system stakeholders to establish a feasible and acceptable system Functional Baseline (FBL) of [System], documenting it in the SSS. The RE shall host the [System] System Requirements Review (SRR). The RE shall prepare and submit technical reports and construct prototypes as required. A Requirements Engineering Notebook shall be developed and maintained, documenting the development and evolution of the system requirements. The RE shall support the incremental block release strategy for [System] that is approved by the PM and CD.

B.1.2 Phase 2 -- FSD RFP Release Through Contract Award

Throughout this phase, the RE shall maintain the SSS, effecting necessary minor revisions, prior to the receipt of best-and-final offers for contract award for the initial block of [System]. The RE shall also begin refining and documenting system requirements for subsequent blocks, utilizing the procedures employed and interfaces defined for Block 1 requirements.

B.1.3 Phase 3 -- FSD Contract Award Through Final Fielding

Throughout this phase, the RE shall assist the PM in monitoring the FSD of the [System] for requirements compliance. The RE shall identify discrepancies and anomalies between the system requirements and the system in development, recommending appropriate resolution when possible. The RE shall trace the evolving requirements to source documents. The RE shall create the requirements testing plan, cases and he shall perform the actual testing. The RE shall provide the PM with insight on the cost, schedule, and quality impacts of requirements changes and evolution. The RE shall be responsible for the documentation of new or changed requirements through Engineering Change Proposals (ECPs) to the FBL.

The RE shall maintain close liaison with the CD and available end users to identify any new or potential changes to the system requirements. He shall provide them with insight on the feasibility and impact of new requirements or requirements changes.

The RE shall provide technical support to the project's Configuration Control Board to review proposed changes. He shall support the CD in preparation of plans for field testing.

The RE shall answer the requirements related questions of the FSDC.

B.2 Detailed Requirements

The RE shall devise and implement processes for managing and performing all Requirements Engineering activities. Required engineering processes shall include, but may not be limited to the following:

- Develop/ purchase and maintain a Requirements Engineering environment.
- Assess and manage risk.
- Perform system requirements analysis and studies.
- Develop System/Segment Specification.
- Perform System Requirement Review.
- Trace requirements.
- Assess requirement changes and evolution.
- Create the requirements test plan, cases, and perform the testing.
- Monitor FSDC compliance with requirements.
- Interact with the FSDC.

- Perform system requirements analyses for block releases.

(Note: These processes may overlap and/or be applied iteratively/recursively.)

B.2.1 Develop and/or Purchase and Maintain an Automated Requirements Engineering Environment

With [Agency] approval, the RE shall develop or purchase Requirements Engineering software and hardware as required to identify and elicit requirements; evaluate requirements feasibility and alternatives; document requirements; trace requirements to source documents; and communicate with the CD and PM in requirements refinement and validation. The RE shall maintain the software and hardware associated with the Requirements Engineering environment throughout the life of the contract and they shall be delivered to the [Agency] at contract completion.

B.2.1.1 Configuration Management

The RE shall perform configuration management of the Requirements Engineering environment in compliance with the guidance found in MIL-STD-483 and supplements thereto.

B.2.2 Assess and Manage Risk

The RE shall establish and implement procedures for risk assessment that effectively identify, analyze, monitor, and mitigate areas of the [System] procurement that involve potential requirements related cost, schedule, or quality risks. They shall be documented in technical reports.

The RE shall establish and implement procedures for controlling risk to include:

a. Identifying the risk areas of the procurement risk factors in each area.

b. Assessing the risk factors identified, including the probability of occurrence and the potential impact to cost, schedule, and quality.

c. Identifying and analyzing the alternatives available for reducing the risk factors.

d. Proposing the most promising alternative for each risk factor.

e. Obtaining feedback to determine the success of the risk reducing action for each risk factor.

B.2.3 Perform Systems Requirements Analysis and Studies

B.2.3.1 Requirements Engineering Planning

The RE shall develop/revise and maintain plans for the conduct of all activities required by this SOW in a document entitled the Requirements Engineering Plan (REP). The format and content of this plan is provided. (See Appendix C.)

B.2.3.2 Apply Requirements Engineering Techniques to the Development of System Requirements

The RE shall perform and document trade-off analyses of alternatives for optional requirements.

The RE shall apply Requirements Engineering techniques and technology to develop the FBL, from mission needs statements and other high level requirements documents to its validation. The RE shall elicit and confirm the system requirements with all system stakeholders. The RE shall insure that all requirements are feasible, consistent, and testable.

The RE shall propose and develop a block release strategy for the system. The block release strategy shall ensure that the initial block release provides a set of capabilities that is approved by the PM and CD.

B.2.3.3 Requirements Engineering Notebooks

The RE shall document all Requirements Engineering efforts, participants, information, and sources of information in the Requirements Engineering Notebook which shall be in a format proposed by the RE and subject to [Agency] approval. The RE shall utilize this notebook to provide an audit trail of his activities.

B.2.4 **Develop System/Segment Specification**

B.2.4.1 System/Segment Specification

The RE shall prepare and maintain the SSS for the [System] which shall document all planned block releases. The initial SSS shall completely specify the requirements for the block 1 release and it shall be adequate for the solicitation and subsequent development of the first block of [System]. This SSS shall identify, in separate appendices, each subsequent block release to the level of detail possible at this time. The RE shall subsequently maintain the SSS to reflect the evolving requirements of this system.

B.2.4.2 User Interface and Interaction Specification

The RE shall identify and document the user system interface and interaction requirements at the same level as interfaces to external systems. Proposed content for this section is provided. (See Appendix D.)

B.2.4.3 Test Architecture

Section 4 of the SSS shall propose a test architecture for the system quality assurance provisions. The architecture shall be developed in concert with the designated test organization and shall address the system test philosophy. The architecture shall identify the resources that shall be required for system testing, including operational and test hardware and software. The architecture shall include the test evaluation criteria below:

- Traceability

- Consistency with requirements.
- Adequacy of test cases/test procedures.

B.2.5 Perform System Requirements Review (SRR)

The RE shall conduct the SRR in accordance with MIL-STD-1521. The RE shall present the SSS at the SRR for validation. The RE shall ensure that all changes approved at the SRR are incorporated in the revised SSS.

When incremental block FBL's are sufficiently defined and ready for implementation, the RE should host the SRR for it. *This review shall address new requirements*, changes to old requirements, and compatibility issues between blocks.

The RE shall provide the PM with minutes for all SRR's.

B.2.6 Trace Requirements

The RE shall trace all requirements from the high level source documents initially provided to the SSS. This tracing shall be implemented in an automated database and shall be expandable to include design related detail. (The RE shall trace the requirements to the following FSD deliverables: ...)

The RE shall provide traceability of requirements changes.

B.2.7 Assess Requirements Changes and Evolution

The RE shall evaluate all proposed changes identified by the PM/CD as to their completeness, accuracy, consistency, and testability and shall apply Requirements Engineering technology as necessary to refine the requirements. The RE shall assess the cost, schedule, and quality impact of the changes and make recommendations on whether the changes should be included in the current block release or a subsequent block release.

B.2.8 Create the requirements test plan, cases and perform the testing.

The RE shall create the requirements test plan and test cases for all releases that are delivered by the FSDC. They shall be documented in test plan and description documents per DI-MCCR-80014A and DI-MCCR-80015A. The RE shall perform all requirements related tests. The RE shall document the results of the tests in a test report, per DI-MCCR-80017A.

B.2.9 *Monitor FSDC Compliance With Requirements*

The RE shall support the [Agency] in the review of the FSD effort to verify that the design and implementation is consistent with approved requirements. The RE shall review all FSDC deliverables for requirements compliance; and attend progress review and formal design review meetings. The RE shall identify discrepancies and anomalies between the system requirements and the system in development, recommending appropriate resolution when possible.

B.2.9.1 Corrective Action Process (CAP)

The RE shall establish and maintain a CAP for all requirements related problems detected in items under development or configuration control. This process shall be closed-loop, ensuring that all detected problems are promptly reported and entered into the ECP process, actions are initiated on them, resolutions are achieved, status is tracked and reported, and records of the problems are maintained in the Requirements Engineering Notebook for the life of the contract. The RE shall prepare reports, as required. The RE shall classify each problem identified by category (i.e., requirements, code, design, etc.) and by priority (i.e., high, medium, and low); and perform analyses to detect adverse trends in the problems reported. The RE shall closely monitor the FSD effort to verify that problems have been resolved, adverse trends have been reversed, changes have been correctly implemented in the appropriate FSDC processes and products, and no additional problems have been introduced.

B.2.10 **Interact with the FSDC**

The RE shall support the FSDC by answering requirements related questions, seeking clarification from the CD when necessary. The RE shall report all queries and his responses to the PM.

B.2.11 **Perform System Requirements Analysis for Block Releases**

The RE shall apply Requirements Engineering techniques, tools, and methodologies to define, refine, and document the evolving requirements of block releases in concert with the PM and CD. The RE shall solicit and record user feedback on their experience with fielded releases to clarify and define the evolving requirements. The RE shall revise the SSS, using ECPs to reflect the evolving requirements. Later blocks have the added constraint that they must be compatible with fielded blocks. The RE shall repeat the relevant activities in this SOW for each block release.

B.3 **Reporting**

B.3.1 Monthly Status Reports

The RE shall submit monthly status reports identifying the status of the development of the requirements; the areas addressed; the stakeholders that have been contacted; the requirements that have been identified; issues that have been clarified; and any problems that have been encountered. This status report shall also contain information on personnel assigned during the reporting period; personnel expected to be assigned next reporting period; travel completed this period; travel anticipated next period; costs during this period, cumulative through this period, and projected for next period; and anticipated activities for the next period.

B.3.2 Quarterly Progress Reviews

The RE shall conduct quarterly progress reviews describing all the efforts of the previous quarter. All areas addressed in the Monthly Status Reports shall be discussed, but on a quarter-wide basis.

B.4 Deliverables

B.4.1 Prototypes, Models, Simulations, and Tools

By the end of this contract, the RE shall deliver all prototypes, models, simulations, and/or tools generated in the process of developing the [system] requirements. They shall be delivered, with documentation, as proposed by the contractor and approved by [Agency], and with unlimited rights to the Government. The RE shall ensure that the systems and documentation, as delivered, are sufficient to allow each item to be installed and executable on a commercially available Government owned host computer. The RE shall insure that sufficient documentation and special purpose hardware/software are provided to enable the Government to run and modify all prototypes. [Note: Serious consideration should be given as to whether the need for acquiring the above justifies the potential costs.]

All hardware and software that was purchased by the RE with contract funds shall be delivered to the [Agency] at the end of this contract.

B.4.2 System/Segment Specification

The RE shall finalize and deliver the SSS for the [System]. The initial SSS shall be presented at a System Requirements Review for validation. The revised SSS shall be delivered for use in the [System] acquisition. The RE shall maintain this document throughout the life of this contract, providing the PM with the latest version upon request.

B.4.3 Requirements Test Plan

The RE shall develop a requirements test plan for all releases of the objective system. It shall be in the format of DI-MCCR-80014A. These plans shall be provided (# months) prior to the Governments's acceptance testing of the FSDC system releases.

B.4.4 Requirements Test Description

The RE shall specify the requirements related test cases. They shall be documented in the format of DI-MCCR-80015A. They shall be provided (# months) prior to acceptance testing of the FSDC system releases.

B.4.5 Requirements Test Report

The RE shall document the results of all requirements related tests that he performs. They shall be in the format of DI-MCCR-80017A. They shall be provided within (#) days of the tests.

B.4.6 Requirements Engineering Notebook

The RE shall develop and maintain the Requirements Engineering Notebook, providing an audit trail on the development of the [System] requirements. This notebook shall be available to the PM at any time for inspection, reference, and reproduction. This notebook shall be delivered to the [Agency] at the end of the contract.

B.4.7 Technical Reports

The RE shall provide RE technical reports as required in a mutually agreed upon format.

B.4.8 Minutes of the SSR

The RE shall provide the PM with minutes of all SRR's that are hosted.

B.4.9 Monthly Status Reports

The RE shall submit Monthly Status Reports.

B.5 Constraints

By selection as the RE for the [System], the RE shall be precluded from bidding on future FSD efforts for the [System]. In addition, the RE shall be precluded from subcontracting to perform work for the FSD effort.

Appendix C

REQUIREMENTS ENGINEERING PLAN FORMAT AND CONTENT

This appendix provides the format and content of a Requirements Engineering Plan. This plan provides the Requirements Engineering approach and identifies all activities, schedules, tools, and resources that will be used in the Requirements Engineering effort.

Since the Acquisition Model can be tailored, not all sections of this document may be applicable to a specific project.

C.1 Requirements Engineering Activities

C.1.1 Systems Requirements Analysis.

This specifies the approach for performing the objective system's requirements analysis.This should discuss the plans and schedule for interaction with the stakeholders throughout the life cycle of the effort.

C.1.2 Risk Assessment.

This section addresses the specific procurement risk assessments that will be performed for the objective system.

This should provide a plan for mitigation of risk for each risk area, specifying specific methods, techniques, and tools.

C.1.3 Alternative Concepts and Trade-off studies.

This section addresses the specific studies that are needed. This will include the aspects of the system that will be analyzed and the criteria that will be used in making trade-off decisions between conflicting requirements.

C.1.4 Systems Requirements Review.

This portion of the plan discusses the approach for the Systems Requirements Review. This should include criteria for passing the review and a plan for the validation of the functional baseline which will be formed as a result of this review.

C.1.5 Objective System RFP Enhancements.

This section identifies the products that will be developed to enhance the RFP for the objective system. This should minimally include the SSS and a recommended tailoring of the SOW and CDRLs of the objective system.

C.1.6 Cost, Schedule, and Impact Estimation

This specifies the approach for preparing cost, schedule, and quality impact analyses for proposed changes to requirements during the development of the objective system. This should discuss the measurement of the uncertainty of the estimates, based upon the uncertainty of the requirements. This section should also describe the approach for coordinating the development and data management efforts to ensure interface compatibility and maintainability.

C.1.7 Requirements Tracing.

This section identifies the method and techniques that will be used to trace the requirements.

C.1.8 FSD Contractor Monitoring Support.

This section identifies the approach and extent of support for FSD contract monitoring to insure that the objective system is in compliance with system requirements.

C.1.9 Change Management.

This section identifies the approach for supporting the management of changes to requirements. It should include change request procedures, tracking change requests and their implementation, and trend analysis.

C.1.10 Systems Analysis for Block Releases.

This section presents the approach for performing system analysis for requirements during incremental development.

C.1.11 System Requirements Testing

This section presents the approach for planning the system requirements tests, specifying the test cases, and participation in the actual testing.

C.2 Requirements Engineering Techniques

C.2.1 Requirements Engineering Tools

This section identifies the methodologies and tools that will be used for the development, validation, documentation, and management of requirements. This section should discuss how and when they will be used. This section should also provide the rationales for their selection. Tools should be described in terms of vendor, function, operating procedures, operational requirements, and products. When applicable, the interaction and integration of divergent tools should be discussed.

C.2.2 Prototyping.

This section should present the prototyping approach. It should address the approach for identifying those aspects of the system that need to be prototyped. This section discusses the standards for the development, documentation, and delivery of any prototypes developed during this effort.

C.2.3 Requirements Engineering Notebook.

This section proposes the format and content for the Requirements Engineering Notebook. The notebook should be constructed in such a way as to provide a traceable audit trail for the development and evolution of the system requirements for all Requirements Engineering activities and products.

C.3 <u>Requirements Engineering Management</u>

C.3.1 Configuration Management Plan.

This section identifies the plan for configuration management of requirements and Requirements Engineering products, such as documents and prototypes. This should address version control and cross referencing between versions of Requirements Engineering products; identification procedures; problem and change reports and review boards; configuration status accounting; audits; authentication procedures; and major milestones.

C.3.2 Resources, Organization and Staffing.

This section identifies the resources, organization, and staffing plan of the Requirements Engineering effort, describing the RE's facilities; Government-furnished equipment and services required; and organization, personnel, and resources for Requirements Engineering.

C.3.3 Total Quality Management (TQM).

This section discusses the process for establishing and performing Total Quality Management of the Requirements Engineering tasks. TQM requirements, procedures, evaluations, metrics, internal controls, and reports utilized should be specified. This should also address the software development file; associated access and control procedure; and procedures and reports used to prepare for formal reviews.

C.3.4 Technical Status Reviews and Reporting.

This section specifies the approach for providing detailed status reviews and reports for this effort. This must include, at a minimum, problems encountered, technical approaches, technical status, plans for future work, requirements, cumulative and projected costs, and schedule.

C.3.5 Evaluation, Testing, and Standards

This section discusses the approach for evaluation of Requirements Engineering products, implementation of a quality evaluation reporting system, format of all test documentation, corrective actions plan, and design and coding standards.

Appendix D

USER INTERFACE SPECIFICATION

This appendix provides user interface and interaction requirements for a typical C2 system which should be addressed in the SSS. This includes all visual, audio, and tactical interfaces and interactions.

D.1 User Interface Hardware

A description of specified capabilities of the target user interface hardware for the system shall be specified, including quantitative parameters such as bandwidth, response and access time, and storage capacity. The interface hardware shall include but not be limited to the following:

- Number of independent monitors supported
- Description of monitor, such as size, number of pixels, colors bit planes etc.
- Description of keyboard, such as number of function keys, keypad, etc.
- Description of mouse or trackball, such as number of buttons
- Audible signal capability
- Description of other man-machine interface related hardware, such as scanners, touch-panels, special displays, etc.

D.2 Screens

D.2.1 Formats

This shall define every class of screen presentations, including menus. The formats shall clearly identify screen zones and their purpose. The screen definitions shall include but not be limited to the following:

- Use of multiple windows. If used, how many, window moving and sizing, when is a new window created, etc.
- Screen format, including reserved zones, placement and format of classification markings, placement of titles, icons, menus, buttons, error messages etc.
- Window format, including reserved zones, placement and format of classification markings, placement of titles, icons, menus, buttons, error messages, etc.
- Use of color and their meaning, including standard colors for all screen components
- Use and type of fonts for every class of presentation.

- Vocabulary form and standard words to use for each occasion. For example, using DONE to indicate job is completed vs. OK or EXIT

- Help facilities, indicating the standard method for invoking help, and the format and contents of the help displays

- Format and usage of menus, buttons, scroll bars, icons, etc.

- Standard interaction sequences, such as a requirement to confirm every data base change, or the ability to undo or cancel some operations after they have been initiated or completed

D.2.2 Presentations and Flows

Pictures of all screen presentations and all logic flow between them shall be specified, including back-out and selection sequences. Interaction between the state of the system and screen flow shall be specified.

Screen refresh requirements shall be specified, indicating whether data appearing on screen represents information accurate only at the time the screen was generated or whether there exists a requirement to continually update presented screen data. Refresh rates shall be specified, if needed.

Density of screen content shall be specified, defining what constitutes operator overload thresholds that mandate screen declutter. Declutter techniques shall be specified.

D.2.3 Data Elements

The individual data elements which appear on the screen shall be specified, providing the meaning, significance, units, and/or data type of each data element, and the source of each data element whether calculated or retrieved from internal or external sources.

Appendix E

GLOSSARY OF TERMS AND ACRONYMS

CAP - CORRECTIVE ACTION PROCESS

CD - COMBAT DEVELOPER.

Command or agency that formulates doctrine, concepts, organization, material requirements, and objectives. For the US Army, this is the Training and Doctrine Command (TRADOC). May be used generically to represent the user community role in the material acquisition process.

END USER

The command or agency which will ultimately be the recipient and/or operator of a system under development.

FSD - FULL-SCALE DEVELOPMENT

Norm[illegible] phase in the material acquisition process during which a system, including all items necessary for its support, is fully developed.

FSDC - FULL-SCALE DEVELOPMENT CONTRACTOR

FBL - FUNCTIONAL BASELINE

See DoD-STD-480.

INCREMENTAL DEVELOPMENT

A software system development process where the user requirements are not fully known before acquisition. The system is developed in a series of partial implementations. Each implementation is used to clarify and refine the requirements for the next implementation.

MILESTONE 0

Program Initiation/Mission-need Decision, approved by the Defense Acquisition Board (DAB) or designated authority, which determines mission-need and approves program initiation and authority to budget for a new program. Normally, a concept exploration/definition phase follows this approval.

OBJECTIVE SYSTEM

The system under consideration for development for which Requirements Engineering is needed.

PEO - PROGRAM EXECUTIVE OFFICER

Individual responsible for administering a defined number of major and/or non-major acquisition programs who reports to and receives direction from the Army Acquisition Executive.

PM - PROJECT MANAGER

Individual chartered to conduct business on behalf of the Army who reports to and receives direction from either a PEO or the Army Acquisition Executive and is responsible for the centralized management of a specified acquisition program.

REQUIREMENTS

Requirements are the quantifiable and verifiable behaviors that a system must possess and constraints that a system must work within to satisfy an organization's objectives and solve a set of problems.

RE - REQUIREMENTS ENGINEER

A functional entity comprised of Government and/or contractual personnel that performs the Requirements Engineering tasks required by the PM.

REP - REQUIREMENTS ENGINEERING PLAN

REQUIREMENTS ENGINEERING

Requirements Engineering is the disciplined application of scientific principles and techniques for developing, communicating, and managing requirements. See Appendix B.

RFP - REQUEST FOR PROPOSAL

STAKEHOLDERS

All commands, agencies, or personnel who are directly concerned or affected with the outcome of a system acquisition. Stakeholders may include the end user, the developing agency, post deployment software support centers, the test and evaluation agencies, operational commanders, logistics support agencies, and many others, depending on the system.

SSS - SYSTEM/SEGMENT SPECIFICATION

A system level requirements specification whose format is specified in DoD Data Item Description DI-CMAN-80008A.

SRR - SYSTEM REQUIREMENTS REVIEW

UNPRECEDENTED SYSTEM

A system which does not parallel a system which has been previously developed. This may be due to the planned use of new technologies, a new mission-need, a need that has never been met, or a significant increase over previous system capabilities and performance.

Appendix F

BIBLIOGRAPHY

[1] Beam, Walter R. et al, "Adapting Software Development Policies to Modern Technology," Washington,DC:National Academy Press, 1989.

[2] Black, Harlan H. et al, "The Technical Cooperation Panel (TTCP) Requirements Engineering and Rapid Prototyping Workshop Proceedings," US Army CECOM Center for Software Engineering Technical Report C-0103400000100, May 1990.

[3] Beam, Walter R. et al, "Adapting Software Development Policies to Modern Technology," Air Force Studies Board, 1989.

[4] Hess, James A. et al, "Report of the AMC Software Task Force,", US Army Materiel Command, February 1989.

[5] Sumrall, George E., "Requirements Engineering and Ada," Proceedings of the 6th National Conference on Ada Technology, March 1988.

[6] Davis, Alan M., "Software Requirements: Analysis and Specification," Englewood Cliffs, NJ:Prentice Hall, March 1990.

Appendix G

GRAPHICAL OVERVIEW OF THE ACQUISITION MODEL

The CECOM CSE Acquisition Model Stresses Requirements Engineering, Emphasizing Techniques For Requirements Definition And Change Management.

It Recommends Six Risk Reduction Strategies, Which Can Be Tailored To Apply To A Wide Range Of Acquisitions.

ACQUISITION MODEL FOR THE CAPTURE AND MANAGEMENT OF REQUIREMENTS FOR BATTLEFIELD SOFTWARE SYSTEMS

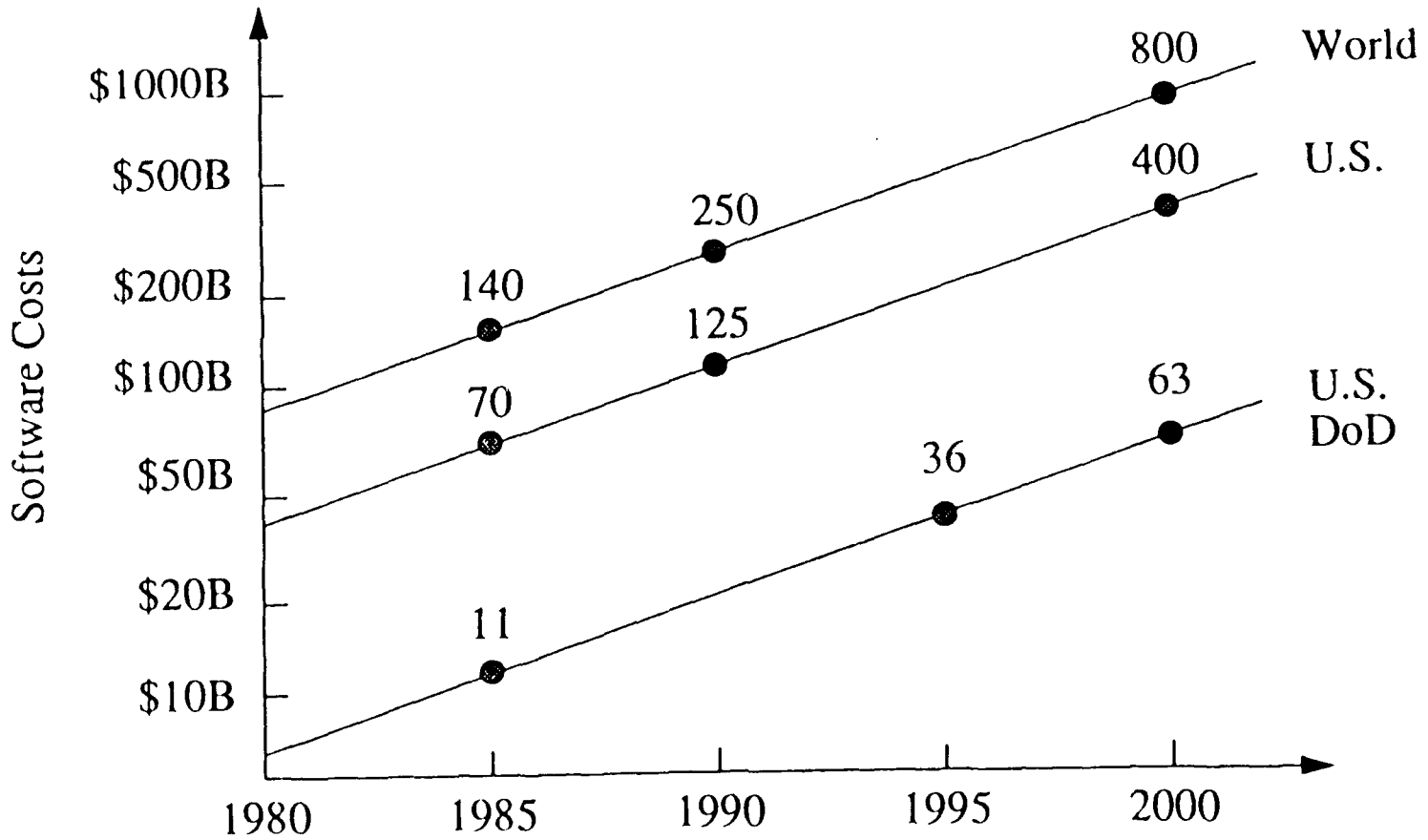
ORDER OF PRESENTATION

- The Problem And Its Significance
- Proposed Approach - An Acquisition Model
  - Six Risk Reduction Strategies
  - Relevance Of The Model To US Army System Acquisition
  - Application Of The Model To US Army System Acquisition
- Plans For Implementation

The Problem And Its Significance

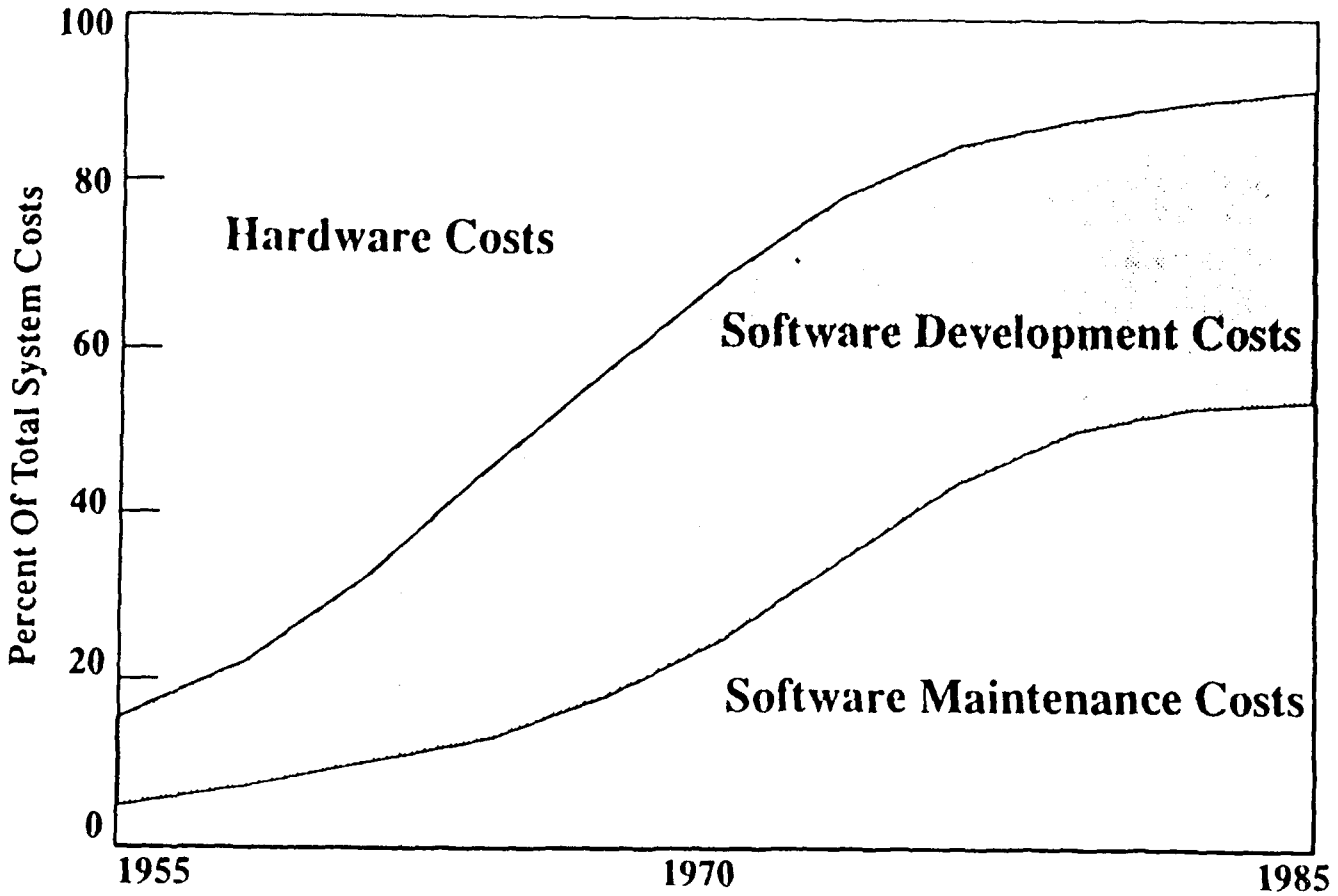
- The Problem And Its Significance
- Proposed Approach - An Acquisition Model
  - Six Risk Reduction Strategies
  - Relevance Of The Model To US Army System Acquisition
  - Application Of The Model To US Army System Acquisition
- Plans For Implementation

Software Needs Are Great And They Are Growing Exponentially



Source: Barry W. Boehm, "Improving Software Productivity", IEEE Computer Sept'87

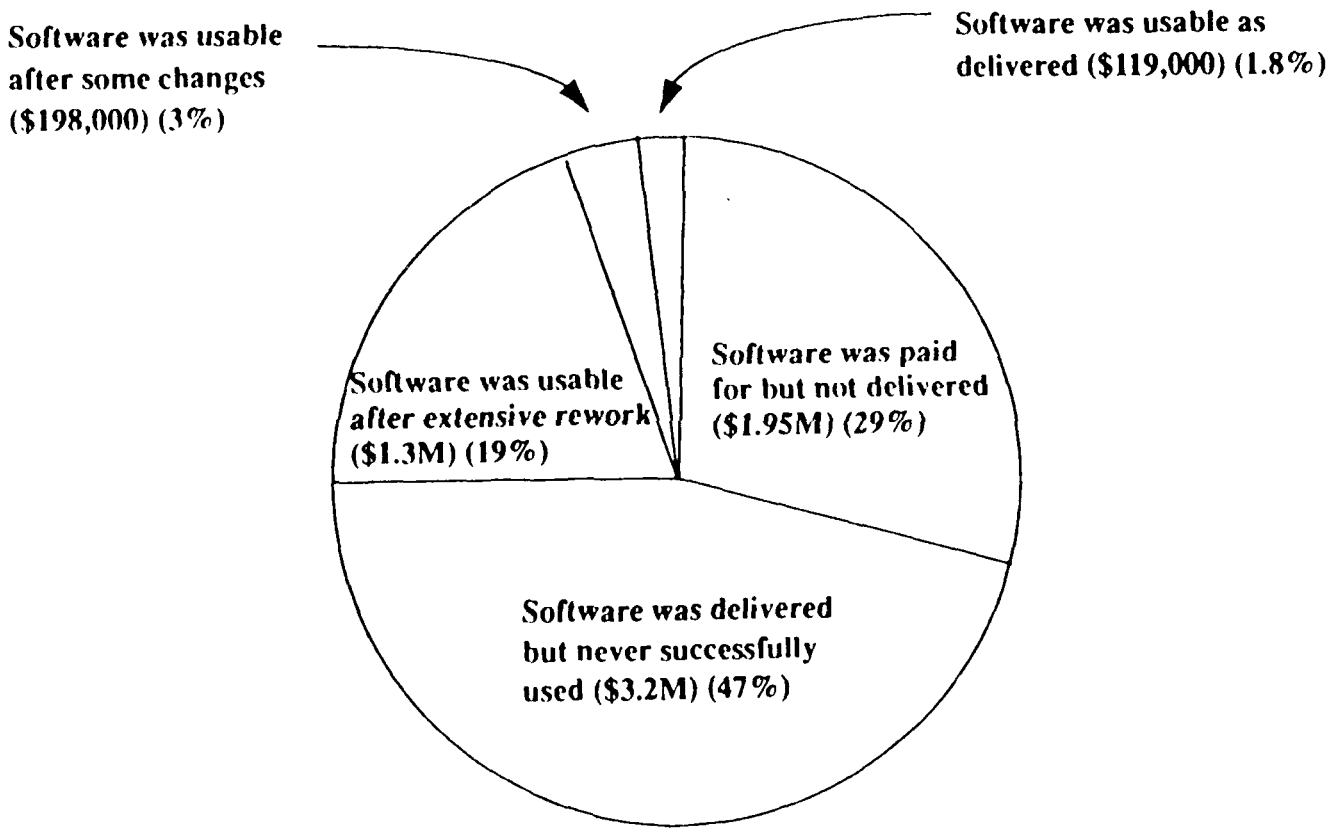
The Cost of Software Dominates Total System Costs



Software Versus Hardware Cost Trends

Source: Barry W. Boehm, "*Software Engineering Economics*", New Jersey: Prentice Hall Inc.

Software Problems Have A History And A Reputation



1979 GAO Survey of nine acquisitions totaling $6.8M identified major problems in contracting for software.

Intensive Efforts Were Devoted To Finding The Causes Of Software Acquisition Problems.

Requirements Related Problems Were Frequently Involved.

"In nearly every software project which fails to meet performance and cost goals, requirements inadequacies play a major and expensive role in project failure." [1]

"There are four kinds of problems that arise when one fails to do adequate requirements analysis: (a) top-down design is impossible, (b) testing is impossible, (c) the user is frozen out, and (d) management is not in control." [2]

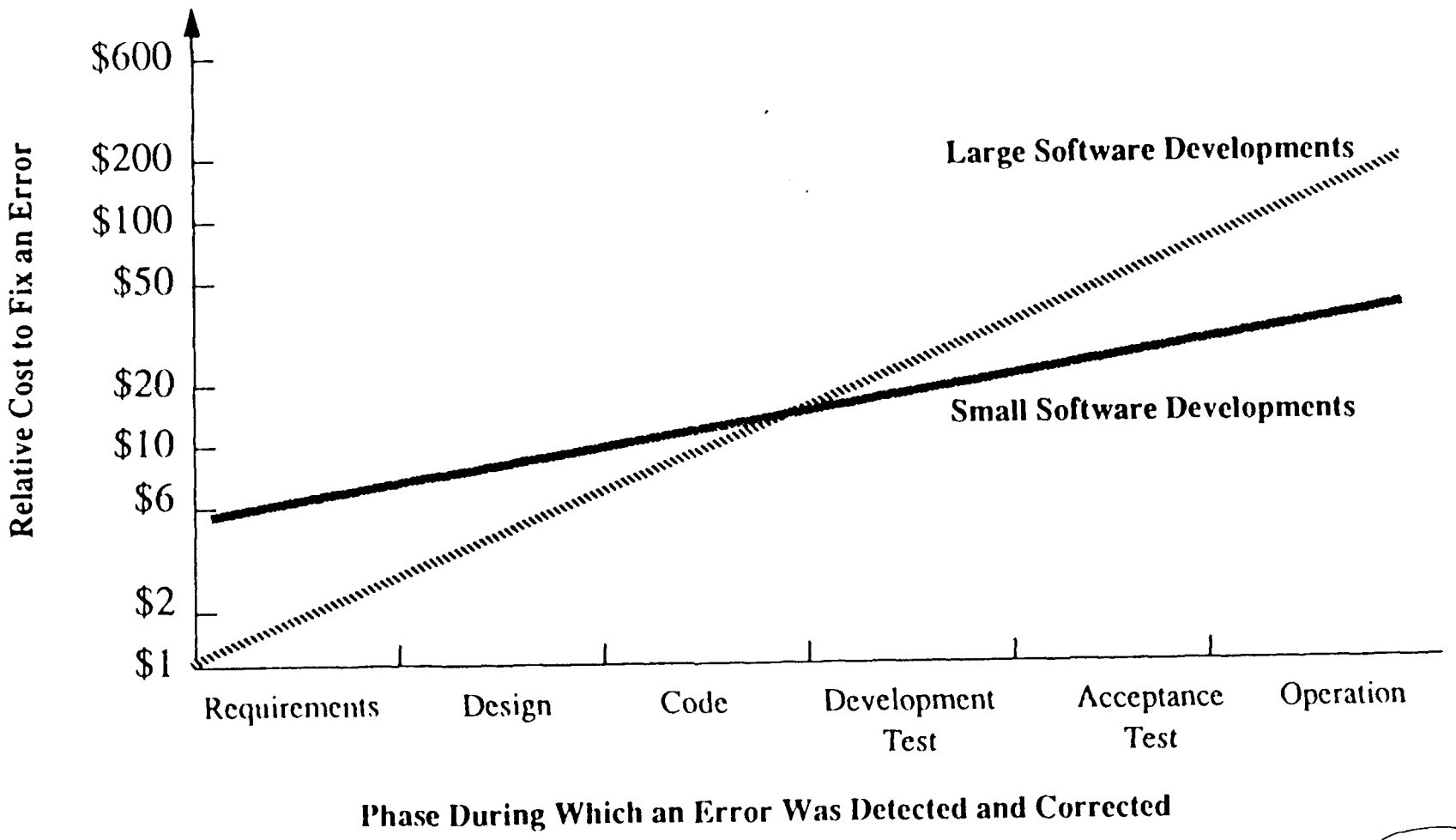
Development of the requirements specification "in many cases seems trivial, but it probably is the part of the process which leads to more failures than any other." [3]

[1] M. W. Alford and J. T. Lawson, *Software Requirements Engineering Methodology (Development)*, (USAF RADC-TR-79-168).

[2] W. W. Royce, "Software Requirements Analysis" in *Practical Strategies for Developing Large Software Systems*, E. Horowitz, editor.

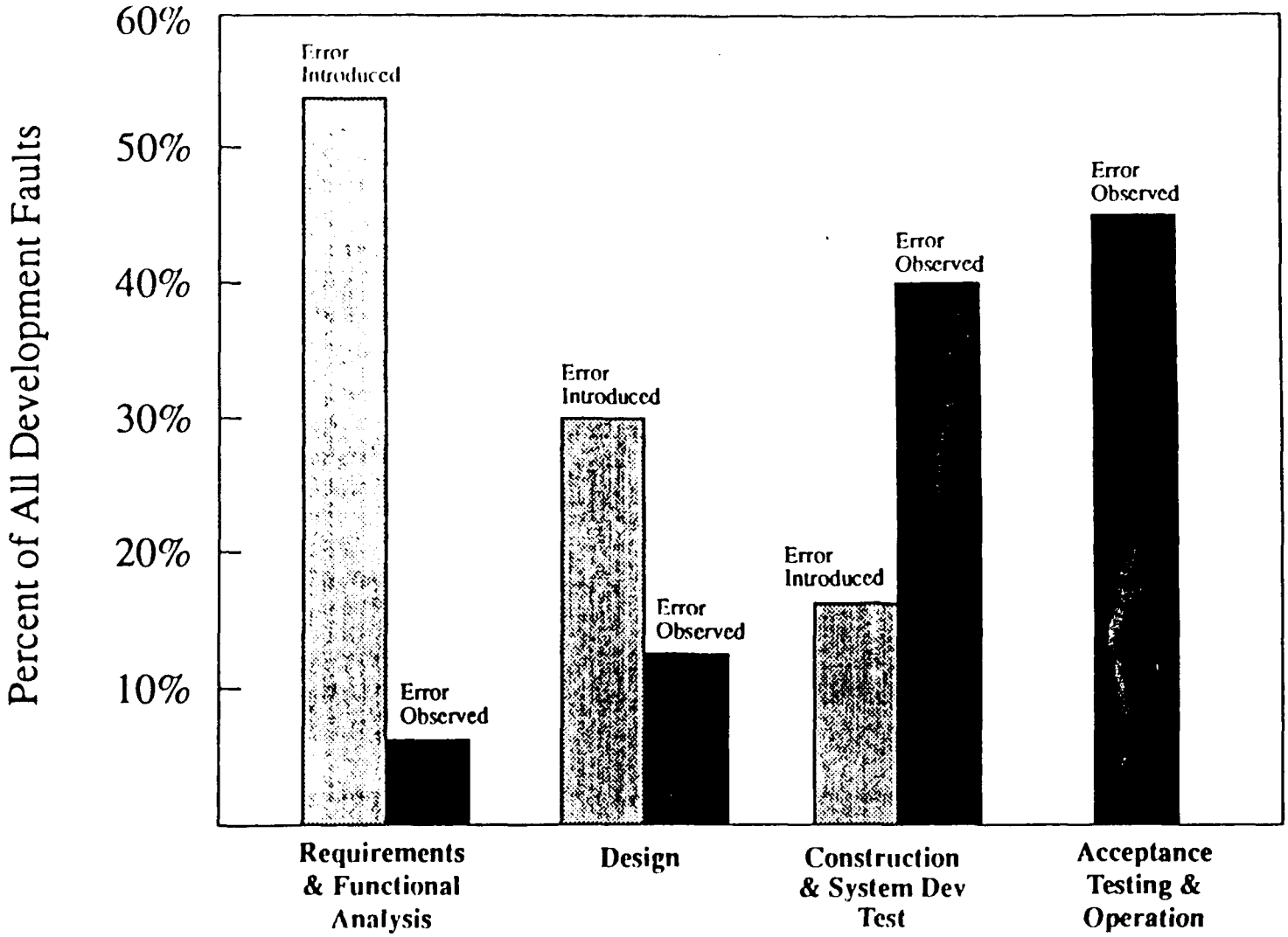
[3] J. L. Schwartz, "Construction of Software: Problems and Practicalities," in *Practical Strategies for Developing Large Software Systems*, E. Horowitz, editor.

The Cost Of Solving Requirements Related Problems Increases Drastically With The Time Of Detection



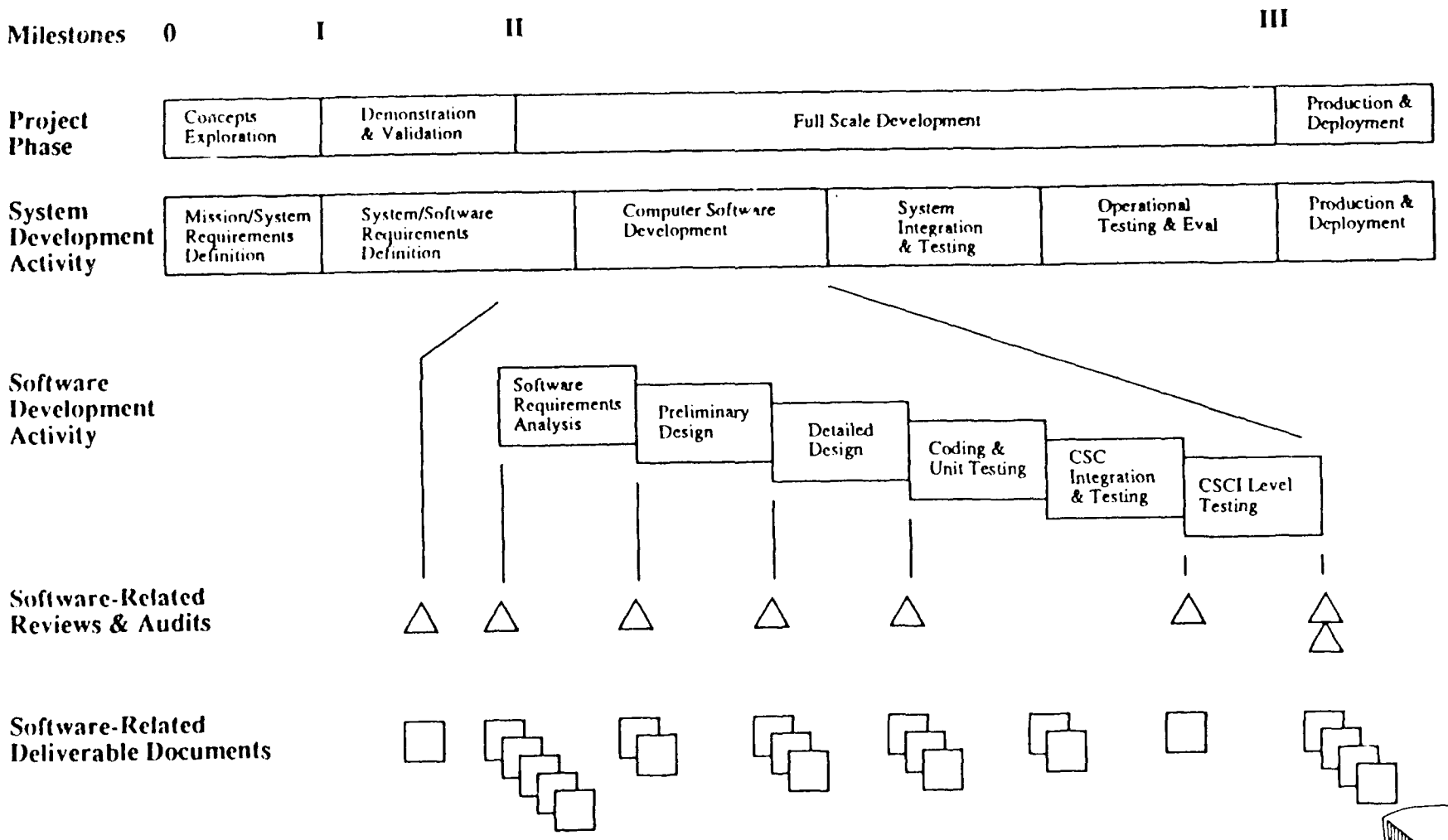
Source: Barry W. Boehm, *Software Engineering Economics*, New Jersey: Prentice Hall, Inc.

Most Software Faults Are Introduced In The Requirements & Functional Design Phase.

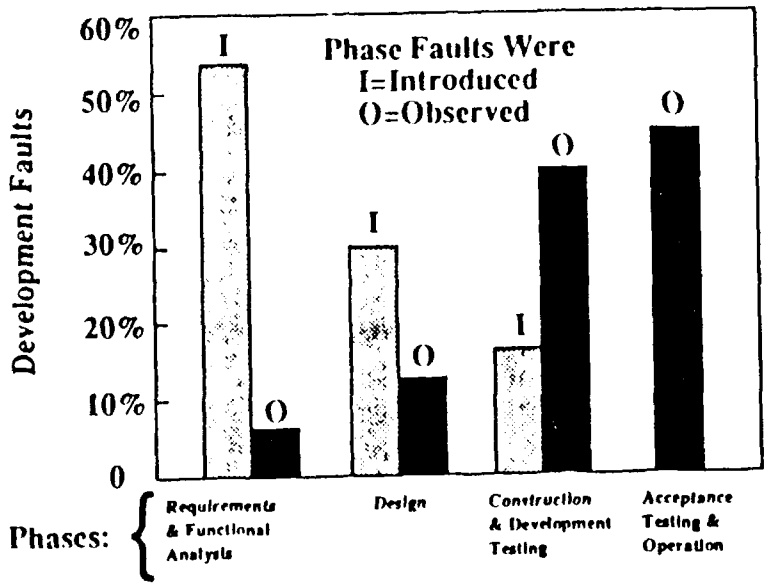


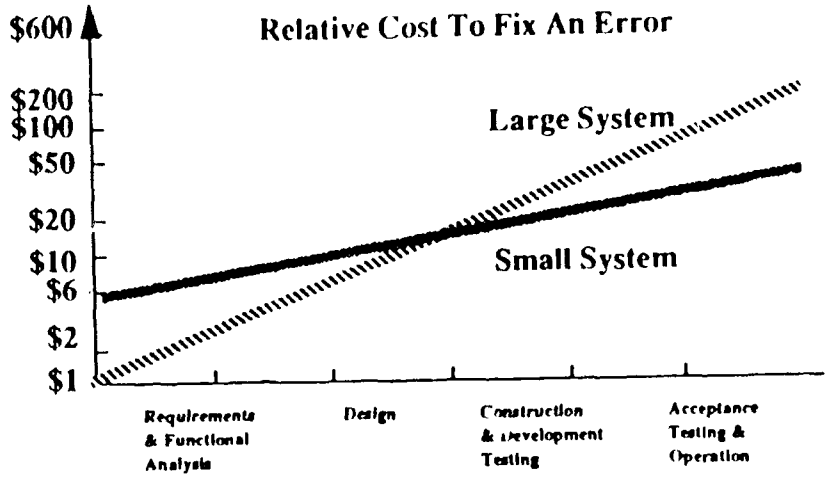
Source: "Software Engineering,", Ramamoorthy, et al., IEEE Computer, 10/84

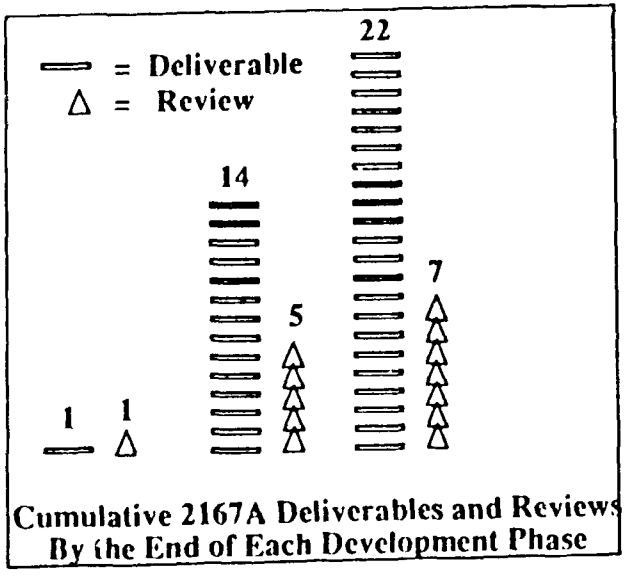
Defense System Software Development Establishes Requirements For Software Development, Mandating An Extensive Sequential Review Process And List Of Deliverables.



DoD-STD-2167A

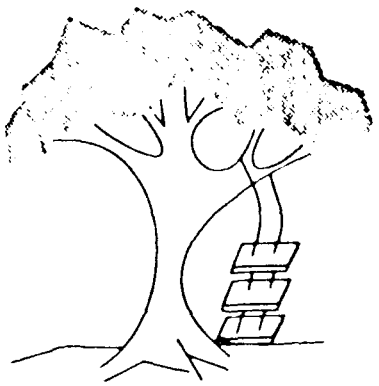




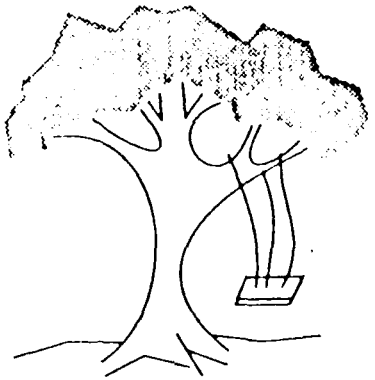


Latent System Requirements Defects May Cause Massive Rework, Substantially Impacting Cost, Schedule, And Quality.

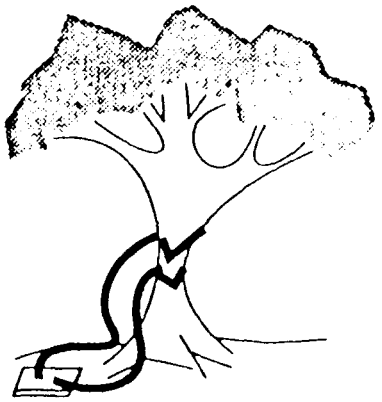
Developing Large-Scale Systems Is A Tough Job! Despite Everyone's Good Intentions And Expertise, Things Don't Always Turn Out Right.



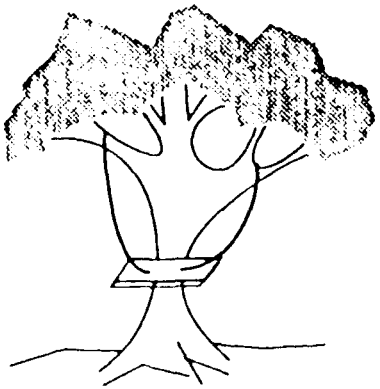
What The RFP ally Asked For



What The Contract Said Would Be Delivered

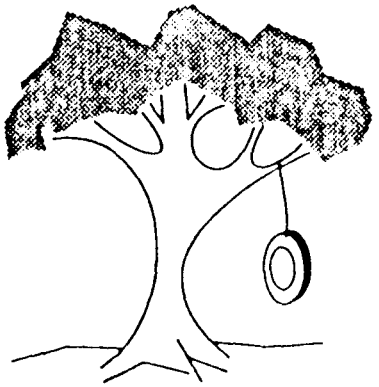


What Engineering Designed



What Was Built

How It Got Tested



What The User Really Wanted

Proposed Approach - An Acquisition Model

- The Problem And Its Significance
- Proposed Approach - An Acquisition Model
  - Six Risk Reduction Strategies
  - Relevance Of The Model To US Army System Acquisition
  - Application Of The Model To US Army System Acquisition
- Plans For Implementation

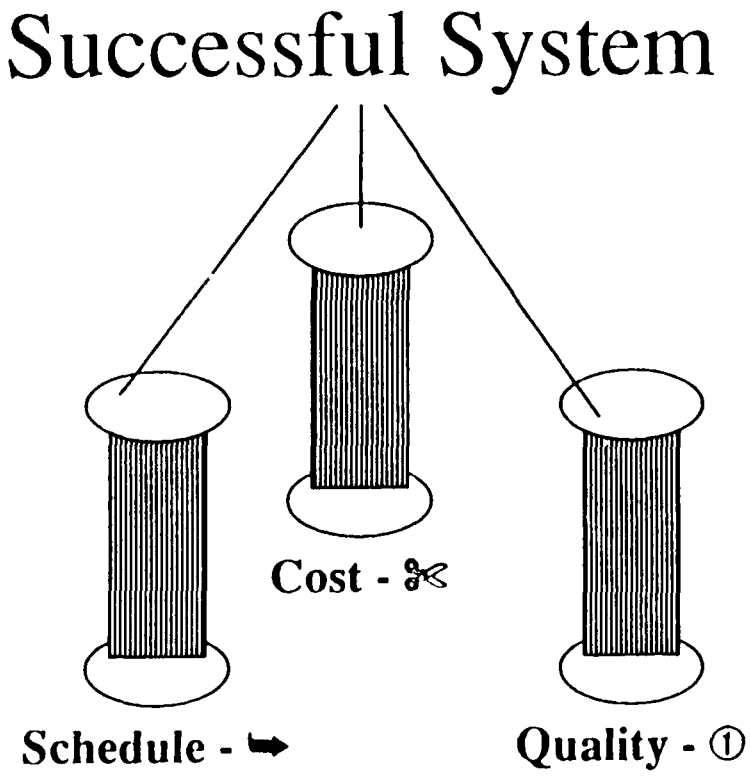
Six Risk Reduction Strategies

1. Engineer The Requirements
2. Decouple Requirements Definition
3. Functional Baseline Before RFQ
4. User Interface & Testing In SSS
5. Provide Structure For FSD Relationship
6. Evolutionary System Development

Preface #1:

The US Army CECOM Center For Software Engineering Recommends Six Strategies For Software Intensive Acquisitions.

When Applied Properly, They Should REDUCE Requirements-Related RISKS



These strategies have already been recommended by numerous DoD studies and workshops.

While this model should reduce the quantity and severity of requirements related problems, it is not envisioned that they will or can ever be eliminated. We will always have valid needs to change requirements, from such reasons as advances in technology and changes in enemy tactics and capabilities.

Preface #2:

WE CAN NO LONGER AFFORD TO *WRITE* REQUIREMENTS.

RATHER, REQUIREMENTS MUST BE *ENGINEERED* and *MANAGED*.

∃ΩΣ∫∉ ΠΦφΠ אϑ∴θρτψυιπασδφγ

Requirements are the quantifiable and verifiable behaviors that a system must possess and constraints that a system must work within to satisfy an organization's objectives and solve a set of problems.

Requirements Engineering is the disciplined application of scientific principles and techniques for developing, communicating, and managing requirements.

Requirements Engineering deserves recognition as a branch of the engineering sciences.

Civil Engineering

Electrical Engineering

Industrial Engineering

Mechanical Engineering

Requirements Engineering

Software Engineering

The CECOM CSE Acquisition Model Stresses Requirements Engineering, Emphasizing Techniques For Requirements Definition And Change Management.

It Recommends Six Risk Reduction Strategies, Which Can Be Tailored To Apply To A Wide Range Of Acquisitions.

The Six Strategies Are As Follows:

1. Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.
2. Contractually decouple requirements definition from the Full Scale Development Effort.
3. Establish a Functional Baseline with an approved System/Segment Specification (SSS) prior to the solicitation and make the SSS a part of the solicitation package.
4. Document the user interface and interaction in the SSS, together with system testing information.
5. Provide structure for the relationship and interaction between the user and the full-scale development contractor for all requirements related matters
6. Plan to develop systems in an incremental, evolutionary manner.

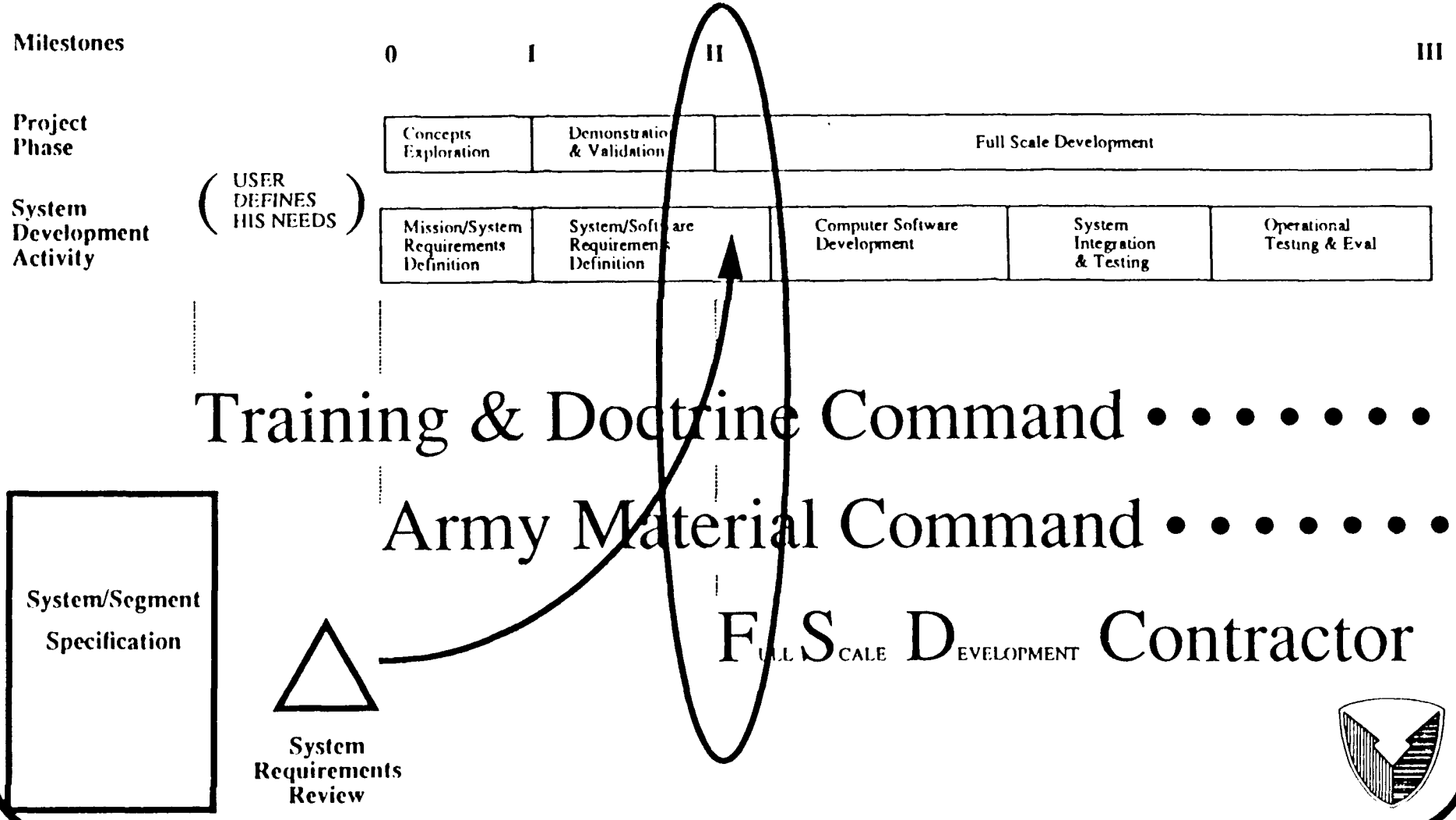
Relevance Of The Model To US Army System Acquisition

- The Problem And Its Significance
- Proposed Approach - An Acquisition Model
  - Six Risk Reduction Strategies
  - Relevance Of The Model To US Army System Acquisition
  - Application Of The Model To US Army System Acquisition
- Plans For Implementation

The Full Scale Development Contractor Is Typically Responsible For Requirements-Related Tasks

Besides Constructing the System, He Helps To Define It.

But, If The System Needs Significant Definition, How Are We Able To Engage The Development Contractor, Beforehand?



The Army Material Command, The Software Buyer, Must Translate User Requirements Into Acquisition Documents.

This Task Requires Significant Expertise.

Milestones		0	I	II		III
Project Phase		Concepts Exploration	Demonstration & Validation	Full Scale Development		
System Development Activity	(USER DEFINES HIS NEEDS)	Mission/System Requirements Definition	System/Software Requirements Definition	Computer Software Development	System Integration & Testing	Operational Testing & Eval

PARTICIPANTS

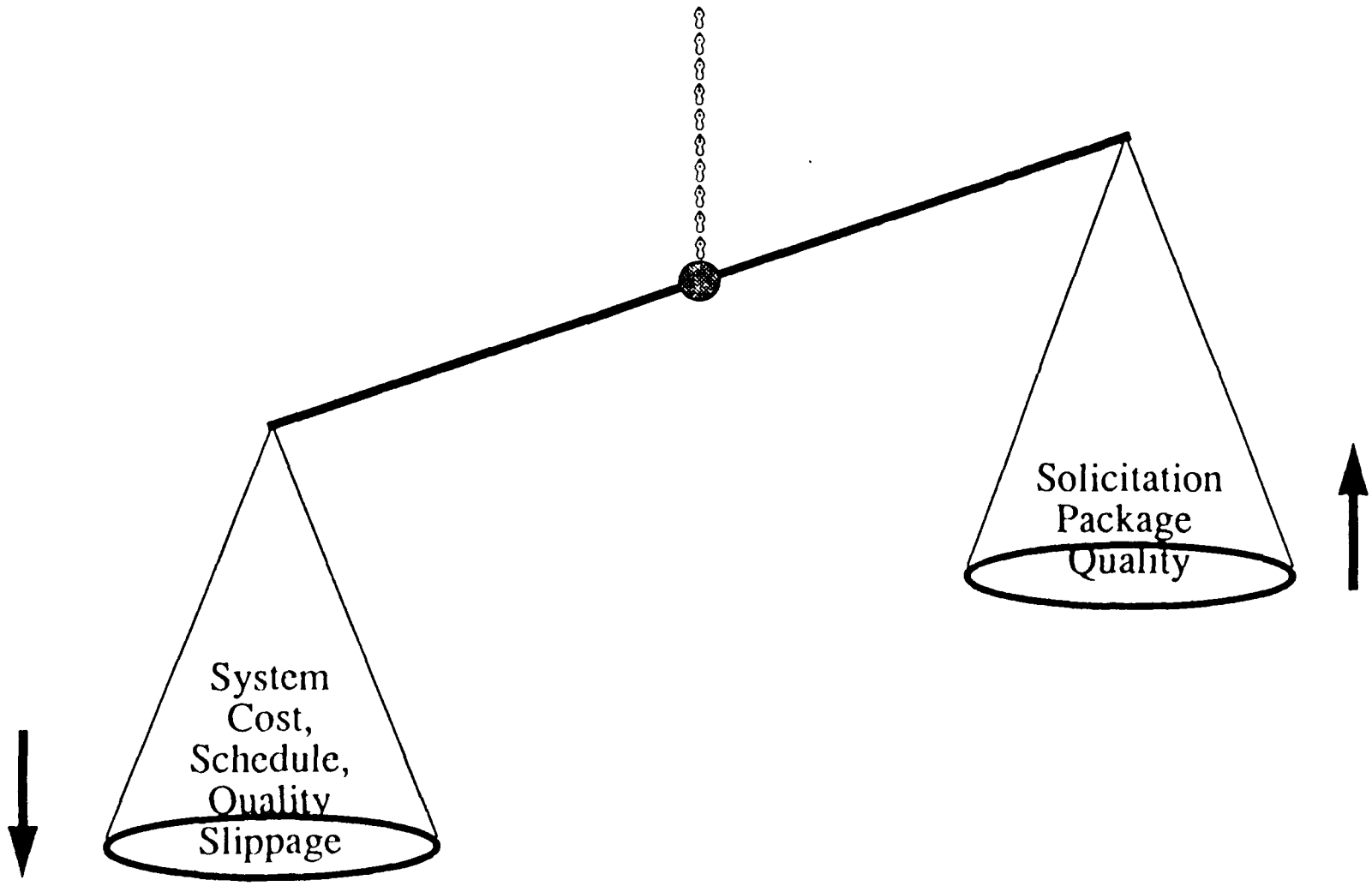
Training & Doctrine Command • • • • • • •

Army Material Command • • • • • • •

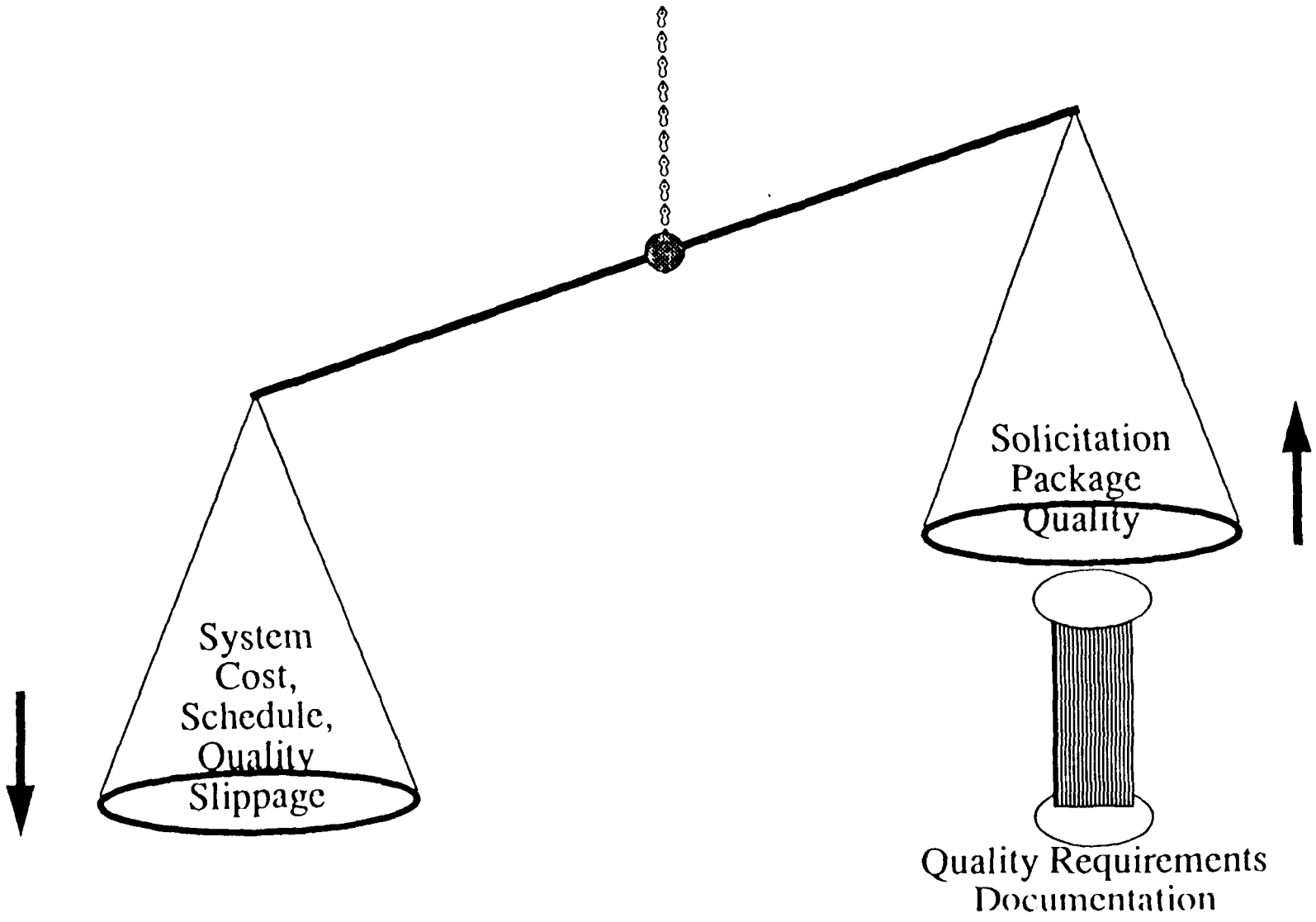
Full Scale Development Contractor

USER (Rep)
BUYER
BUILDER

We Can Significantly Improve The Acquisition Process By Improving The Quality Of The Solicitation Package.

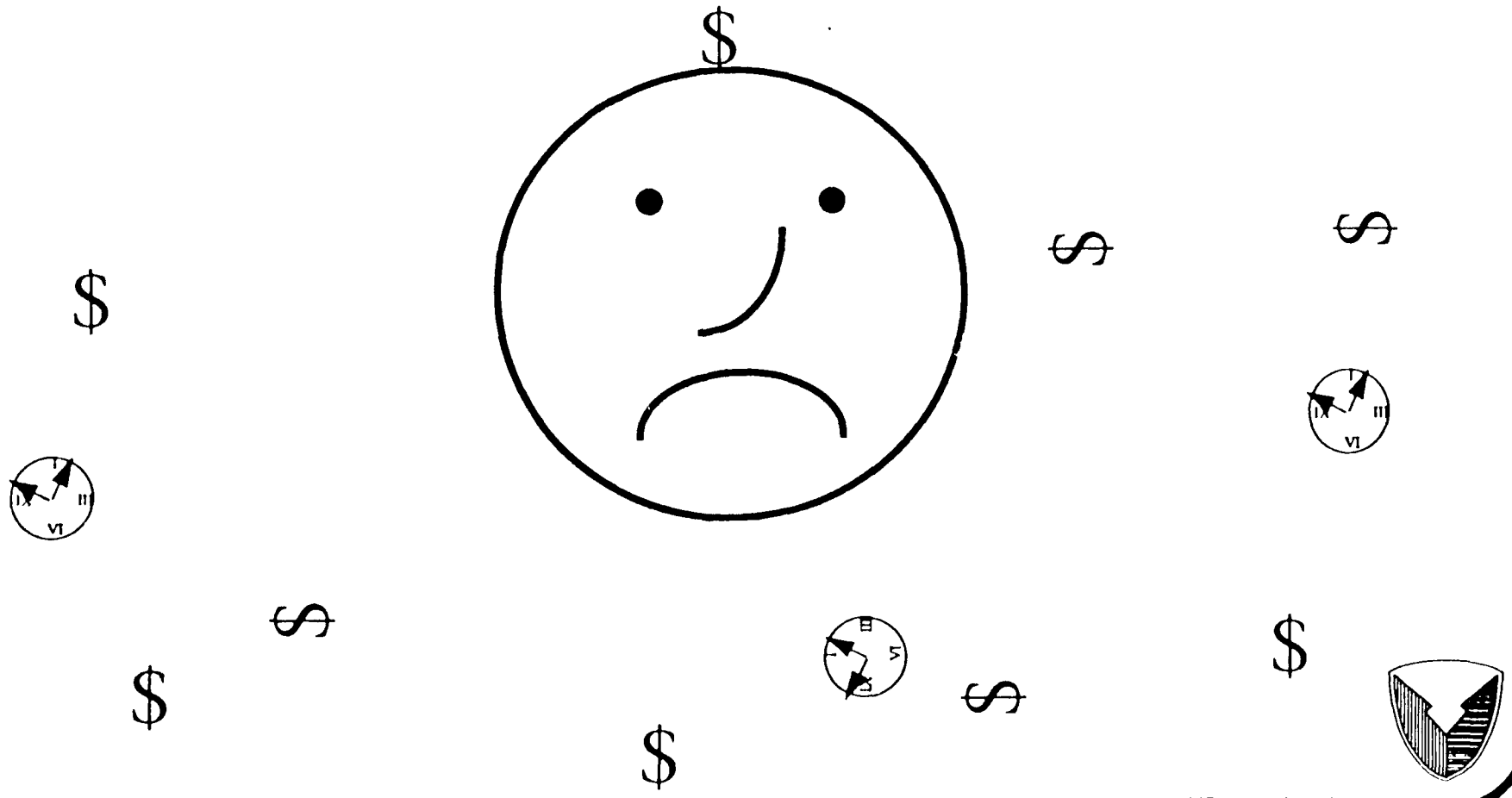


Quality Solicitation Packages Need Quality Requirements Documentation.

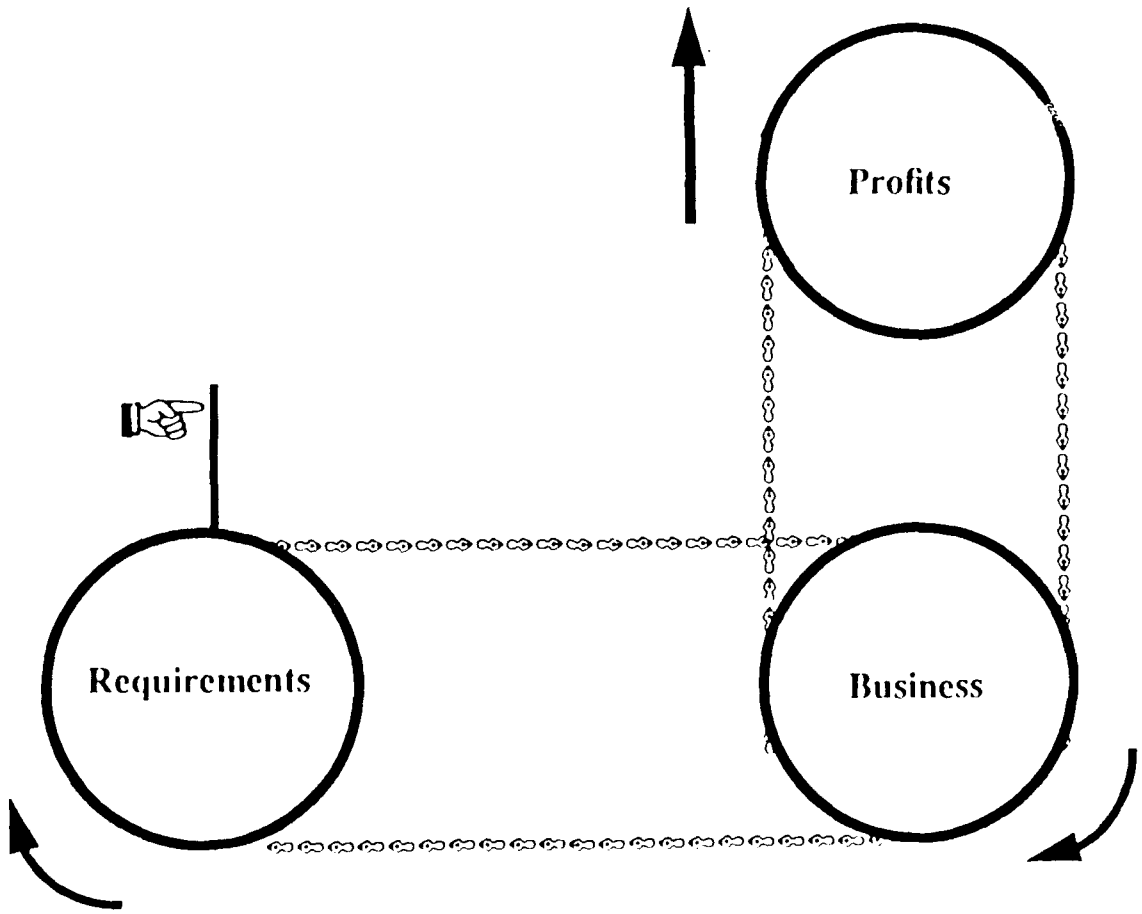


Sadly, Crucial Details, Such As The User Interface, Are Not Always Specified In Detail Until The Critical Design Review (!)

By The Time The System Reaches That Stage, Changes And Improvements Are Very Costly In Cost And Schedule



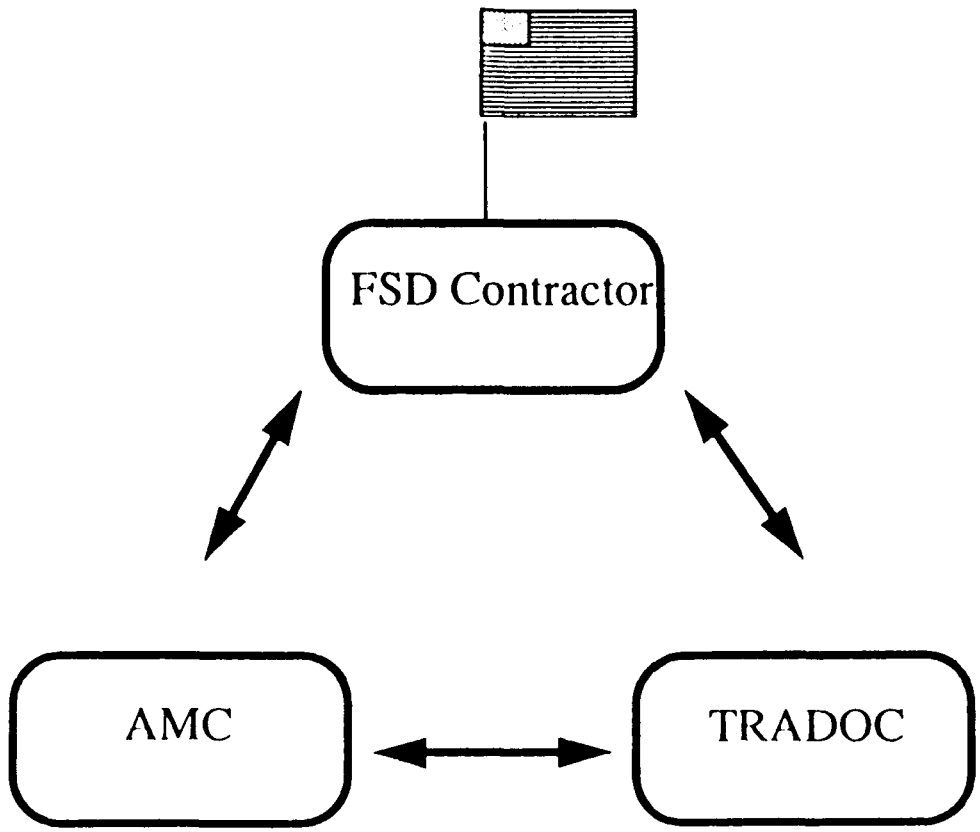
The Requirements Related Role That Has Been Given To The FSD Contractor Can, At Times, Have The Appearance Of A Conflict Of Interests.

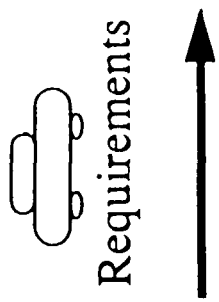


To Serve The Customer And The Country, The FSD Contractor Must Keep The Government Apprised Of New Capabilities That Can Enhance The System.

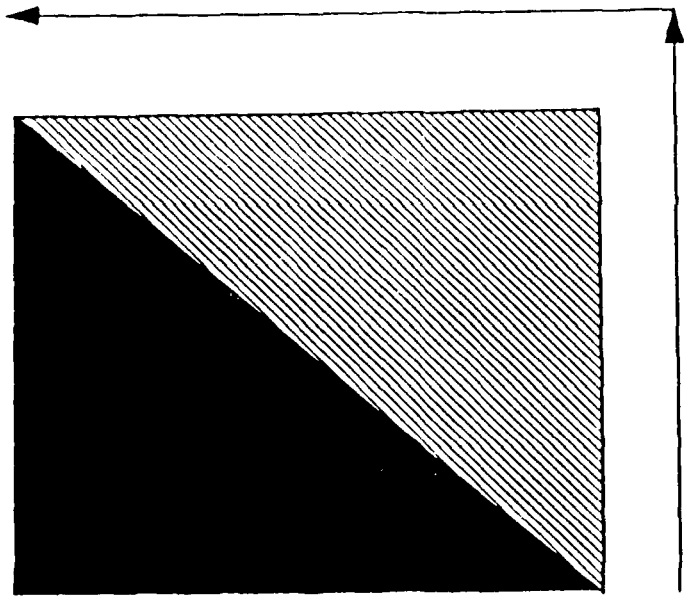
Users Are Typically Receptive And Supportive Of Additional Requirements, Which They Perceive As Providing More Options And Functionality.

In The Complex FSD Relationship, The Contractor Has Ample Access To The User And Is In An Awkward Position Of Appearing To Drive Up The Requirements.

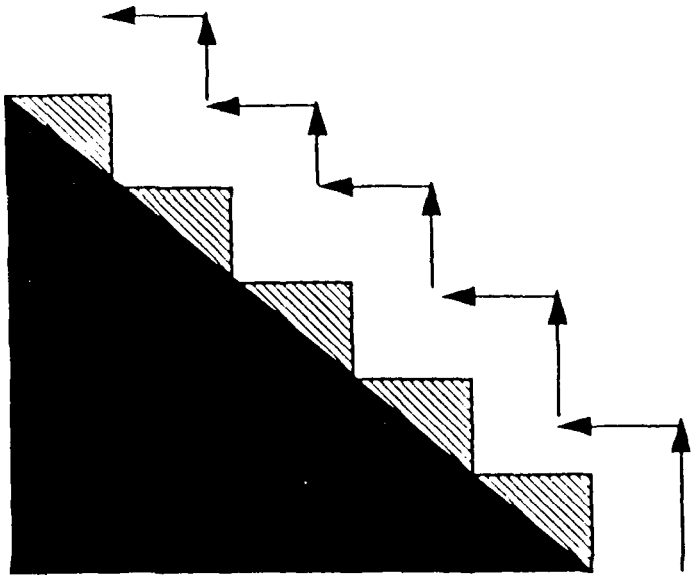




At Times, A System Acquisition Team Plans To Develop And Field The System In A Single Step, Not Allowing New And Unforeseen Requirements That Materialize As The System Matures To Be Easily Incorporated.



SINGLE STEP



STEP BY STEP

Application Of The Model To US Army System Acquisition

- The Problem And Its Significance
- Proposed Approach - An Acquisition Model
  - Six Risk Reduction Strategies
  - Relevance Of The Model To US Army System Acquisition
  - Application Of The Model To US Army System Acquisition
- Plans For Implementation

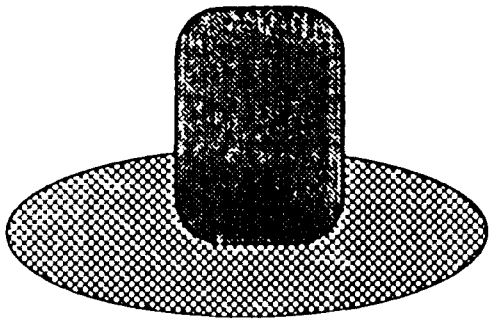
The Model

1. Engineer The Requirements
2. Decouple Requirements Definition
3. Functional Baseline Before RFQ
4. User Interface & Testing In SSS
5. Provide Structure For FSD Relationship
6. Evolutionary System Development

❶ • Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.

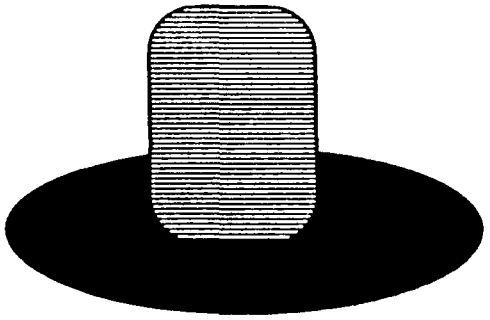
The Requirements Engineer is a consultant. He should be under the control and direction of the Project Manager. He, or his team, must have the needed expertise.

He must wear many hats.

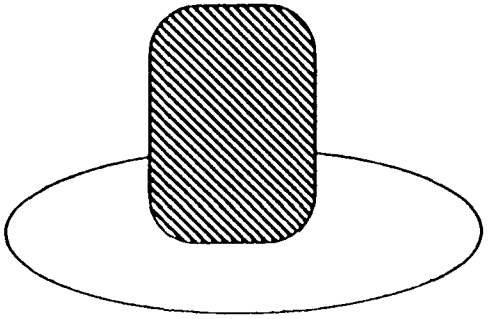


To the user, he is a developer, exploring the feasibility and impact of their requirements and then validating them.

To the developer, he is the user, answering requirements related questions.

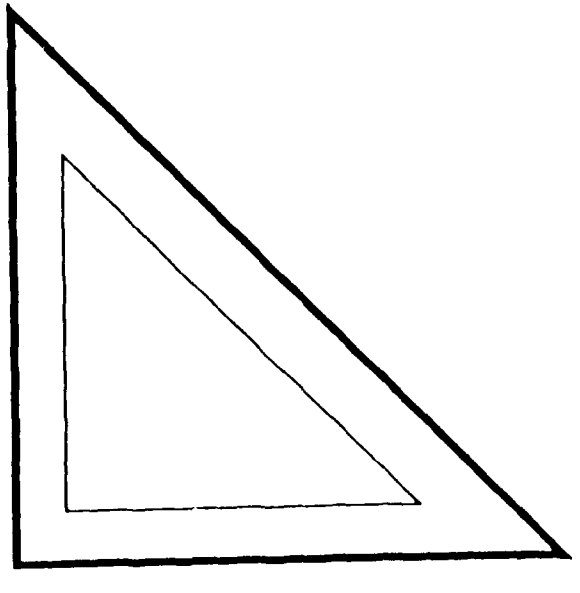


To the Project Manager, he is a consultant on requirements and their impact.



❶ • Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.

The Requirements Engineer's role is much like that of an architect in a building construction project.



❶ • Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.

The Requirements Engineer must have expertise and capabilities to apply the latest Requirements Engineering techniques and technologies.

JOINT APPLICATION DEVELOPMENT

SIMULATION

RAPID PROTOTYPING

❶ • Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.

This model recommends that the Requirements Engineer be involved with requirements related activities throughout the lifetime of the project.

USER FEEDBACK

REQUIREMENTS TRACING

REQUIREMENTS VERIFICATION

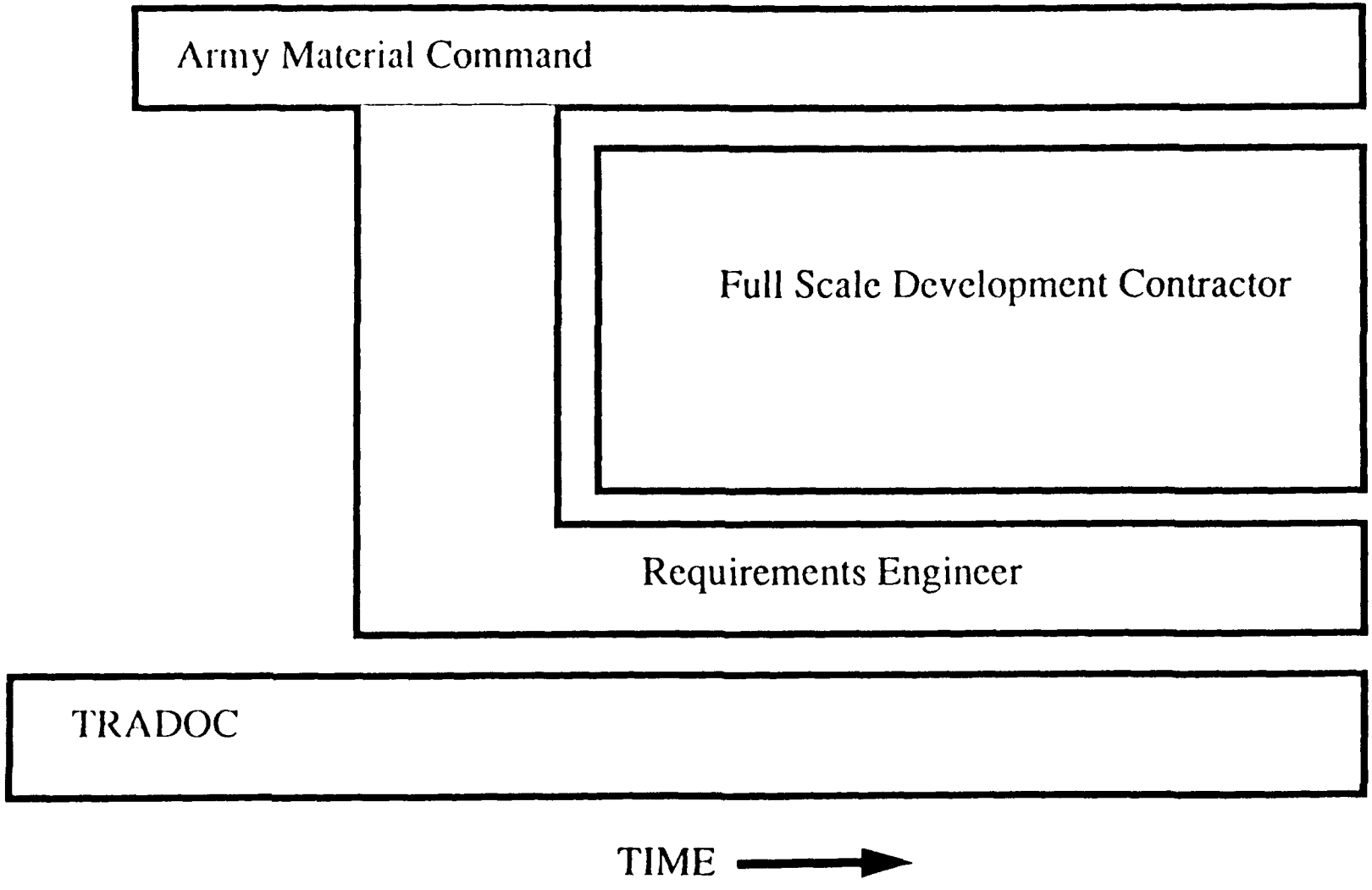
SUPPORT DURING REVIEWS

❶ • Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.

The Requirements Engineering function should be filled by Government personnel. Alternatively it may be performed via contract.

The Project Manager must carefully assess the requirements for this effort and monitor it carefully. Just as with the Full Scale Development effort, the risk of requirements proliferation exists. Unlike the Full Scale Development effort, this effort is on a much smaller scale, reducing impact of risk.

❶ • Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.



❷ • Contractually decouple requirements definition from the Full Scale Development Effort.

The Requirements Engineer, if he is a contractor, should be precluded from the Full Scale Development competition and subcontracting, to maintain his independence.

The Full Scale Development Contractor should only be responsible for activities beginning with software requirements analysis. This would insure that the design effort commences with a well stated set of requirements.

❸ • Establish a Functional Baseline with an approved System/Segment Specification (SSS) prior to the solicitation and make the SSS a part of the solicitation package.

QUALITY

RFP → Full Scale Development Contract Award

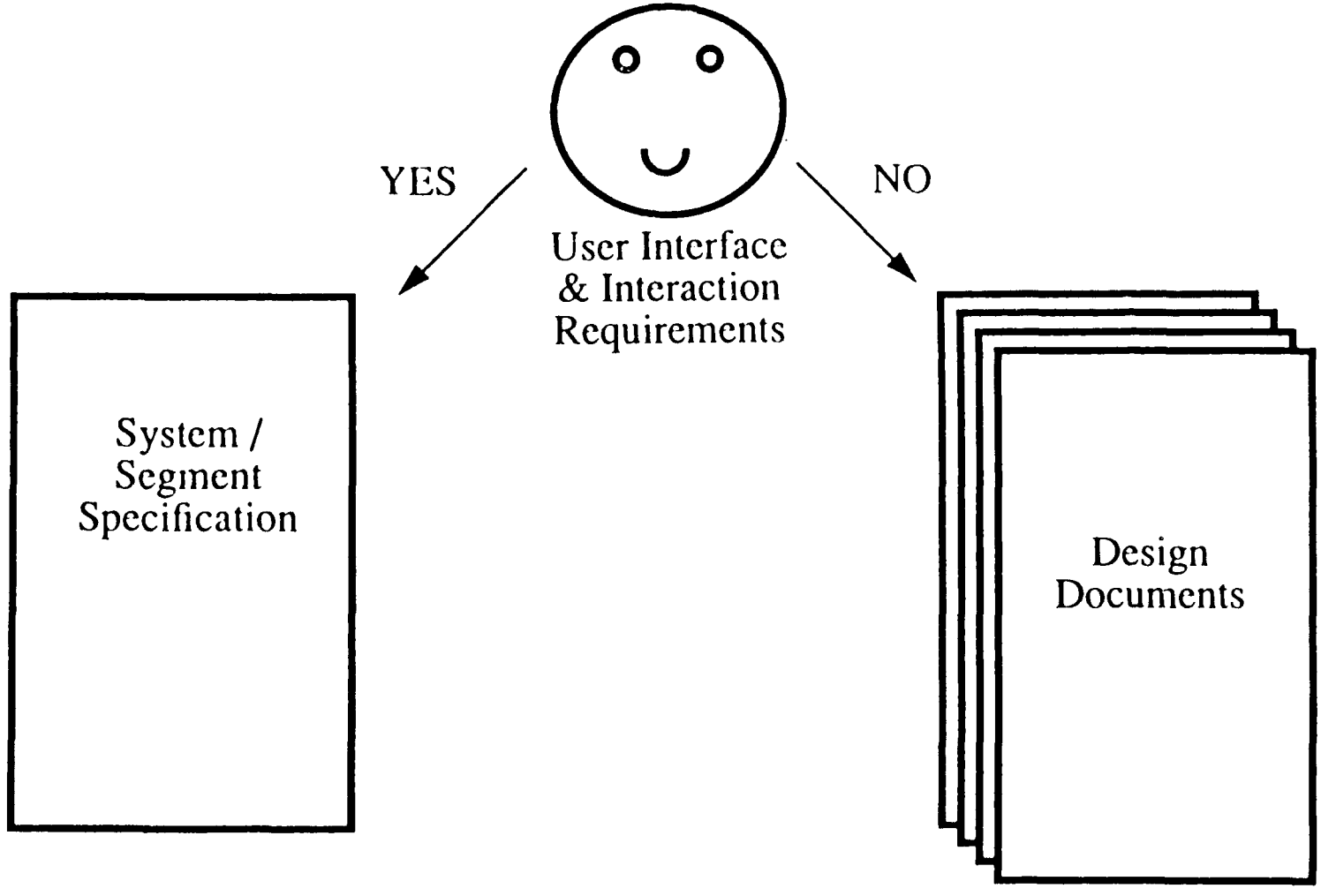
↑

Requirements Engineering Effort → SSS
(+ System Requirement Review)

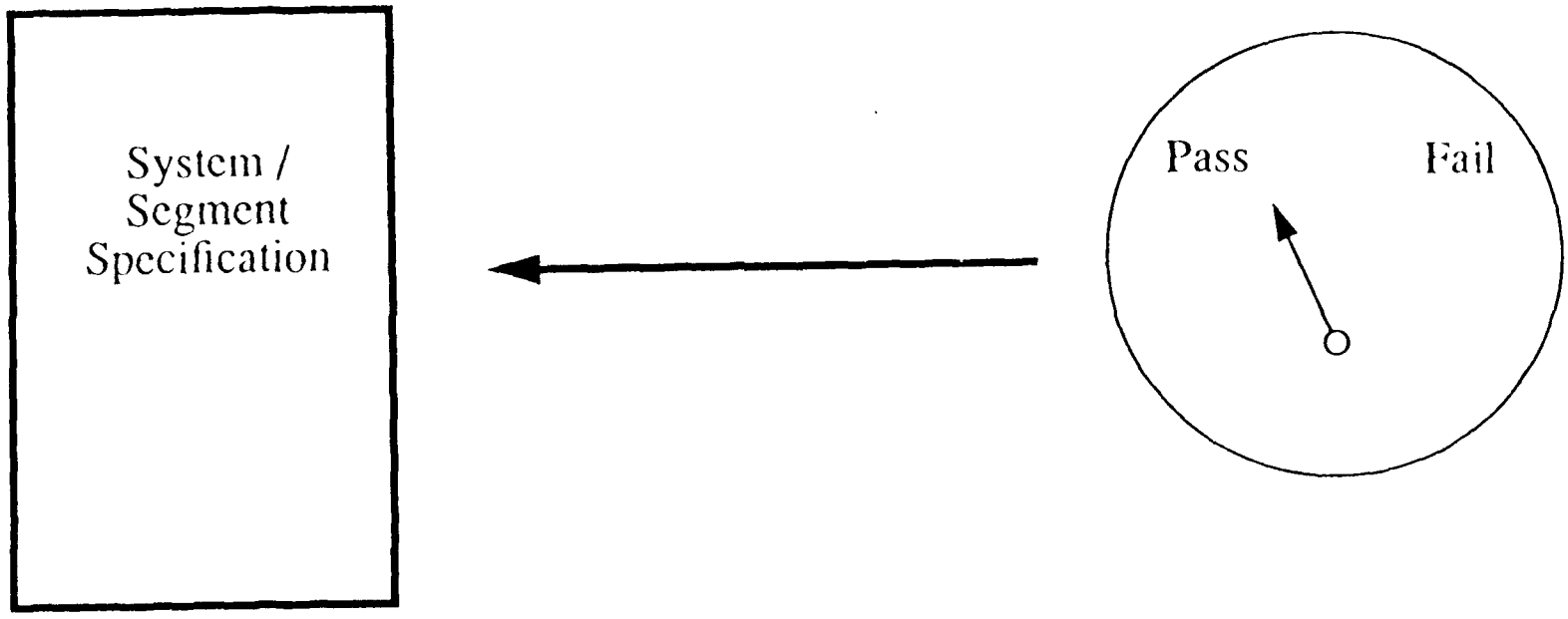
In doing so, we will better know what we are buying and bidders will know what we really want.

This approach does not eliminate the possibility of requirements changes during the solicitation period. It does, however, reduce the opportunities for changes with serious impact to occur.

❹ • Document the user interface and interaction in the SSS, together with system testing information.

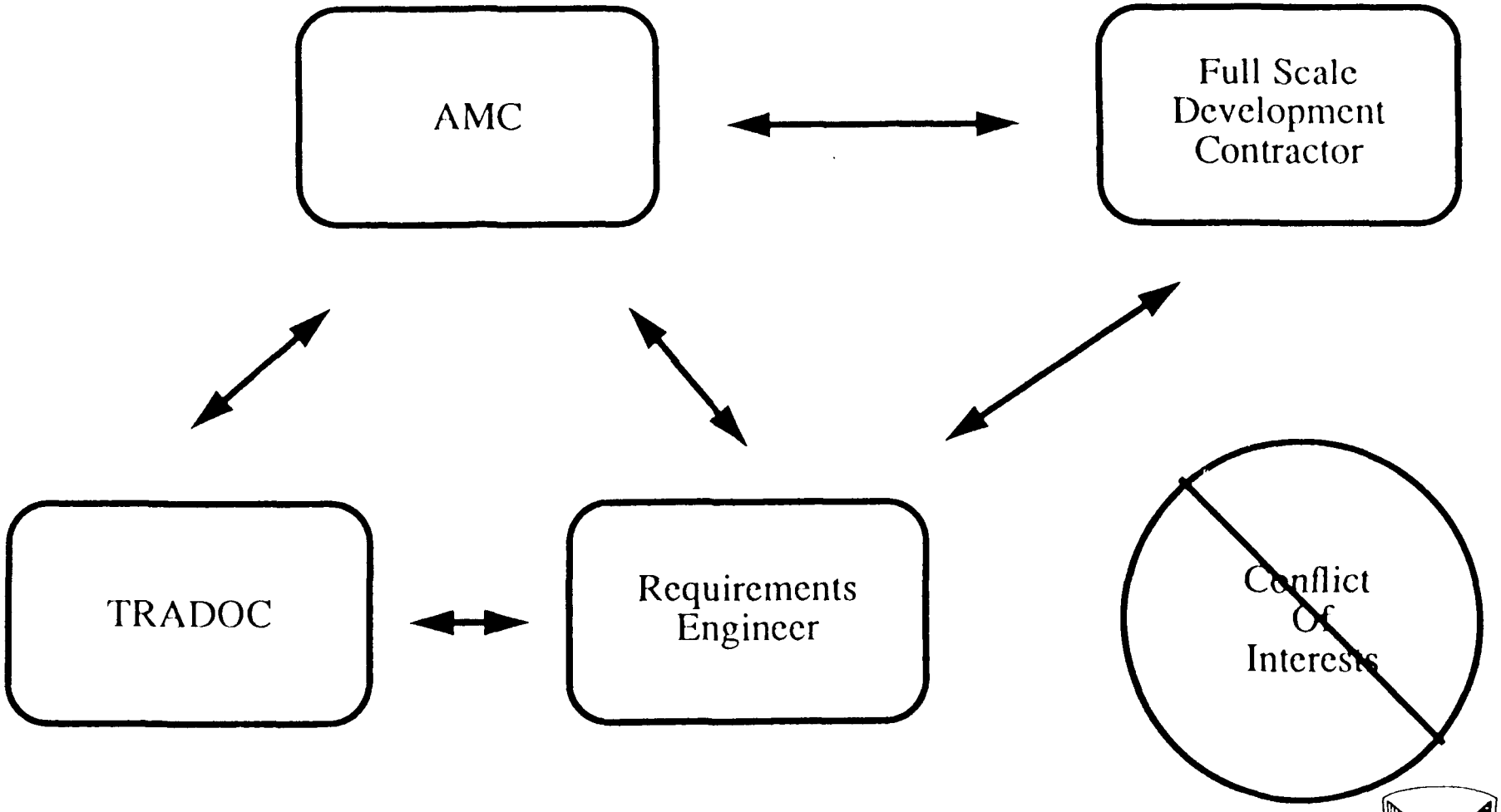


❹ • Document the user interface and interaction in the SSS, together with system testing information.

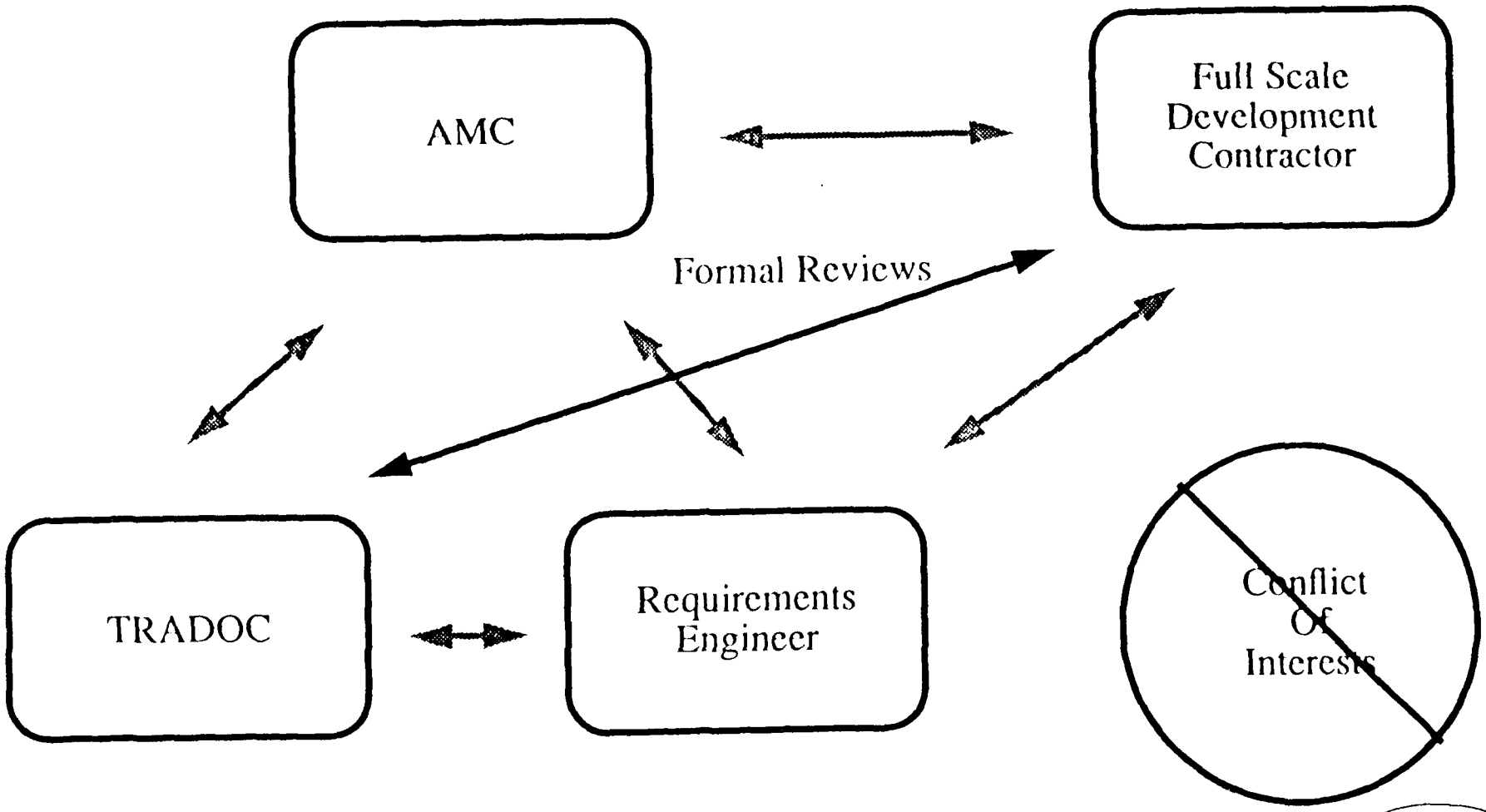


It should be noted that section 4.0 of the SSS deals with provisions for quality assurance. Test case requirement coverage and general system test philosophy should be specified by the RE in this section. Additionally, the RE should specify the system requirements test plan and cases in separate documents and perform the actual testing.

❺ • Provide structure for the relationship and interaction between the user and the full-scale development contractor for all requirements related matters.

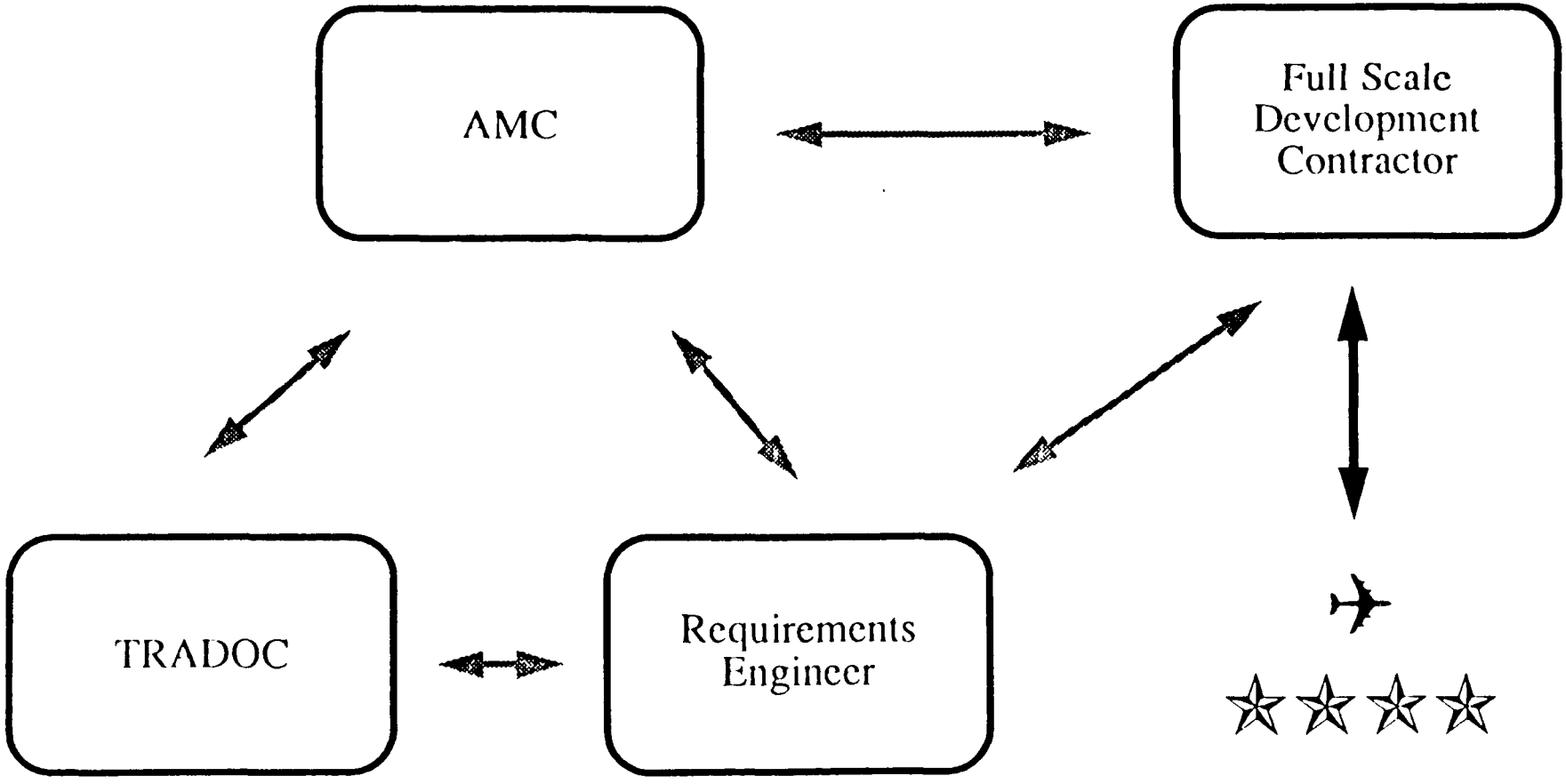


❺ • Provide structure for the relationship and interaction between the user and the full-scale development contractor for all requirements related matters.



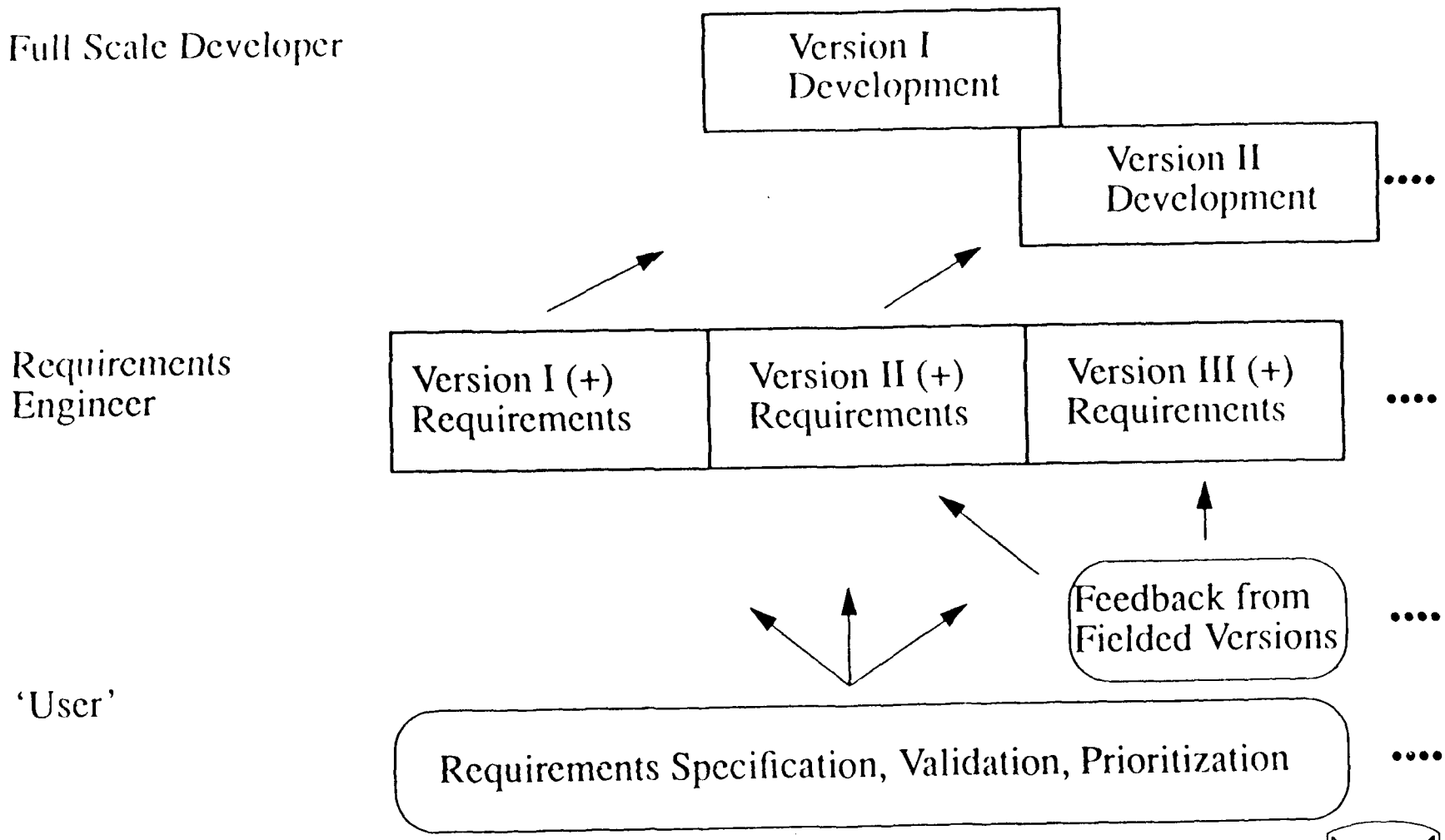
The Combat Developer should be an active participant in the system's formal reviews.

❺ • Provide structure for the relationship and interaction between the user and the full-scale development contractor for all requirements related matters.



This does not preclude the system end user from participating in system testing and training at the contractor's site, under formal arrangements.

⑥ • Plan to develop systems in an incremental, evolutionary manner.



The CECOM CSE Acquisition Model Stresses Requirements Engineering, Emphasizing Techniques For Requirements Definition And Change Management.

It Recommends Six Risk Reduction Strategies, Which Can Be Tailored To Apply To A Wide Range Of Acquisitions.

1. Designate a Requirements Engineering effort which applies Requirements Engineering techniques from the early project phases and on.
2. Contractually decouple requirements definition from the Full Scale Development Effort.
3. Establish a Functional Baseline with an approved System/Segment Specification (SSS) prior to the solicitation and make the SSS a part of the solicitation package.
4. Document the user interface and interaction in the SSS, together with system testing information.
5. Provide structure for the relationship and interaction between the user and the full-scale development contractor for all requirements related matters.
6. Plan to develop systems in an incremental, evolutionary manner.

POSITION PAPER 042591

Requirements Process Analysis [proposed]

Luke Campbell
NATC SY30
bldg 2035
Patuxent River, MD 20670
(301) 862-7601
lcampbell@paxrv-nes.navy.mil

Abstract:

Due to the complexity of today's system developments, coupled with the conditions for greater desire of accuracy, accountability, and smaller development times, proper and accurate capture of requirements for a system's design is becoming a necessary element to enable these conditions. Capture of requirements, which is currently largely unmechanized, can be enhanced with a method not unlike the Software Engineering Institute (SEI) developed Software Process Assessment (SPA).

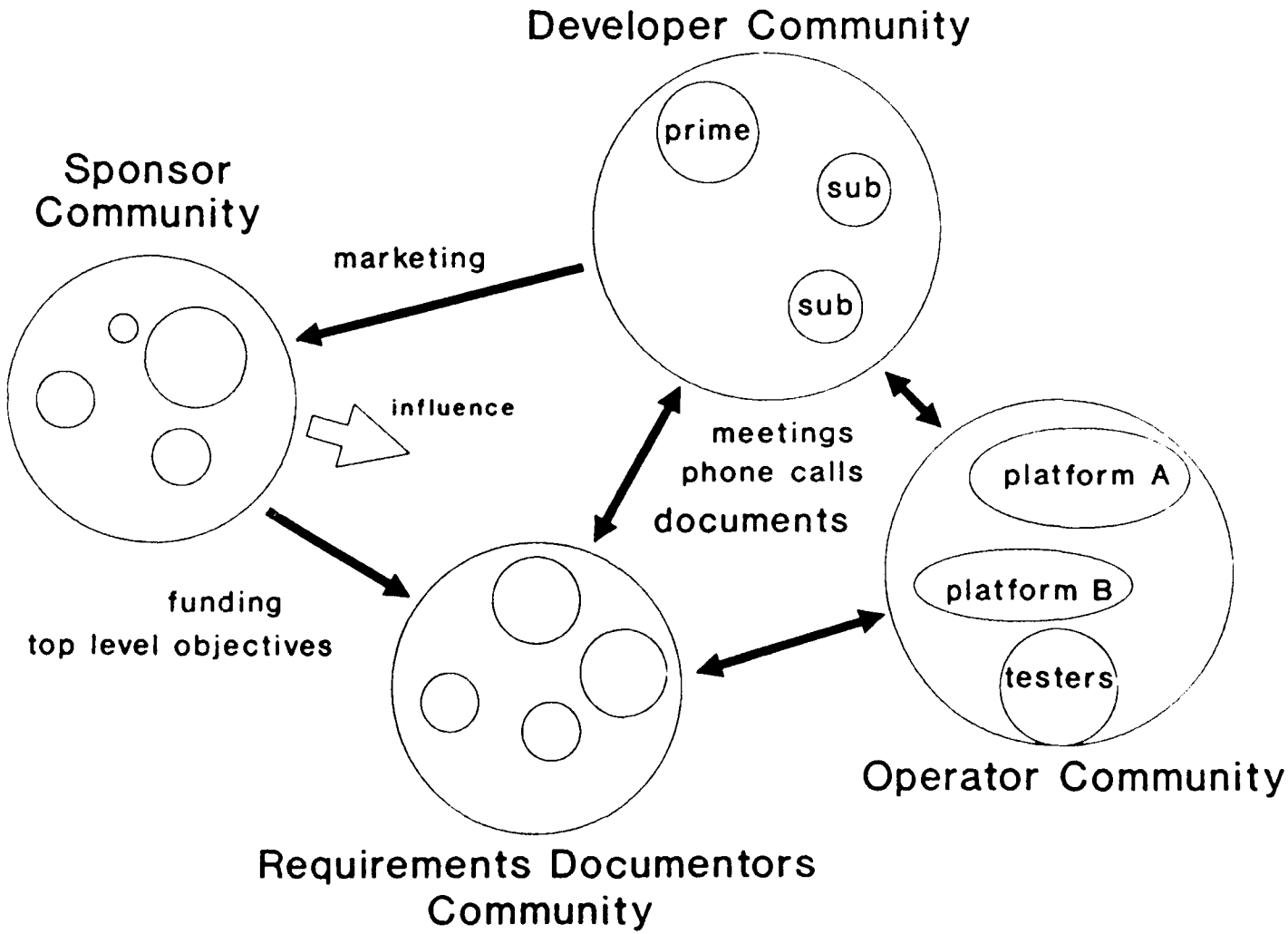
Discussion:

Properly defining and understanding requirements is perhaps one of the single largest areas for lowering program development costs and timeframes in todays world. The earlier in the development cycle the requirements can be properly identified, the lower can be the final cost of the system, and the shorter its development time to market. Additionally, its testing can be more complete and efficient. Moreover, capturing the requirements with a systematic engineering method can lead to a robust development which can better survive the indignities of requirements change or creep later in the life of the product. These changes are due note only to technology and organizational capability, but also to elements completely outside these areas i.e., political.

This document is not intended to be the final thoughts on the subject of elicitation, and certainly, much work remains to be done. However, at the least, it is intended to provoke thought in the concept of a requirements process analysis.

Figure 1 outlines typical communities which are involved in the requirements formulation, capture, and use. This figure should be thought of as tool for visualization of discussions of requirements methods, elicitation, tools, and communications, and not an absolute process for requirements.

Elicitation ... identifies and bridges the disparities among communities involved for the purpose of defining and distilling requirements to meet the constraints of these communities.



There are four communities shown in figure 1. To briefly explain each:

- **SPONSOR COMMUNITY**: In general, one has a sponsor community which supplies funds for work to be performed. While this group is generally not responsible for detailed requirements, they generally set overall goals, and may often influence the design because of their hierarchy and position in industry or government.

- **REQUIREMENTS DOCUMENTORS COMMUNITY**: A second group is one which actually captures requirements. Named here the requirements documentors community. These peoples have the task of translating the high level requirements from the sponsor group (and these requirements may be more thoughts than detailed designs) into working specifications for use by developers and users.

- **DEVELOPER COMMUNITY**: The developer community actually takes the specifications and translates those into products, using the funding provided by the sponsor community. The developers have previously met with the requirements documentors through articles, trade shows, and past program developments, while they use these same marketing techniques and their industrial base to modulate and inform the sponsor community.

- **OPERATOR COMMUNITY**: After the developers are finished and deliver their products, these products come under test and use by an operator community. The operator community should affect the requirements by discussions with the requirements documentors and developer communities (since they are the true customers) by ratings on the products.

In general, these communities operate autonomously and in concert with each other (with various degrees of success), while many products are being developed. As a result, due to feedback from the past, coupled with a considerable amount on-going work, and influence from higher sources, we have an amply complicated scenario for any product development.

The thought is that the process by which requirements occur should be analyzed, understood, and captured, with much the same principle that Watts Humphries of the SEI has developed the Software Process Assessment (SPA). To this end, a Requirements Process Analysis (RPA) is proposed.

One of the major contributions of the SEI SPA work is that it does not force one to adopt a particular process (perish the thought of more than one company or group using the SAME process), rather, it requires that one THINK about the process which is used, and refine this process to continuously improve through metrics. As such, the SPA can be used as a set of thought provoking questions.

It is from the four communities of figure 1 that the requirements process analysis is based and developed.

Requirements Process Description:

In keeping with the methodology similar to that of the SEI SPA, and to

reuse as much accomplished work as possible, five levels of Requirements Process Maturity (RPM) are proposed. Given below are thoughts considering the meanings of the five levels, and a very first cut at possible questions which could be used to guide organizations from one level to the next.

Level 1; Initial. The initial organization environment has ill-defined procedures and controls. While positive response to some of the organizational questions are likely, the organization does not consistently identify or utilize a requirements process, nor does it use modern tools and technology for capture of the requirements.

Level 2: Repeatable. At maturity level 2, the organization has identified the communities which affect requirements for its products, and formally accepts that these identified communities can affect the requirements.

2.1 Has a comprehensive investigation been performed to identify all internal and external communities which affect product requirements?
2.2 Has a formally accepted document been developed that indicates each identified community?
2.3 Has a formally accepted document been developed that itemizes how each identified community can affect the requirements?
2.4 Have timeframes for requirements development among the communities been formally identified and documented?
2.5 Have all people involved in requirements determination been appraised of their position and actions in the process?
2.6 Are metrics in use to characterize the quantity of program requirements?
2.7 Are metrics in use to characterize the changes to requirements?

Level 3; Defined. At maturity level 3, the organization has identified pathways among the communities and utilizes these pathways with formal methods for requirements identification and change. Automated tools are being investigated for use. Also at level 3, a requirements engineering process group is established and staffed, which focuses on requirements process including volatility, elicitation, and tools, and the adequacy with which the process is implemented.

3.1 Have formal communications paths been established among all the identified communities?
3.2 Is there a mechanism for each identified community to propose and modify requirements?
3.3 Are automated tools used in the process of collecting requirements?
3.4 Are the communications paths among the communities sufficiently timely to modify requirements within phases of the development to positively affect the product?
3.5 Has a requirements engineering process group (REPG) been established to focus on the requirements process?
3.6 Have requirements volatility causes been formally identified?
3.7 Do established metrics regarding requirements quantities help formulate future program costs?
3.8 Do established metrics regarding requirements changes help formulate

future program costs?

Level 4; Managed. At maturity level 4, the organization assures that requirements are mandated only by formally established methods. Requirements domains have been established which can be referenced by calling elements. Automated tools are used to capture and modify requirements.

4.1 Do all steps of the requirements process use automated tools?
4.2 Can the automated tools interact to share all the requirements data base?
4.3 Can all parts of the end product be completely and automatically traced to original requirements?
4.4 Can all requirements be traced to appropriate parts of the end product?
4.5 Are deviations from program costs estimated by requirements metrics formally identified?
4.6 Are deviations from program schedules estimated by requirements metrics formally identified?
4.7 Have products from requirements domains been identified?
4.8 Are there attempts to utilize products from requirements domains in various products?

Level 5; Optimized. At maturity level 5, the organization has not only achieved a high degree of control over its process, it also has a major focus on improving and optimizing its operation. This includes more sophisticated analyses of the error and cost data gathered during the process as well as the introduction of comprehensive requirements volatility analysis and prevention studies. Domain models continue to be used heavily, and are updated as a matter of course.

5.1 Is the sponsor community inhibited from influencing requirements except through the use of documented methods?
5.2 Is the requirements documentor community inhibited from influencing requirements except through the use of documented methods?
5.3 Is the developer community inhibited from influencing requirements except through the use of documented methods?
5.4 Is the operator community inhibited from influencing requirements except through the use of documented methods?
5.5 Are products from the requirements domain used in future products?

Unstable Requirements
A Position Paper

Patrick R.H. Place
Software Engineering Institute
(Sponsored by the U.S. Department of Defense)
Carnegie Mellon University
Pittsburgh, PA 15213

March 8, 1991

Abstract

The problems of developing systems when the requirements for those systems are unstable is discussed. Some reasons for instability in requirements are presented and the impact of this instability is discussed. Some suggestions are made for handling unstable requirements. These suggestions include the adoption of practices under a traditional development process as well as some discussion of the advantage of alternative development processes.

1 Introduction

This position paper discusses development issues that arise from instability in system requirements. The suggested approaches for reducing the impact of requirements instability are primarily intended for software development. However, the approaches should be sufficiently general to permit their use for systems as well as software development.

In order to discuss requirements instability, it is useful to assign rôles to key participants in requirements engineering.

Requirer: The individual or group describing a system that is to be built. Examples are government agencies or private companies acting on behalf of some agency. For the purpose of this document users of the system that have some input into the description, for example users who provide feedback to the procuring agency, will also be considered to be requirers.

Requirements: The document (or series of documents) developed by the requirer that describe the system to be built.

Specifier: The individual or group who read the requirements and develop a more detailed description (a specification) of the system. An example is a group within a company that has won the contract who perform high-level analysis and design of the system.

Developer: The individual or group who take a specification of a system and develop the design and implementation. Although such a responsibility is typically taken by more than one group: A design group then, a potentially distinct, implementation group, for the purpose of this paper these groups have similar problems with respect to unstable requirements.

The terms defined will be used throughout the rest of this document. It is recognized that in themselves, the terms may cause some controversy, and we would not object to other terms being used. However, we do consider that they are appropriate labels for the rôles we wish to discuss throughout the paper.

2 Why are Requirements Unstable?

Requirements are unstable for a number of reasons and in this section we will attempt to list those reasons.

2.1 Change in the environment

Systems often have a long life span, where long is a relative term, in some application areas, five years may be a long life span, and in other areas thirty years may be considered normal. Over the cause of the life of a system, the environment in which the system has to operate may change. In such a case, the system must be adapted to the change in environment. Although the overall intent of system operation will be unchanged the requirements may need to be changed to describe the environmental change. For example, a tracking system may need to be extended with the characteristics of a new device that also needs to be tracked.

2.2 Requirer desires new features

Throughout the life of a system, particularly once a system is in operation, requirers will see opportunity for the system to perform new tasks. These new opportunities may arise from changes in the environment as discussed above, or from a deeper understanding of the problem the system is addressing. It is often the case that, as a system performs its assigned function, requirers will see extensions of the system that can make their own tasks simpler. In either case, the requirer will desire that the new features be added to the system, thus necessitating a change in the requirements.

2.3 Requirer insufficiently understands needs

There are times when the requirer will make mistakes in the description of the system due to a lack of understanding of the needs that the system is to satisfy. This may lead to functions being described that appear reasonable, but do not satisfy the needs of the requirer. If a developer does not understand the context in which the system is to perform, the developer may also fail to notice that the needs are not being satisfied. Indeed, the developer is more

likely to construct the system required rather than the system needed since the developer should have some confidence in the competence of the requirer. From the best case (the specifier detects the error) to the worst case (the error is detected when the system is in operation), the error will need to be corrected for the system to perform useful functions. Such a correction will lead to a change in the requirements and, of course, subsequent redevelopment.

2.4 Removal of ambiguities and inconsistencies from requirements

As the systems being built become more and more complex, with increasing function, so will the requirements describing the systems become more and more complex. The requirer will describe the system in terms of natural language descriptions of the functions, of the components, and of other features of the desired system. The requirer may also use mathematics, diagrams, and pictures to describe the system. Unfortunately, natural language text is often subject to interpretation and the specifier and developer will read a meaning in a text other than the meaning intended to be written by the requirer. Also, as the requirements documents become more and more complex, the possibility of inconsistencies in the description of the system increases. For significant systems it is highly unlikely that any one person will have a vision of the entire system, or will write the requirements for the system, again increasing the possibility of inconsistencies in the requirements. Ambiguities and inconsistencies in the requirements will lead to a system that does not satisfy the needs of the requirer. The solution is to alter the requirements for subsequent system redevelopment.

2.5 Experimentation shows a requirement is undesirable

At the early stages of the development process, the specifier will perform experiments in order to gain more understanding of the system to be built. These experiments may be in the form of building prototypes or mathematical models of the system. Each experiment will provide the specifier more information about the operation of the system and it may be the case that a specifier determines that a requirement is either very costly to implement, or is possibly in error, or is unsatisfiable. In each case, the specifier will need to discuss the requirements with the requirer and present appropriate evidence to support the case that the requirement is undesirable. Some requirements will be changed based on this evidence.

3 Why are Unstable Requirements a Problem?

As discussed in the previous section, there are many reasons why the requirements may change. Further, the change in the requirements may occur at any time in the lifetime of the system, from the point at which the contract is first given to any time until the system is decommissioned. In the early life of a system, changes in the requirements may not have much impact. For example, if no development has been performed, then there are no designs or implementations to be altered in order to satisfy the new requirements. Conversely, if the system has been fully implemented and deployed, then a change in the requirements

that is to be installed in all existing systems, for example, a new requirement in response to a change in the environment, will require much effort. New development will have to take place and effort involved in ensuring that the new version of the system is installed appropriately.

3.1 Requirements Change Before Development

As touched upon, a change in requirements before development has begun has less impact than the same change at a later stage. However, there is still an impact on the development of the system. Whether the change is caused by the requirer altering the requirements for the reasons given previously or because of evidence presented by the specifier, the result is the same, that the requirements will be altered.

The new description will have to be examined for any ambiguities or inconsistencies introduced by the change (even when the change is to clear up ambiguities or inconsistencies). Furtner experiments will need to be performed to ensure that the new requirements are desirable, that is, that they satisfy the needs of the requirer and are implementable. Further, new test scenarios may need to be developed in order to determine that a system will satisfy the requirements.

3.2 Requirements Change During Development

If, during the development of the system, the requirements for the system change, then a considerable amount more of effort must be expended in order to build a system to satisfy the needs of the requirer. The work performed in the previous section must be performed since it ensures that the new requirements more accurately describe the system.

Once the new requirements have been accepted an analysis of the changes must be performed in order to determine the components of the system affected by the change. Then, each of these components must be either modified or redeveloped in order to satisfy the new requirements. In the worst case, the entire system may need a redesign and reimplementation.

3.3 Requirements Change After Development

If the requirements are changed after the system has been developed, than all the work of the preceding two sections will need to be performed. The impact of the changes will be the same or greater than that described when the change occurs during development. Since the system has been completed, there is now no chance that affected components may not have been fully developed.

Further, if the decision is made to install the updated system in place of the currently deployed system, then effort will be spent updating existing systems and, potentially, partially retraining users in the operation of the new system.

3.4 Result of a Change in Requirements

As shown in the preceding subsections, any change in requirements causes more work to be performed in the development or operation of the system. Such work is not performed without cost, both in terms of delay to the system satisfying operational need and economically because of the additional effort needed to construct a system satisfying the needs of the requirer.

4 What can be Done?

In this section, we will outline suggestions that should reduce the cost of a change in requirements or make changes unnecessary. These suggestions are made without comment and are intended to stimulate discussion.

4.1 Change the Existing Development Process

Current standard practice for the development of a system is based on a waterfall model of development, with little interaction between the requirers and the developers, other than through the requirements. The process leads to long periods between deliveries of the system. If we accept that the requirer is, for good reason, continually adapting the requirements to a changing environment, then it can be shown that the system will, as time progresses, satisfy fewer and fewer of the requirer's needs. An incremental development and delivery process will get reduce the time between the requirer describing the system and having a system to experiment with and determine how it meets the needs of the requirer. An early delivery will not be expected to satisfy all of the needs, but it can be determined how well the system performs with respect to the needs it is claimed to satisfy. The feedback from the requirer to the developer based on enhancements seen by the requirer from using the incrementally delivered system and from the performance of the system will permit the developers to improve the satisfaction level of the system while adding new features the requirers desire.

Incremental development and delivery will also reduce the effects of contractors delivering an undesirable system (due to misunderstanding the requirements) since the requirer will be able to check early rather than late that the system performs as desired.

4.2 Use of Formal Methods

One of the purposes of the requirements document is to communicate the ideas of the requirer to the specifier and developers. As has been discussed, current practice leads to requirements that are ambiguous and inconsistent. Mathematical methods may be used to describe the system and the descriptions may be analyzed. Ambiguities will be removed by using a notation which does not permit ambiguous expression, unless done so explicitly. Inconsistencies may be detected using mathematical analysis and may be removed subsequently. Improvement of the requirements will lead to fewer misunderstandings by the developers and therefore to a higher probability that the delivered system will satisfy the needs of the requirer.

4.3 Requirements Tracing Through Development

If current practice is improved to provide a capability of tracing the development path from requirements to implementation components then, when the requirements change, the components affected by the change will be readily identifiable. The impact of the change in requirements will be predictable as will the cost in terms of delay to development and additional effort required to reimplement the new requirements.

Conversely, if a component of a delivered system is determined not to satisfy the needs of the requirer, the development path may be traced back to the appropriate requirements that the component is intended to satisfy. The developer and the specifier may then discuss the meaning of the requirements and predict the effort required to reimplement the component or to change the requirement.

4.4 Prototyping Requirements

One of the hardest problems facing the specifier is knowing whether or not the specification really satisfies the needs of the requirer. The ability to prototype systems and demonstrate them to the requirer will lead to the requirer being able to validate the specification without necessarily needing to understand the details of the specification itself. The requirer will be able to generate test scenarios and determine how the system performs with respect to the expectations of the requirer.

4.5 A Process for Changing Requirements

At present, when a requirement is changed, the requirer may have little conception of the impact on the system of the change. Some form of change management procedures should be implemented which permit all parties to determine the impact and desirability of the change. Further, just as we have a system development process, and a process for developing the requirements. So the process for changing the requirements should be investigated so that system development may proceed even when the requirements are unstable. With the developers having confidence that the system they are delivering will satisfy some (or all) of the needs of the requirer.

POSITION PAPER

Requirements Engineering Processes and Products[1]

Kyo C. Kang

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
NET: kck@sei.cmu.edu

Application of the waterfall process in the development of complex, large-scale, multi-function systems has created problems as it forces acquisition managers and developers to write highly-detailed requirements for systems whose boundaries are extremely fluid, therefore removing some flexibility in adjusting to evolving circumstances [CONG89]. This "process misfit" has resulted:

- long requirements development time
- validation difficulty
- considerable requirements volatility and requirements traceability problems
- high development and maintenance costs, and brittle software resulting from continuous modifications
- unmanageable documentation

An evolutionary acquisition strategy [JNTL87] has been proposed to address these problems, by supporting incremental evolutionary systems development. With this model each acquisition of a new system involves:

- outlining of the final desired capability of the system and identification of the core capability
- development of detailed specifications for the core/baseline capability
- development of an architectural framework which supports incremental and adaptive development
- development and deployment of the core capability

Each increment to the core capability is treated as a separate acquisition with this approach.

This process model, however, appears to present new technical difficulties. For example, it may be difficult to define an architectural framework that will allow incremental additions, minimizing modifications to the pre-developed components. Also, it is unclear if the current software techniques can support this approach adequately. There was an earlier attempt [CECO89] by the Army CECOM to address some of these issues. Therefore, I propose that this working group make an in-depth investigation of this approach and identify (potential) problem areas, building upon the results (Appendix) from the CECOM's workshop. Some of specific questions this process working group might address are:

[1]Sponsored by the U.S. Department of Defense

- Has anyone applied this model? What were your experiences?
- Would this model be applicable in your project environment? What problems do you see?
- Could this model be applied to most (all) of the current and future systems development? Will it solve most of the process problems? Are there any applications or project situations that may require a different process model?
- How do we verify overall system properties (e.g., security, performance, safety, etc.) without a complete system?
- Will the current software engineering methods and tools be applicable? How should DoD2167A be adapted? How about DIDs?
- What new software engineering methods, tools and techniques do we need? Do you see any new problems (e.g., configuration management issues, validation difficulty, documentation standards) that might be resulted from this approach?
- How should this model be adapted to a specific project? What are the parameters (e.g., risk assessment, domain maturity, resource and time constraints) that have to be considered to develop a project-specific development strategy? How do we use these parameters to develop a project-specific process model?
- How does reuse fit?
- What other process models (e.g., spiral model, transformational model) have you applied? What were your experiences? Any problems?
- Are there any unique problems you have that any of the process models discussed cannot address?

References

[CECO89] Proceedings of the Requirements Engineering and Rapid Prototyping Workshop, Center for Software Engineering, U.S. Army Communications-Electronics Command, November 14-16, 1989

[CONG89] "BUGS IN THE PROGRAM: Problems in Federal Government Computer Software Development and Regulation," Staff Study by the Subcommittee on Investigation and Oversight transmitted to the Committee on Science, Space, and Technology, U.S. House of Representatives, September 1989

[JNTL87] "EVOLUTIONARY ACQUISITION: An Alternative Strategy for Acquiring Command and Control (C2) Systems," The Defense Systems Management College, Fort Belvoir, VA, March 1987

Appendix

A Summary of the Results from the CECOM Workshop

The key process-related problems identified by the Requirements Engineering Process Working Group (at the CECOM's workshop) and their recommendations for those problems are summarized below.

Problems:

1. Uncertainty and change are difficult to cope with.
2. Validation of requirements is critical to project success.
3. Multiple stakeholders make it difficult to reach closure.
4. We do not know how to track progress in requirements development.
5. Different processes are needed for different problems.
6. System/Software Requirements Analysis/Design phase differentiations are unclear.
7. The existing inventory of systems needs to be retrofitted to new requirements engineering technology.

Recommendations for the problems 1 through 4:

1. Freeze requirements in small incremental builds.
2. Develop more testbeds like AIN (?) to validate interoperability earlier in the development process.
3. Develop/Transition new techniques to isolate acceptable requirements partitions.
4. Develop/Transition new techniques to accommodate change in requirements and designs.
5. Develop and refine practical formal requirements techniques.
6. Define a multi-stakeholder requirements process.
7. Develop thorough understanding of requirements "normalization." Somewhat analogous to database normalization, this envisioned technique would enable two sets of requirements to be shown to be equivalent.
8. Define and understand requirements process models.
9. Define and understand models of requirements progress.
10. Perform experiments to determine what conditions make evolutionary acquisition and prototyping practical.
11. Develop tools/techniques to capture merits/tradeoffs among requirements.

The Integrated Requirements Process
A Position Paper

William S. Gilmore
Software Engineering Institute
(Sponsored by the U.S. Department of Defense)
Carnegie Mellon University
Pittsburgh, PA 15213

March 4, 1991

1 Introduction

In the search for the more "ideal" requirements process, it can be easy to lose sight of the real problem: to have a more ideal system life cycle process. We might define "successful" system life cycle process to mean maximizing the benefit / cost ratio over the lifetime of the system, with reasonable predictability and control. Having a successful requirements sub-process is essential to system life cycle success. However, because the requirements process is an essential ingredient does not also mean that solving the requirements process problem is a problem that can be worked on in isolation of the other ingredients.

A motivating question for this paper is "Can the best requirements process be conceived by focussing on the requirements process alone, or is the best requirements process conceived as part of a Whole System Life Cycle Process (WSLCP) model. More simply, we ask two questions in this paper:

Q-1: Can the requirements process problem be isolated?

Q-2: What does an integrated requirements process look like?

To the first question I claim "no". Justification follows. As a result, the second question is important. Some characteristics of an integrated requirements process are described to try to motivate thinking and discussion as to what the best integrated requirements process really is. "Requirements level decision curves" are proposed as an analytic tool to help characterize the requirements evolution process, and in so doing can help in decision making during that evolution.

2 Isolating the Requirements Process Problem

Why is the requirements process not an isolated problem? Unless one can conceive <u>all</u> the requirements up front and no change or flexibility will ever be needed, then there will be a <u>two way</u> dependence between requirements evolution and "other" parts of the WSLCP, including architecture, project management, and the system acquisition process. It is rare and unlikely that all requirements, including all the lowest level details, can be completely specified before any design, building, and planning occur. Architecture, design, and requirements will need to interface. For example, requirements specify what to build, but efforts to design and resolve architecture raise issues of cost, benefit, and risk for what can be built. Furthermore, since architecture decisions cause constraints in what future designs and design changes are possible, in essence they also constrain future requirements evolution. Hence, architecture decisions depend on the requirements flexibility one is willing to live with.

Furthermore, requirements and project management will need to interact. For example, the venture of building and developing depends on what is wanted to be built; hence, one derives what resources must be allocated based upon the requirements. On the other hand,

the cost to allocate such resources may become a factor in deciding upon some requirements when weighed against the relative benefit of alternative requirements.

In addition, the acquisition and requirements processes will need to interface. Contracts are the basis of work plans; such work plans should be matched to the state of requirements, helping them to evolve or seeking their translation into real systems. For example, if requirements are not worked out well enough when a full-scale development contract is established, there may easily be cost and schedule problems associated with a long build time. During such long builds, technology and requirements may become outdated, and needed changes will become more difficult and expensive to accommodate. Appropriate contract content and administration can support requirements evolution through demand for evolution-supporting deliverables and through flexibility and good communication.

Therefore, each sub-process will have a dependency on information that arises as part of the others; each will have need for information exchange and traceability with the others to support decision-making, trade-off analyses, and planning.

3 Characterizing the Integrated Requirements Process

Observe that the argument against developing a requirements process model in isolation from a WSLCP model depends upon being unable to be completely specify requirements in advance of any designing, building, or planning. Although this is generally true, it is paradoxical in that one wants to know as much about requirements as possible in advance in order to proceed as efficiently and effectively as possible. Fortunately, not all requirements must be known in order to proceed.

In general, there is a shift over time in the main target level at which requirements decisions occur, from understanding and defining requirements at a high level to emphasizing details at lower levels. This shift corresponds with identifiable stages of progress in a WSLCP evolution. The stages and their progression are: defining the system, defining the architecture, then fielding the system. This progression is due to a fundamental information dependency between the stages. Architecture that adequately supports evolving requirements and design is needed before serious system building occurs. However, defining system context and objectives and estimating the area and extent of future changes motivate the architecture. Figure 1 qualitatively depicts this shift. For discussion, these curves are named "requirements level decision curves".

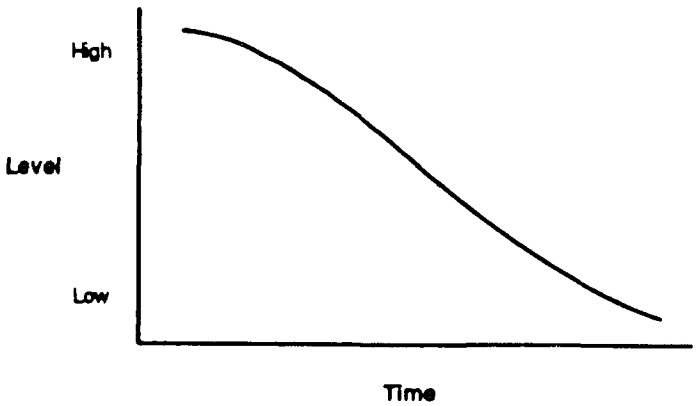


Figure 1. Requirements evolution can be characterized by requirements level decision curves that show the shift over time in level of emphasis of requirements definition.

Requirements level decision curves show the shift over time in level of emphasis of requirements definition. Such curves may be used to help analyze the requirements process. Examples follow.

First, note that actual requirements level decision curves have thickness and oscillation, as depicted in figure 2. What causes these effects? Thickness is due to the need to work on high and low level details together. High level direction is needed to scope and direct relevant low level investigation, yet some understanding of low level capabilities and feasibilities is needed to define the desired high level direction. Observe that most of the thickness falls below the main target level for decision making (the solid smooth curve). Oscillation indicates backtracking mainly due to the need to correct errors in earlier decisions. After some trial time with a decision, one may find that something different is wanted. Whenever development and implementation lag behind specification of requirements, change of what is wanted is inevitable. Such changes arise during the development, due to both seeing better ways to engineer the solution, better ways to apply technology, and changing one's mind as to what is wanted after seeing what the real emerging product is like.

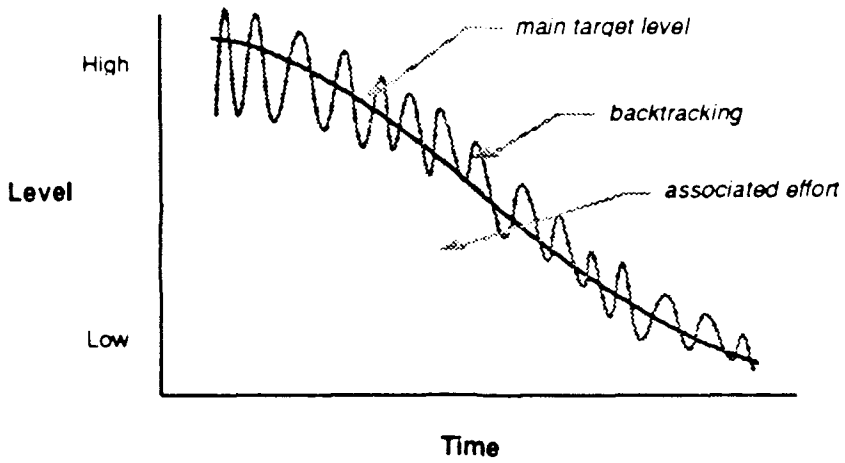


Figure 2. In real life, requirements level decision curves show thickness and oscillation.

Clearly, the objective in the requirements process is to lower the main target level of decision making as expediently as possible and minimize the oscillation due to backtracking. Expediently in this case means as quickly as possible, but not quicker than is reasonable based on corresponding progress in architecture and lower level design. Figure 3 shows this emphasis.

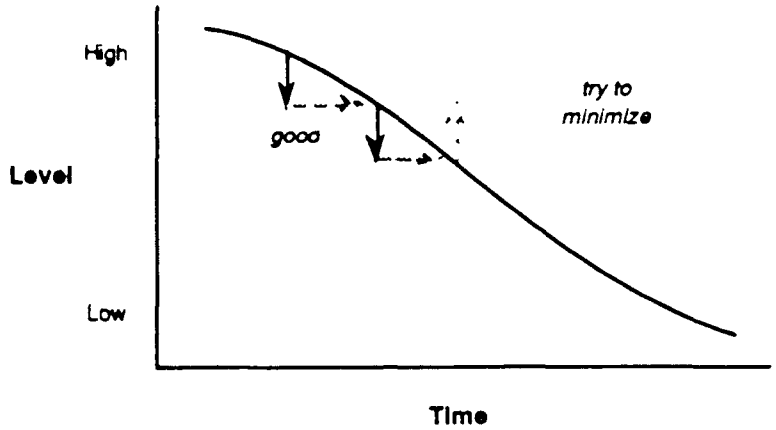


Figure 3. The objective in the requirements process is to lower the main target level of decision making as expediently as possible and minimize backtracking.

Now what would be the meaning of a system evolution that displayed a huge oscillation, as shown in figure 4? Such oscillation connotes multiple rebuilds. This would be extremely expensive, indicative, for example, of much rebuilding occurring during PDSS (Post-Deployment System Support). The surplus expense arises because of substantial backtracking and excessive low level work done to support the wrong high level objectives. If requirements definition is not well integrated and paced with the other parts of process, e.g., architecture, project management, the acquisition process, then substantial backtracking may result. For example, if project management fails to support adequate staff development in application and technology domain knowledge or in learning about specific requirements wanted, preventable backtracking will result. Or if communication between engineers and the customer is poor, engineers' discoveries of "wrong direction" may go unused throughout the contract, but the product shortcomings may later be found during use of the product. Or if a contract calls for full-scale development of a working system when the requirements, user needs, and architecture are not yet worked out to a sufficiently low level, delays and even rebuilding may result.

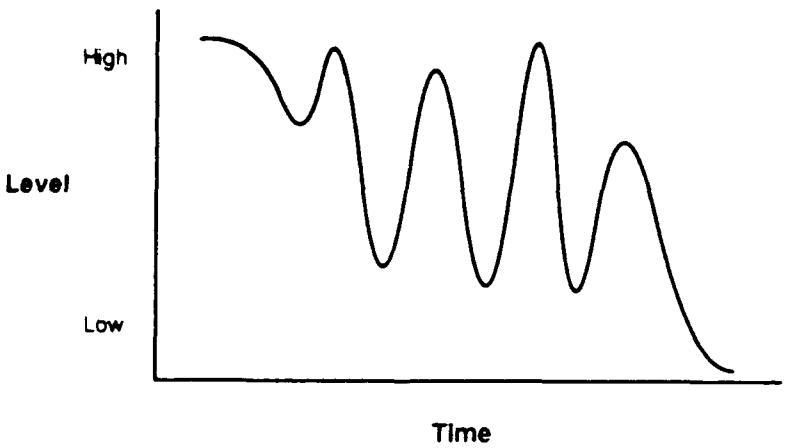


Figure 4. Large oscillation in requirements level decision curves corresponds to expensive backtracking and rebuilding.

This position paper asserts that the best requirements process is one that is integrated with other parts of the WSLCP, and that the best way to model such a process is to model the WSLCP and identify the requirements process within it. Requirements level decision curves are suggested to help analyze these processes. One can analyze life cycle cost with respect to requirements level decision curves. Furthermore, one can assess what are good requirements tools, when they should be used, and how they coincidently support the WSLCP by assessing one's position with respect to a requirements level decision curve and asking what tool and analysis support can best help to make main target level requirements decisions expediently and minimize future backtracking.

Tools and good process both can help a system evolution trace an optimum path for a requirements level decision curve. One can never have zero oscillation in a requirements level decision curve – this would correspond to "no mistakes" in requirements definition; but one can try to proceed efficiently while minimizing backtracking. To do this, one not only tries to catch mistakes as soon as possible, but also one tries to anticipate what is most desirable, through modeling and analyses. Tools and good process each can help. Tools can support "what-if" rather than "how did we fare" analyses. Requirements tools such as rate monotonic scheduling analysis, user interface prototyping, cost / benefit modeling, analysis for requirements analysis consistency and completeness all can be seen as helpful in this regard. Process can support efficient and effective information flow, and exchange, so that decisions are repeatedly made on the basis of the "best" information available. CASE tools can support such process because they not only help engineers work through problems, they also help teams communicate their intentions among each other.

In practice today, many managers opt not to purchase such tools. Considering requirements level decision curves may add the perspective to help in doing the cost / benefit analysis behind making the decision whether or not to purchase and use such tools.

4 Perspective on Tools and Process Support

In order to further analyze the integrated requirements process, one must consider what activities are needed at various stages of system evolution – an evolution that follows some descending requirements level decision curve. As figure 5 suggests, activities such as prototyping, simulation, and cost modeling are used time and again, but not at all in the same way. The purpose and nature of such activities is defined according to the stage of evolution and major thrust of effort. The indicated changes in purpose and content suggest how different tools may be needed.

Cross sections in time in figure 5, shown as vertical "stage" rectangles, correspond to "level" points on requirements level decision curves. The "thickness" in requirements level decision curves is elaborated in figure 5, where explicit emphasis on "lower" level activities is shown. However, figure 5 also shows that activities have substantially different emphasis and purpose depending upon the stage of evolution. For example, early in a system's lifetime when one is epxloring what system is needed, prototyping and simulation activities support high level requirements decisions by modeling system context and objectives, and exploring technical details to the extent that they affect feasibility and cost. At a later stage in the system's evolution when the emphasis may be on making major architecture decisions, and so more specific requirements must be settled, prototyping and simulation now focus on studying behavior and high level structure.

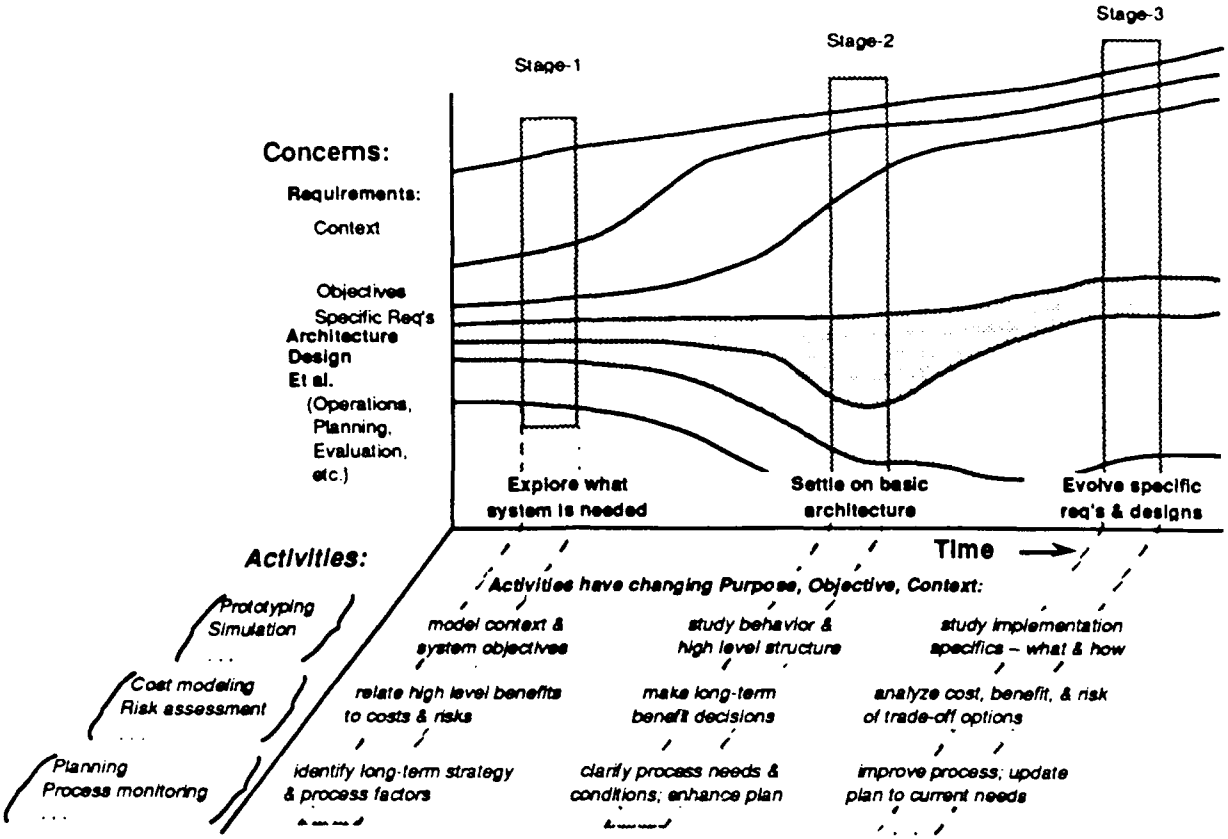


Figure 5. The changing thrust of various activities, depending upon stage of emphasis and progress in a system evolution.

Though the stages of progress shown in figure 5 are sequential, the process exemplified by activities markedly differs from a Waterfall Model sequence. The same activities – prototyping, simulation, modeling, planning, etc., – support each stage, but the role and nature of these activities vary as the system matures. Any WSLCP model designed to support a system's evolution will have include continuity and parallelism of activities.

An underlying assumption of this paper has been that if one can understand the progression and the nature of the parallelism and interaction of the activities, one may have better perspective to make decisions to support it. Requirements level decision curves may help in this analysis.

Seven (Plus or Minus Two) Challenges for Requirements Analysis Research

Colin Potts

MCC Software Technology Program

Requirements are the wellspring of software development. As many authors have noted, errors made during the requirements phase of a project tend not to be detected until the system is delivered, by which time it is very expensive to correct them. In spite of the criticality of the requirements phase, it has been the target of much less research attention than the later stages of software design and implementation. What work has been done in universitites and industrial research laboratories has had little impact on practice. As a result, requirements analysis is performed in much the same way in many organizations as it was twenty years ago. And those organizations that are innovating by using structured and object-oriented methods are doing so largely as a result of the writings of a small community of articulate and experienced consultants.

I do not believe that this is an anomaly, that the software industry is so short-sighted that it cannot see the value of research innovations and apply them in practice. The simple truth is that the research community has offered little to practical people that they can use.

We will make breakthroughs in requirements analysis practice only by addressing a hard core of research challenges. None of these are very exotic or unexpected. Indeed, their comparative mundaneness may have contributed to our attention being directed elsewhere. Nevertheless, they have to be grappled with; all seven (plus or minus two) of them.

Requirements analysis is more than anything else a process of human communication, and the power of computational and representational techniques should not blind us to this fact. Most of the challenges we face involve bridging some communication gap or other. There are two techniques that can be used to bridge any gap, conceptual or physical: choose the narrowest gap possible, and build a good bridge. Research into requirements analysis will benefit practitioners only to the extent that it succeeds in these simply stated tasks.

Challenge 1: Bridging the gap from concept to essential model

Existing structured methods have popularized the notion of the "essential model," a model of the envisaged system stripped of assumptions about imperfections in the environment or implementation. Although the term was introduced by McMenamin and Palmer (1984) in their book on Structured Analysis, an essential model does not have to be constructed from a functional viewpoint. One could equally well talk of an object-oriented essential model, or an entity-relationship essential model. Whatever paradigm is adopted, the first challenge is how to build such a model in the first place.

Building an essential model for a large system is a multi-stage process. In practice, system requirements are usually put on paper in the form of a textual requirements document, and this is regarded as the origin of the model. In reality, the origin resides not on paper but inside a number of people's heads, and the problem an analyst faces, whether the intermediate medium of a requirements document is used or not, is to get at those concepts and check that what is being explicitly modeled is consistent with them.

Challenge 1a: Bridging the gap from concept to document

Most methods assume some form of requirements document. But where does that come from? How are requirements formulated in the first place? There are really two separate issues, here: elicitation strategy and document style and organization.

An elicitation strategy is a set of guidelines for identifying the correct sources of requirements (as well as background information), eliciting requirements from them, and resolving conflicts among them. Of all the aspects of requirements analysis, this is the most communication-rich. As a result, most of the techniques that prove useful do not stem from computer science research, but from organizational theory, group interaction research, ethnological interviewing techniques, and most of all from seat-of-the-pants experience. A good example of a prescriptive but flexible set of guidelines that is surprisingly little used in practice is the viewpoint identification phase of the CORE method (Mullery, 1985). This can be used to identify key system viewpoints and representatives for each viewpoint. CORE has some recommendations for conflict resolution when requirements from different representatives conflict, but ultimately little general advice can be given.

Another specialized elicitation technique that has been used for problem formulation in various design disciplines is the issue-based approach of Horst Rittel. This has been adopted for at least one medium-size development project (Yakemovic and Conklin, 1990) for keeping track of requirements interpretations and early design decisions.

Document style and organization, however, is a potentially more fertile area of research. Complex organization and cross-referencing schemes that had been infeasible are now achievable with current hypertext technology. Hypertext has been used for engineering documentation (DeLisle and Schwartz, 1986), and requirements information is an ideal application. Now that the enabling technology is available, the research issue is how to organize hypertextual information for specific software engineering applications. What is needed is a set of organizing principles that are as appropriate to the enabling technology and the systems of the 1990's as the SCR document organization guidelines produced as part of the A7 aircraft software project (Heninger, 1980) were for the late 1970's.

Less prosaically, there are probably innovations waiting to be made in artificial dialects of natural language for specialized sub-domains. For example, a "temporal English" that had a precise semantics for temporal relations (including duration) would be very beneficial in describing subtle real-time interactions in embedded systems. The prospect of natural language translation from a free-form requirements document to a formal specification is infeasible (although the inverse translation is not), but for highly constrained idioms, such as those describing timing properties, it could probably be done. The real problem in practice would be ensuring that the interpretable idioms were actually used and not some other set.

Challenge 1b: Bridging the gap from document to essential model

Most methods provide some recommendations about the transition from a requirements document to an essential model. What guidelines there are, however, are not very strict. The analyst is supposed to give the document a close reading so that events, actions, objects, states and so on can be identified. However, this is not a very systematic process.

With the promulgation of new object-oriented approaches this is starting to change (Wirf-Brock, et al., 1990). In common with JSD (Jackson, 1983), detailed guidelines for object identification, pruning and elaboration can be used for going fairly directly from a textual description of the requirements to an initial model of the system.

Like the production of a document this is also a two-stage process. The first stage involves sifting and evaluating the raw text. It is a time-consuming and largely uncreative task. Aguilera and Berry (1990) discuss a prototype system for extracting and cross-referencing repeated phrases in a requirements document. Without fine-tuning for the concepts central to particular methods, this is unlikely to be of universal applicability (for example, Wirf-Brock et al's. object identification heuristics require a greater emphasis on noun phrases). Nevertheless, it is a very promising application of text processing techniques.

Even more challenging is support for the subsequent stage: interpretation and reformulation. What, for example, is the analyst to make of a list of repeated phrases? The principal breakthroughs in this area are more likely to be methodological than technological. It is difficult to say what they will be or how wide-ranging. It is no coincidence, however, that the strongest heuristics come from methods which emphasise domain structure rather than behavior. Among the former classification are JSD, object-oriented methods and entity-relationship modeling; whereas among the latter are the variants of Structured Analysis and state transition models. As object-oriented programming and design becomes more common, the need for a comparatively seamless representational progression will drive the development and refinement of object identification heuristics.

Challenge 2: Bridging the gap from essential model to system architecture

An essential model is supposed to be a model of the essential characteristics of the system without regard to implementation constraints. For this reason, requirements are usually described as being about 'what' the system will do and not 'how'. This is a naive distinction. Requirements frequently include performance or hard real-time constraints, and their feasibility cannot be assessed independently of the system architecture. Moreover, system cost is a direct function of architectural complexity and component cost, an issue that cannot be ignored until a later phase of the lifecycle. The process of requirements negotiation can only be informed by some model of the system architecture. So: how can the gap between essential model and architecture be bridged? and how can an analyst avoid corrupting the essential model by implementation bias, while at the same time guiding the project planning process?

For some methods, there are already partial answers to these questions. For example, ADARTS (Gomaa, 1989) is a method for transforming a real-time Structured Analysis essential model into a task structure for an Ada system. Some of its heuristics are sufficiently general that they could be applied to other target implementations. Having identified the likely subsystems, it is much easier to do preliminary system planning and cost estimation.

Challenge 3: Bridging these gaps backwards

System development is not a one-way process. During requirements analysis, communication from the model back to the concept is as important as deriving the model. The main mechanism for doing this is currently document inspecuons. This is one area, however, in which technological solutions are already available with more on the horizon.

Many representational techniques either have an operational semantics or could have their semantics specified without major revisions to the spirit of the technique. Ward (1986) has given an informal execution semantics for the Ward/Mellor variant of real-time Structured Analysis and JSD is defined in terms of concurrent sequential processes, similar in many respects to Zave's(1982) PAISLey. Kramer et al (1988) have developed animators for Structured Analysis variants, and Potts et al. (1985) developed translation rules from JSD models to simulation programs in Ada. Finite-state machines, statecharts and Petri nets are also operational models. Among the many tools that animate these types of models, one can single out STATEMATE (Harel et al., 1988) which animates statecharts. VERDI (Shen et al., 1990) is a specialized tool for visually executing distributed systems designs.

Direct execution of an essential model has much in common with prototyping. It has the advantage that the executable model is directly derived from the essential model and therefore can be used to validate the requirements. A rapid prototype, on the other hand, cannot be guaranteed to portray the system accurately. Prototyping has the advantage that it can be used even when an essential model is not available; for example, while the requirements are being identified. A hybid approach, in which a prototype is developed manually but using a visual/textual language based directly on Structured Analysis concepts, is also possible (Luqi and Berzins, 1988).

To be as useable as a prototype, an executable essential model must be enriched by a domain-specific visual interface. The interface might be a mockup of the planned system's user interface, but it might equally provide a "bird's eye view" of the system's behavior in a domain-specific way. The challenge here is to provide a sufficiently generic visualization engine that prototype construction does not require too much time-consuming interface programming. This is an active area of research at MCC and elsewhere.

A system architecture also needs validating in much the same way as the essential model. Although the user may not be directly concerned with the architectural decisions made by the development project, a user may be very interested in the performance implications of these decisions. It is unrealistic to detach these concerns altogether from requirements validation proper, because it often turns out that performance or timing requirements are not as hard as the customer originally claimed. In the face of evidence of infeasibility or increased cost, these requirements may be withdrawn or renegotiated.

Of course, performance analysis and simulation techniques have been in use for many years for computer system modeling. However, most techniques are not fully appropriate early in the system lifecycle. They either depend on unrealistic assumptions in the case of analytic techniques or require numerous parameter estimates and computationally intensive simulation (e.g. TAGS or REVS). These techniques are valuable when the architecture has stabilized and the environment can be modeled with some confidence, but even before that, it is desirable to assess the feasibility of the system. A preferable alternative would be the development of techniques for modeling approximate or bounded properties; somewhere between back of the envelope calculations and heavy-duty simulation. Smith's (1990) Software Performance Engineering method incorporates a number of approximate and heuristic techniques. Unfortunately, it uses a flowchart-like notation that makes it appear less suited to requirements phase modeling than it is.

Challenge 4: Abandoning the tabula rasa assumption

Most requirements analysis research is based on the simplifying assumption that a customer wants a 'new' system. But few systems are new systems. Although the existing system, if there is one, is a rich source of information about the requirements for the new one, the assumption is that the new system will be a total replacement. There are in fact two related assumptions to be retracted: that an existing system may need to

be changed, not replaced; and that the system to be developed is a closed world, not part of an existing system. These assumptions run contrary to two of the most pervasive trends in systems development today: the importance of system evolution, and the increasing importance of system integration.

Challenge 4a: Requirements for evolution

Few system development efforts start from a tabula rasa. Instead of developing a new system, many projects have the goal to build extensions to an existing system. In practice, in such a situation, the only requirements statement the developer sees is often the behavior of an earlier version of the system and the statement to "add X" where X is an ill-defined feature. Requirements documents, and design documents for that matter, are seldom updated when changes are made to a system, so evolutionary development is often a more hit-and-miss process than tabula rasa system development.

Everybody pays lip-service to evolution, but there are few methodological or technological aids available to system developers.

Challenge 4b: Requirements for integration

More companies see themselves in the business of system integration. In system integration, the emphasis shifts away from designing and implementing a system to choosing and gluing together pre-existing parts. In the abstract, this is just another form of system development. In practice, however, there are large differences. For example, the availability of pre-existing components and subsystems is a major factor in choosing among alternatives. The essential model becomes correspondingly less important, and the system architecture more so.

Advances in specifying and analyzing requirements for evolution and integration depend on the development and adoption of better abstraction mechanisms for system development and component reuse. These mechanisms are present in Ada and especially in object-oriented languages. Their increasing use at the design and implementation stages will be a further force in the direction of object-oriented essential modeling techniques.

A longer-term challenge is to develop techniques for predicting the impact of required changes. There are two sub-problems: tracing requirements, so that the locus of an impact can be predicted, and inferring the consequences of the change. The first sub-problem is partly addressed by current traceability techniques, but would be greatly aided by hypertext technology. The second requires more formal models of design components and change than are currently used. Feather (1988) has posited a taxonomy of change types for specifications. However, this work is still in its early stages and has not been applied in practice. There is also a crucial granularity tradeoff to address: the smaller the components, the sharper the prediction, but the higher the update overhead whenever changes are made.

Challenge 5: Bridging the gap from system to project

Requirements analysis research exclusively addresses system specification and requirements elicitation. In practice, a requirements document also must include project planning information such as schedule (often including incremental releases) and cost estimation. However, incremental releases can be defined only after preliminary architectural decomposition. Cost estimation is on firmest ground when based on a reliable size estimate for the delivered system. These needs conflict with the desire to avoid implementation bias and premature commitment to solutions when the problem is still being formulated.

Function point theory (Albrecht, 1979) is one approach to estimating system "scope" independently of size. It has the advantage over lines of code estimates of being based on the essential functionality of the

system. Function points are defined in such as way as to be unsuitable for some types of system, however. The challenge is to devise an alternative with broader applicability.

To address project planning, it will be necessary to develop better ways of tying together system data and project data in a single information repository. Given that there is no simple way to overcome the conflict between the need to delay implementation commitments and the need to plan the project around a delivery schedule, all that can be done is to track carefully the relationship between putative design objects and project activities. As the requirements change and the design components are revised, so the project activities can be replanned accordingly. The problem is very similar to predicting the consequences of change: what is needed is enabling technology that stores the objects and links (i.e. engineering database and hypertext systems), and a more formal model on which to base traceability inferences.

Not so much a challenge as a fruitless exercise: Developing a process model for requirements analysis

There are some false challenges, challenges that I believe that we should avoid because they aim to achieve unrealistic or unnecessary goals. One of these is generic process modeling for phases of the lifecycle, like requirements analysis, that are poorly understood. Different organizational settings require different documentation and procedural conventions. For example, an air defense system is defined very differently from a shrink-wrapped business product. I have never found discussions about the differences between 'verification' and 'validation' or 'requirements' and 'specifications' to be very enlightening, but that is the only type of debate that general- purpose process models and their accompanying flowcharts seem to engender. For that reason, I have scrupulously avoided 'explaining' the challenges by diagrams. This is not to say that process modeling cannot help a particular organization or a particular project, but I do not think that general-purpose process models are precise enough to help yet.

Not so much a challenge as a philosopher's stone: Artificially intelligent requirements analysis

A peculiar twist of rhetoric is often used to justify some applications of artificial intelligence. It goes as follows: "We don't know how to do X. Therefore algorithmic techniques can't be used to do X. Therefore AI techniques may be useful for X." As far as requirements analysis is concerned, all three propositions hold: we don't know how to do it very well, it is intrinsically informal and therefore non-algorithmic in nature, and AI techniques may indeed be useful. But there is of course no sound chain of reasoning to the conclusion. The evidence that AI techniques, such as knowledge-based domain modeling, can be applied to requirements analysis is so far promising but far from convincing.

The evidence from over thirty years of research is that AI has had its most sustained intellectual impact in those areas that were ill understood at the time but amenable to formalization, such as early visual perception or natural language processing, not in those areas that require mimicking creative human problem solving. There are such fields in software engineering. Some of them, such as the intelligent generation of test cases could apply at any stage of the lifecycle, including requirements analysis. However, it would be more prudent to see how AI techniques fare in the better understood downstream phases of development before venturing into the unknown.

In short, advances in requirements analysis are most likely to come from the judicious application of available enabling technologies to methodological principles. Powerful technology independent of methodology is unlikely to help.

References

Aguilera, C. and D. M. Berry, 'The use of a repeated phrase finder in requirements extraction'. *J. Sys. Software 13:* 209-230, 1990.

Albrecht, A.J., 'Measuring Application Development Productivity', *Proc. Joint SHARE/GUIDE Symp.* 83-92, 1979.

De Lisle, N & M. Schwartz, 'Neptune: A hypertext system for CAD applications.' *Proc ACM SIGMOD*, 1986.

Feather, M., 'Constructing specifications by combining parallel elaborations', *IEEE Trans. Software Eng. SE-15:* 198-208, 1988.

Gomaa, H. 'Structuring criteria for real-time system design', *Proc. 11th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press, 1989.

Harel, D., H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman and A. Shtul-Trauring, 'STATEMATE: a working environment for the development of complex reactive systems', *Proc. 10th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press,1988

Heninger, K.L., 'Specifying software requirements for complex systems: new techniques and their applications', *IEEE Trans. Software Eng. SE-6:* 2-11, 1980.

Jackson, M.A., *System Development*, Prentice-Hall, 1983.

Kramer, J., K. Ng, C. Potts and K. Whitehead, 'Tool support for requirements analysis', *Software Eng. J.*, May, 1988, 86-96.

Luqi and V. Berzins, 'Rapidly prototyping real-time systems', *IEEE Software*, Sept., 1988, 25-38.

McMenamin, S.M. and J.F. Palmer, *Essential Systems Analysis*, Yourdon Press, 1984.

Mullery, J. 'Acquisition - Environment' in M.W. Alford, J.P.Ansart, G. Hommel, L. Lamport, B. Liskov, G.P. Mullery and F.B. Schneider, *Distributed Systems: Methods and Tools for Specification, An advanced course*, Springer-Verlag, 1985.

Potts, C., A. Bartlett, B. Cherrie and R. MacLean, 'Discrete Event Simulation as a Means of Validating JSD Design Specifications', *Proc. 8th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press,1985.

Shen, V.Y., C. Richter, M.L. Graf and J.A. Brumfield, 'VERDI: a visual environment for designing distributed systems' *J. Parallel Distrib. Comp. 9*: 128-137, 1990.

Smith, C.U., *Performance Engineering of Software Systems*, Addison Wesley, 1990.

Ward, P.T., 'The transformation schema: An extension of the data flow diagram to represent control and timing', *IEEE Trans. Software Eng. SE-12:* 198-210, 1986.

Wirf-Brock, R., B. Wilkerson and L. Wiener *Designing Object-Oriented Software*, Prentice-Hall, 1990.

Yakemovic, K.C. & J. Conklin, Report on a developement project use of an issue-based information system *Proc. Conf. Computer-Supported Cooperative Work*, ACM, 1990

Zave, P. 'An operational approach to requirements specification for embedded systems', *IEEE Trans. Software Eng. SE-8*: 250-269, 1982.

Requirements Techniques and Tools: A Position Paper

Dennis B. Smith
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pa 15213
(Sponsored by the U.S. Department of Defense)

1. 0 Introduction

Although there is general agreement about the crucial importance of requirements, there are a wide variety of techniques and tools for doing requirements engineering. This session will examine the major functions of requirements engineering, identify techniques and tools appropriate for each function, and ground the techniques and tools in specific experiences of participants with project development. Generalizations will be made as appropriate, and major gaps or untested areas in current techniques and tools will be identified.

A 1989 workshop, "Requirements Engineering and Rapid Prototyping Workshop", sponsored by U.S. Army Communications-Electronics Command Center for Software Engineering (CECOM) identified the following six generic requirements engineering subprocesses:

1) Objective analysis - the analysis of problem space and application domain; deals with description of problems only, not solutions.
2) Objective analysis - the analysis of the solution space and system objectives for life time use.
3) Requirements determination - the specification of characteristics the system must meet to satisfy user needs.
4) Requirements analysis - the analysis of expressed requirements, including related *refinement*, *elaboration*, and correction.
5) Synthesis - the formation of a cohesive specification from the detailed analysis, involving the integration of partitioned analyses occurring due to problem complexity and breadth.
6) Validation - the assurance that the expressed requirements match real user needs and constraints.

These six sub-processes will be used as a point of departure for the session. The session will briefly discuss the sub-processes to determine if these categories represent a reasonable starting point, and whether any changes or amendments should be made to the categories.

For each sub-process, the techniques identified by the CECOM workshop will also be used as a point of departure. These techniques identified by the CECOM workshop are not meant to be all-inclusive or constraining. They were developed by a group of individuals with a strong conceptual grounding in the field of requirements engineering. It is possible that additional techniques will be identified by session participants, and that some of the techniques will be more relevant than others.

Workshop participants will consider the techniques identified by the CECOM workshop, together with additional techniques. Participants will also consider tools which may support each technique. Given the universe of requirements engineering techniques and tools, participants will then be asked to provide project specific experiences for any of the techniques and tools in which they have experience. The intent of this activity is not to validate or invalidate a specific method, but rather to provide a context for the type of domain, organization or project for which specific techniques and tools are appropriate or not appropriate. It is hoped that the participants will bring a rich set of experiences to this activity.

The output of the session will be a filled in matrix of requirements engineering sub-processes, techniques, tools, and project specific experience summaries. Although the experiences of the participants may be quite random, generalizations will be made as appropriate. In many cases, more questions will be raised than answered. In addition, gaps in current techniques and tools will be noted.

2.0 Sub-processes, Activities, and Methods Identified by CECOM Workshop

This section summarizes the sub-process, activities and methods of the CECOM workshop. It will be discussed briefly to obtain consensus on the overall framework of the approach. The framework will be modified as appropriate from the discussion.

2.1 Context Analysis

Activities

Identify problem space boundaries

Needs identification

Application modeling

Postulating solutions

Methods

Interview

Document reviews

Conceptual modeling

Delphi
Group decision support
Analysis
Surveying current systems
Observation
Role-playing
Walk-through
Gaming

2.2 Objective Analysis

Activities

Define specific problem to be solved
Define system/environment boundary and interface
Define life cycle profile
Define user profile
Identify non-functional requirements
Identify critical success factors
Identify operational capabilities
Conduct feasibility analysis
Uncertainty and risk assessments for major objectives
Perform trade-off analysis of major objectives

Methods

Interview
Documentation review
Trade-off analysis
Build scenarios of high level system usages
Delphi techniques
Group decision support methods

2.3 Requirements Determination

Activities

Determine system requirements
Identify alternatives
Perform trade-off analysis
Identify problems, issues, risk
Do planning

Methods

Prototyping
Interviewing
Templating
Reviews with people
Study and observation
market the idea

2.4 Requirements Analysis

Activities

Consistency checking
Completeness checking
Correctness checking
Analyze feasibility
Review testability
Review traceability and linkage
Evaluate significance, certainty and interdependencies

Methods

Prototyping
Structured analysis
Object oriented analysis
Finite state machines
Other specification methods, e.g.,
E-R models, Operational, Petri-net, PSL/PSA, SREM, RLP,USE
Quantitative analysis
View analysis

Ranking, weighting, prioritizing

Scenario building

Simulations

2.5 Synthesis

Activities

Resolve conflicts

Merge models and viewpoints

Integrate concerns

Integrate non-functional and functional requirements

Collect feedback to correct objectives and specifications

Methods

Prototyping

Simulation

Sanity check

Logical modeling

2.6 Validation

Activities

Collect stakeholders critiques, evaluations, reviews, and analyses

Methods

Walkthroughs

Reviews

Inspections

Evaluations of mock-ups, prototypes and simulations

Testing

2.7 Generic requirements activities

These activities and techniques are required throughout the entire software engineering process. They are considered as a separate category to be certain that they are not overlooked within the

specific stages.

Activities

Creating/revising documentation

Creating/revising dictionaries

Recording and checking rationales

Traceability

Impact analysis

Configuration management

Methods

Prototyping

Interviewing

Reviewing documents

Modeling

References

Proceedings of the Requirements Engineering and Rapid Prototyping Workshop, Center for Software Engineering, U.S. Army Communications-Electronics Command, November 14-16, 1989.

Software Requirements: Analysis and Specification, Alan M. Davis, Prentice Hall, 1990.

Requirements Elicitation Working Group

Michael Christel

SEI

Requirements elicitation is a critical phase in the development of new systems. Mistakes made here can greatly increase the cost of development and lead to the lack of acceptance of software systems. It is also the phase in development where the end-users and buyers must be integrally involved, and therefore there are social problems as well as technical problems associated with the elicitation process.

One often-cited problem associated with requirements elicitation is the lack of common knowledge. The requirements analyst typically knows little about the problem domain, while the customer typically does not understand how to build software systems. The analysts, users, and buyers typically lack any common notation or language (Mittermeir, 1982). Moreover, the focus of discussion is often on particular technical solutions and immediate design plans that may lead to inadequate requirements.

There are many approaches to improving the communication between the parties involved in requirements elicitation. Some focus on the social issues, such as improving the interview process and the skill set of the requirements team (making sure they have some domain knowledge, are good listeners, can ask questions and lead discussions, etc.) (Zucconi, 1989, Zahniser, 1990). Others cite the need to capture the *rationale* behind requirements, which clarifies issues involved in systems development and aids in the traceability of decisions made on those issues (Yakemovic, 1990). Others emphasize the need to recognize several views of the system, perform conflict detection and resolution across these multiple viewpoints, and via that viewpoint resolution validate the facts gathered during requirements elicitation (Leite, 1989, Mullery, 1979). Of course, these approaches are not mutually exclusive and elicitation techniques may address a number of these points.

In order to facilitate communication, some approaches to requirements elicitation advocate the early use of prototypes. The level of involvement of prototypes, mock-ups, and scenarios in the elicitation phase is debatable, with some approaches not considering any of these and others being primarily concerned with the immediate development of a prototype (Jordan, 1989). Many techniques recognize the communicative power of limited scope mock-ups and scenarios during requirements elicitation, and advocate this "middle ground" approach (Mittermeir, 1982).

Information management is another issue associated with requirements elicitation (Loucopouios, 1990). Some techniques extend the scope of the information to be managed beyond modeling diagrams and text documents to also include the video interviews of users. Many facts are gathered during elicitation, the facts evolve with time and the introduction of new viewpoints, and the facts need to be easily accessible by people with a variety of backgrounds, e.g., requirements analysts, users, and buyers.

Many of these problems have been mentioned in past workshops on requirements and papers addressing the subject. One of the goals of this working group will be to focus on requirements elicitation and derive a problem set based upon the experiences of the working group members:

What are the problems which must be addressed in order to improve the process of requirements elicitation?

This problem set will probably revisit deficiencies noted in the past, i.e., conflicts among multiple stakeholders and traceability problems. However, along with identifying the problems this working group will also prioritize those problems so that the follow-up discussion can focus on the most important issues.

The working group will use the problem set it derives in discussing what software engineering techniques can be provided to address the most important issues involved in requirements elicitation:

What are the recommendations on how to solve the most important problems in the process of requirements elicitation?

This brief survey of some of the existing work in the area of requirements elicitation will hopefully stimulate discussion for the working group. Rather than endorsing particular models and approaches to elicitation, though, the working group will focus on prioritizing the problems which plague requirements elicitation and on recommendations for solutions to some of these problems.

References

Jordan, Pamela W., Keller, Karl S., Tucker, Richard W., and Vogel, David. Software Storming: Combining Rapid Prototyping and Knowledge Engineering. *IEEE Computer*, May 1989, , 39-48.

Leite, Julio Cesar S.P. Viewpoint Analysis: A Case Study. *ACM SIGSOFT Software Engineering Notes*, May 1989, *14*(3), 111-119.

Loucopoulos, P., and Champion, R.E.M. Concept acquisition and analysis for requirements specification. *Software Engineering Journal*, March 1990, , 116-124.

Mittermeir, Roland T., Hsia, Pei, and Yeh, Raymond T. Alternatives to Overcome the Communication Problem of Formal Requirements Analysis. In Ohno, Y (Ed.), *Requirements Engineering Environments*, North-Holland Publishing Company, 1982.

Mullery, G.P. *CORE: A Method for Controlled Requirements Specification*, pages 126-135. IEEE Computer Society Press, 1979.

Yakemovic, K.C. Burgess, and Conklin, E. Jeffrey. *Report on a Development Project Use of an Issue-Based Information System*. ACM, October, 1990.

Zahniser, Richard A. How to speed development with group sessions. *IEEE Software*, May 1990, , 109-110.

Zucconi, Lin. Techniques and Experiences Capturing Requirements for Several Real-Time Applications. *ACM SIGSOFT Software Engineering Notes*, October 1989, *14*(6), 51-55.

A Computer Supported Cooperative Work Environment for Requirements Engineering and Analysis

by

James D. Palmer
BDM International Professor of Information Technology

Peter Aiken
Visiting Assistant Professor of Information Systems

Ann Fields
Research Associate

Center for Software Systems Engineering
School of Information Technology and Engineering
George Mason University
Fairfax, VA 22030

Abstract

Performance of requirements engineering and analysis for large-scale complex systems is a group activity, involving user representatives from the necessary domain areas. A Computer Supported Cooperative Work (CSCW) environment is described that takes into consideration these group aspects of requirements engineering and analysis. The ways in which a CSCW environment may be utilized to support requirements engineering are presented. An architecture is presented together with the hardware necessary to implement the system. Two applications are examined to show how the CSCW environment may affect the requirements engineering process. Initial results indicate that the use of the CSCW environment provides assistance to user and designer groups in the development of better requirements than would have been feasible without the use of such an environment.

Introduction

A Computer Supported Cooperative Work (CSCW) integrated environment has been designed to support the development and analysis of system and software level requirements for large scale complex applications. This environment supports two key processing needs that are essential in the elicitation of correct, complete, and unambiguous information: 1) the ability to obtain useful and useable information from users, individually or in groups, and 2) the ability to represent information in appropriate media formats. The primary activities supported by this environment include:

- the reduction of barriers associated with the acquisition of multimedia information from user groups;
- the collaborative processes inherent in software development for a large-scale complex system;
- information integration and interchange functions and use of multiple methodologies for requirements analysis; and
- integrated toolset utilization of analysis tools to address problems associated with imprecision, ambiguity, conflict, or other flaws in requirements.

Preliminary results from applications show it is feasible to perform necessary requirements engineering process tasks involving multimedia inputs. It is also possible to provide for interactive and iterative endeavors between the user and requirements engineering teams. Another substantive result, from the use of embedded CASE tools that have been developed, shows that ambiguity, imprecision, and overspecification in software requirements can be identified and corrected prior to these factors having adverse consequences on the project. This in-turn results in the ability to better manage project risk. Through these applications we have shown that the automated assistance provided by the environment may be used to support requirements engineering elicitation, analysis, classification, and design processes. Further, this environment provides assistance to the user and requirements engineering teams in major design activities related to classification; analysis; indexing; validation; generation of test plans; and ensuring traceability of system level requirements.

In the sections that follow we will examine requirements engineering in light of the attributes of the CSCW environment. Next we will present the CSCW environment architecture and discuss this in sufficient detail to show how the system is constructed and the way it works. Following this we will look at the opportunities that we have had to use the system and finally examine the conclusions that we have drawn regarding the utilization of CSCW technology for requirements engineering.

Requirements Engineering and Computer Supported Cooperative Work

The development of systems and software level requirements for large scale complex systems is inherently a group effort. Most significant large-scale complex software systems engineering projects are not developed by individuals, but are conducted by groups of people that include users, systems designers, and general management. There is increasing evidence that external groups, employed at certain stages of the requirements engineering process, can decrease the number of errors and improve the overall quality of software requirements specifications

[Martin and Tsai, 1990]. The approach that we have taken has been to provide an environment that supports and encourages group participation in large-scale complex software engineering programs through the advanced workstations.

The effort may be conducted by a single group or by a group that is spatially and temporally distributed, but nonetheless is a group effort. The environment facilitates the process of requirements development under both of these conditions. There are times when group efforts are repressed by spatial and temporal distance between users. This aspect is supported by the environment, since it is on-line and accessible via modem and the information is stored in easily accessed objects. Thus, through use of the environment it is possible to address these issues by giving the user and designer the ability to easily interact.

Our efforts are aimed at resolving two important aspects of the requirements engineering process in order to support group efforts. These are:

- the environment represents an attempt to implement necessary support for an **integrated** set of basic requirements engineering functions that are performed at some level in all methodologies; and

- it represents an attempt to provide specific **CSCW support** for the software requirements engineering process.

The integration aspects of the environment are essential because they augment and extend the capabilities of users and designers in the performance of basic functions associated with elicitation and processing of requirements information. This permits us to augment and extend the ability of groups to deal with the content of requirements information from the perspective of the overall information picture rather than deal with it on a discrete or incremental basis. The environment also contains tools that provide assistance in the management and resolution of imprecise, ambiguous, conflicting, or overspecified requirements information [Palmer and Aiken, 1990].

We characterize the requirements engineering process to be similar to an *evolutionary software development life cycle* that is primarily concerned with producing software requirements specifications that are feasible, testable, validatable, traceable, and perhaps most importantly, meet the perceived needs of the user. We have found that it is desirable and necessary to involve the users in the process of requirements elicitation and the subsequent validation procedures to assure that the user view of the system is properly represented. The environment that we present provides the support necessary for users and software requirements engineers to take advantage of the interactive and iterative processes that should be involved in requirements engineering. The process is **evolutionary** in that we encourage rapid prototyping at any stage of completion at the choice of either users or designers. These prototypes may be in any of several standard forms used by software designers, including dataflow diagrams, object-oriented design diagrams, datastructure diagrams, or other forms chosen by either users or designers. At the

completion of the process the intent is to produce a set of software requirements current to that point in time.

The environment that we have developed is a hypermedia-based system capable of providing support for two tasks that assist software requirements engineering. These two tasks are minimally necessary for the development of system and software requirements that involve and meet the needs of the user.

1) The environment supports capturing, organizing, synthesizing, and presenting requirements information and encourages a rapid prototyping approach to software development.

2) The environment supports development of the content, not merely the form, of requirements information including the ability to deal with problems such as imprecise, ambiguous, and incomplete requirements information.

It is important in the elicitation process to be able to include hypermedia information that is developed using CSCW techniques. The correctness and completeness of requirements information derived utilizing this approach has the potential of being both **substantively** and **substantially** better than present techniques that utilize only text and graphics. Through this approach we are able to enhance requirements engineering from elicitation through prototype presentation of partial or complete designs. The technical capabilities necessary to accomplish this are given by four distinct properties for the system that are minimally needed to accomplish CSCW tasks. These properties are as follows [Palmer and Aiken, 1990]:

- ***Property 1: Reduction or removal of the artificial barriers to capture and analyses of dynamic real-world multimedia requirements information;***

- ***Property 2: Implementation of an architecture capable of permitting the application of the most appropriate requirements methodology including Computer-Supported Cooperative Work (CSCW) and Group Decision Support Systems (GDSS);***

- ***Property 3: Utilization of an architecture that encourages the integration and interchange of information sources associated with the specific methodologies; and***

- ***Property 4: Implementation of an integrated set of analysis tools to resolve problems associated with imprecise, ambiguous, conflicting, or otherwise flawed requirements.***

Properties one and two support activities aimed at increasing the acquisition of requirements information and taking advantage of the most appropriate technology for this purpose. Properties three and four are aimed at the development of more comprehensive solutions to the problem. Each of these properties is described in more detail.

Property 1: Reduction or removal of the artificial barriers to capture and analyses of dynamic real-world multimedia requirements information.

This is accomplished by:

1) capturing pertinent information concerning the task, the users, and relevant organizational/situational characteristics;

2) organizing this information into readily accessible forms for use in subsequent phases;

3) synthesizing problem solutions from the information gathered; and

4) presenting these solutions to the user to obtain feedback.

Each of these four functions is described in greater detail below.

CAPTURE

The **capture** function facilitates the acquisition of necessary user information in the form of objects. This information may be modified so as to generate such linkages as desired, enabling the presentation of different views through the **organize** and **present** functions. By this, we effectively remove restrictions on the size and/or media format of addressable information objects. Through the use of the Object Management System and multimedia storage devices, we are able to assure the acquisition of significantly more requirements information for use in the development of system and software requirements. Sounds, still pictures, and motion-based input information may be captured and used to supplement text and graphics requirements information. If the actions of individuals are important in the development of a system, for example, the relative physical position or the information flow for a control station, this information may be captured on video tape and used in the design. In addition, all requirements information is maintained on-line and accessible through a single interface.

ORGANIZE

The way in which information that has been captured and serves as input for activities carried on during the remainder of the software development life cycle is depicted conceptually in Figure 1.

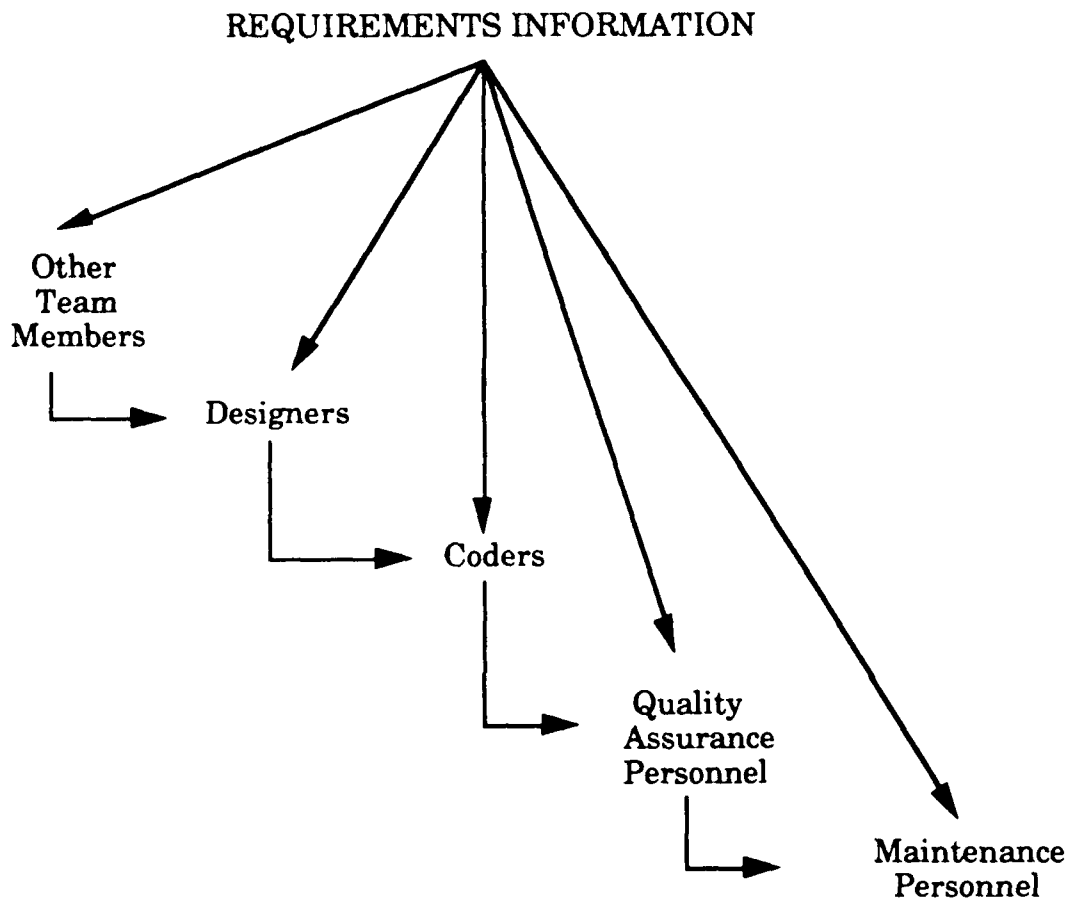


Figure 1 - Uses Of Requirements Information

As we have defined and designed the **organize** function, it acts to extend and augment the ability to quickly and easily access multimedia objects. In many cases, requirements information may be modified through actions that we have denoted as **combination, division**, and **link**, and then be redefined as new objects that have a separate identification Through this process, objects may be grouped into collections based on such criteria as determined by the designer. The basic system design employs object-oriented design (OOD) and an Object Management System for ease of navigation through the requirements, while maintaining the ability to rapidly label, group, and order requirements information.

SYNTHESIZE

We use the term **synthesize** to describe prototype development. The purpose of the synthesize function is to provide the ability to examine and modify information in a manner that extends and augments the processes associated with the development of problem solutions. Activities that we have associated with the synthesize function are as follows: [Palmer and Aiken, 1990]

- Analysis of requirements information to identify constraints which guide prototype development;
- Implementing each requirement in one or more prototype features; and
- Establishing and demonstrating relationships between specific requirements and prototype features.

Through use of the **synthesize** function, we are able to provide the capability to **simultaneously configure** information into hierarchical and directly linked structures by use of a hypermedia requirements matrix. This eliminates the need to force-fit information into a form limited by artificial constraints. It also provides a closer conformity between real situations and internal representations of information structure.

The way in which the synthesize function is utilized is intended to assure that ***visual and cognitive momentum are maintained*** [Woods, 1984] during prototype development. For this purpose, requirements information is maintained in a variety of formats that includes a mixture of media forms. If we fail to maintain visual and cognitive momentum we may lose the current train of thought, waste valuable time, and potentially introduce mistakes into the requirements engineering process.

PRESENT

The fourth basic function is the **presentation** of information from the synthesize function to the user. Production of a prototype early in the requirements engineering process has the potential to yield positive results for the user group. It provides instant feedback as to how the program is progressing, gives the user group several views of the system, reduces the chance of misunderstandings between users and designers, and provides the opportunity to modify the design during the early stages of development. We have provided a number of different formats for presentation of the problem to the users. These include the use of text, text with graphics, graphics, audio, and video, as well as all combinations of these media forms.

Property 2: Implementation of an architecture capable of supporting the application of the most appropriate requirements methodology including aspects of the computer-supported cooperative work and group decision support system; or utilization of multiple requirements methodologies

It has been demonstrated that no single method or approach suffices for successful requirements engineering processes [Sage and Palmer, 1990]. The system architecture that we have implemented for the CSCW environment provides support for many of the requirements engineering tools presently available. Different methodologies may be implemented at different points, as appropriate. We provide support for several aspects of CSCW including:

- group decision support,
- use of common communications techniques;
- distributed groups (spatially and/or temporally);
- techniques for structuring decision analysis; and
- systematically directing the pattern, timing, and/or content of interactions [Palmer and Aiken, 1990].

The environment also facilitates management of the process of creating and tracking multiple versions of system and software requirements. This gives us assurance that we will be able to review the solution approach used and provides a method for post mortum analysis.

Property 3: Utilization of an architecture that encourages the integration and interchange of information sourcesassociated with the specific methodologies.

The CSCW architecture is able to incorporate any of the current CASE tools or provide for any of the other methods that are used for requirements analysis, modeling, specification preparation, and presentation. Another important feature is the ability to interchange information gathered from the application of various methods. An additional attribute is the ability to incorporate multimedia forms. This represents an important departure from most current techniques. Hypermedia technology provides users with the ability to integrate video, audio, graphics, and text formats and provide mechanisms to store and retrieve this information. Through use of the non-sequential linking capabilities provided by the CSCW environment, information exchange linkages are limited only by the ability of the user to determine the forms that are needed/desired.

Property 4: Implementation of an integrated set of analysis tools resolving problems associated with imprecise, ambiguous, conflicting, or otherwise flawed requirements.

As we have noted, the CSCW environment design accommodates any number of tools that are useful during the requirements engineering process. Tools such as CASE tools that are provided by a number of different vendors may be utilized by the system, various simulation and modeling packages may be included, and analytic programs may be provided such as SAS or SPSS.

There are also specific tools that have been designed to address problems that are characteristic of the requirements elicitation process that may be included. These tools are intended to address special concerns related to imprecision, conflict, and overspecification in requirements statements, such as use of quality factors without explicit definition of these terms and their intent, and finally the ability to assign metrics, test tools, and test plans to validate requirements [Myers, 1988, Samson, 1988, and Pfleegher, 1989].

Computer Supported Cooperative Work Environment Architecture

The use of a Computer Supported Cooperative Work (CSCW) environment is primarily concerned with the support of groupwork activities for requirements engineering and analysis functions. The CSCW environment has application in many of the fundamental aspects necessary in the development of software requirements. These application areas include interactive information resource development, information analysis, information retrieval, and the use of group decision support techniques such as conflict resolution, consensus building, and real-time simulation.

For our purposes, requirements engineering is described to be the process of elicitation, analysis, classification, and design of systems and software requirements. The process incorporates a wide range of functions, methods, and approaches that are applied by users and designers. This process is very much concerned with system and software development activities that extend from elicitation of user information to providing for the capture, organization, and synthesis of this information. Finally, the process provides for the presentation of concepts and prototypes that lead to formal software requirements specifications. For large-scale complex systems, this process is almost always a team activity that includes many domain experts who represent the user and team members responsible for software design .

The conceptual architecture of the CSCW environment is shown in Figure 2.

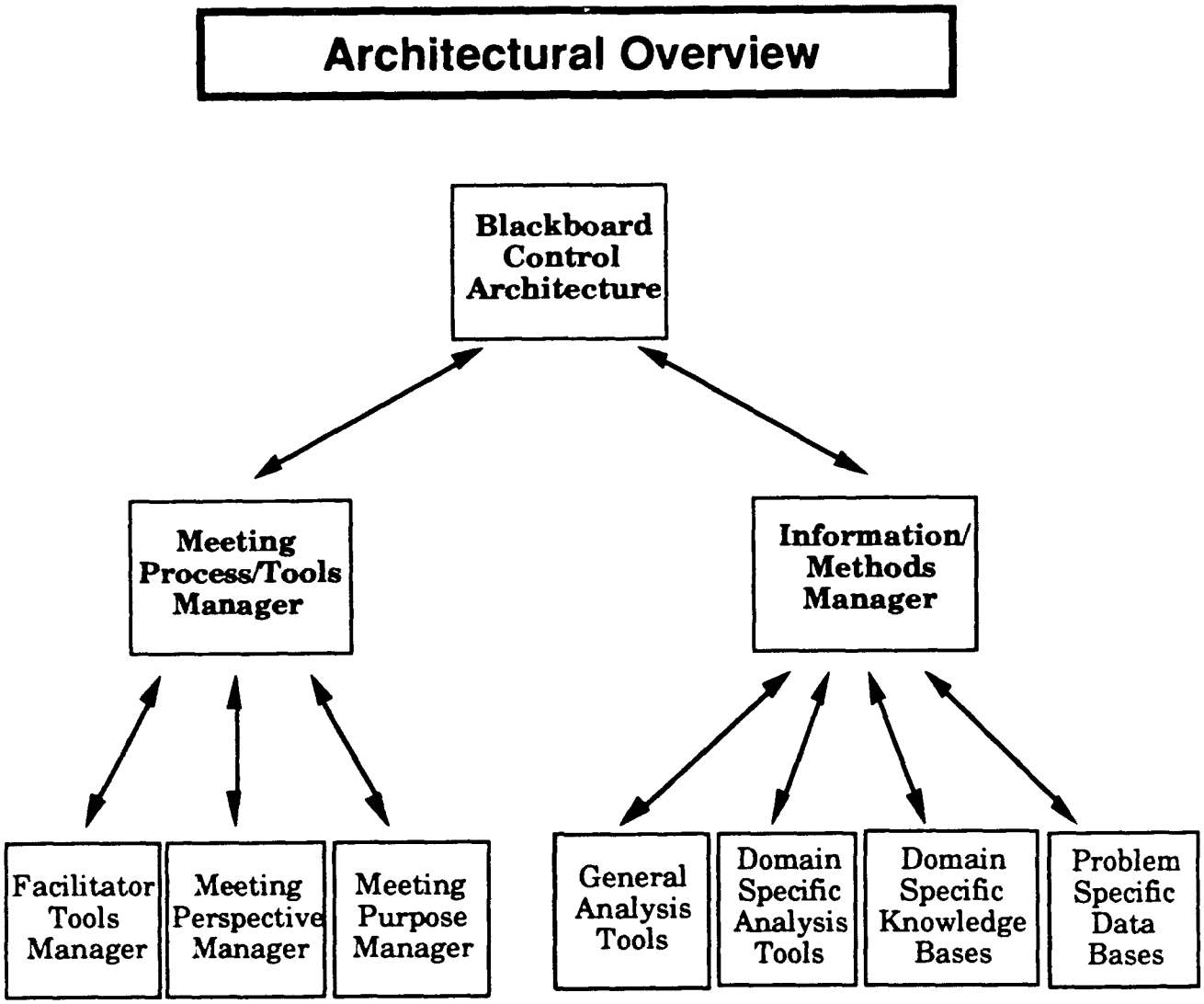


Figure 2. CSCW Architecture

The basic structure of the system utilizes a blackboard control architecture to manage all aspects of the system. Two major subsystems, the meeting process tools control function and the information/methods control function contain a variety of specific tools and activities. The meeting process/tools support the role of the facilitator and the general conduct of the meeting. The information/methods tools contain databases, domain specific knowledge bases, CASE tools, and general tools to assist the designer. These specific functions are depicted in Figures 3 and 4.

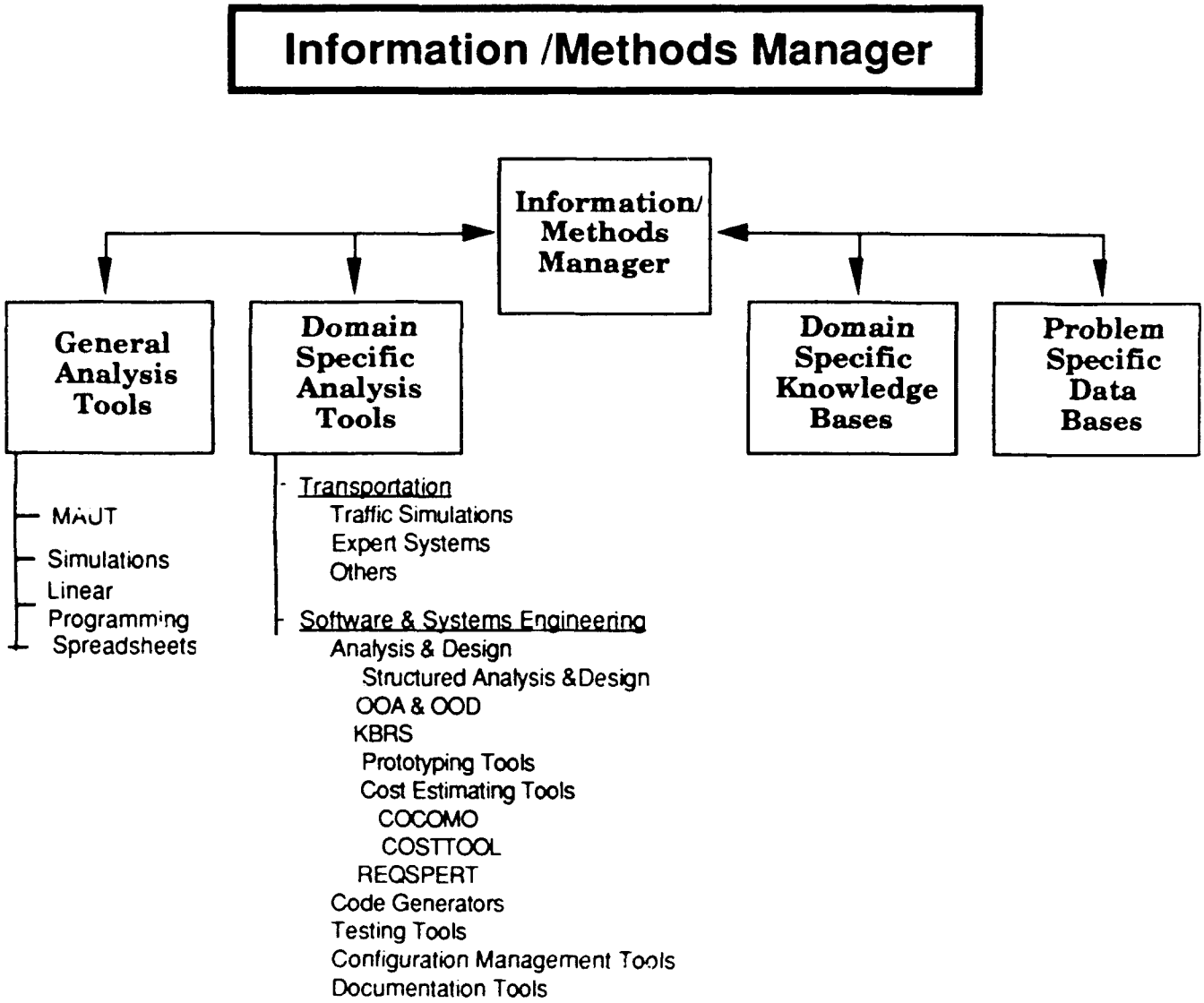


Figure 3. Information/Methods Manager Functions

As may be seen from Figure 3, there are a number of general analysis tools that may be utilized such as, MAUT, general simulation packages and spreadsheets. Domain specific tools are included in the system configuraton as appropriate; for example, when the CSCW environment is used in the requirments engineering domain, tools such as COCOMO, OOA, Code Generators, and KBRS are utilized. These are intended to support the information methods/manager in the development of information to assist in the understanding of the domain problem. Domain specific analysis tools are limited to two domains in the initial environment due to the fact that these are the only areas that have been developed to date. These two domains are in transportation systems analysis, specifically the regional mobility domain, and systems level software requirements domain. The domain and problem specific

knowledge- and databases are also limited at this time to regional mobility and the Howitzer Improvement Program (HIP) of the U.S. Army.

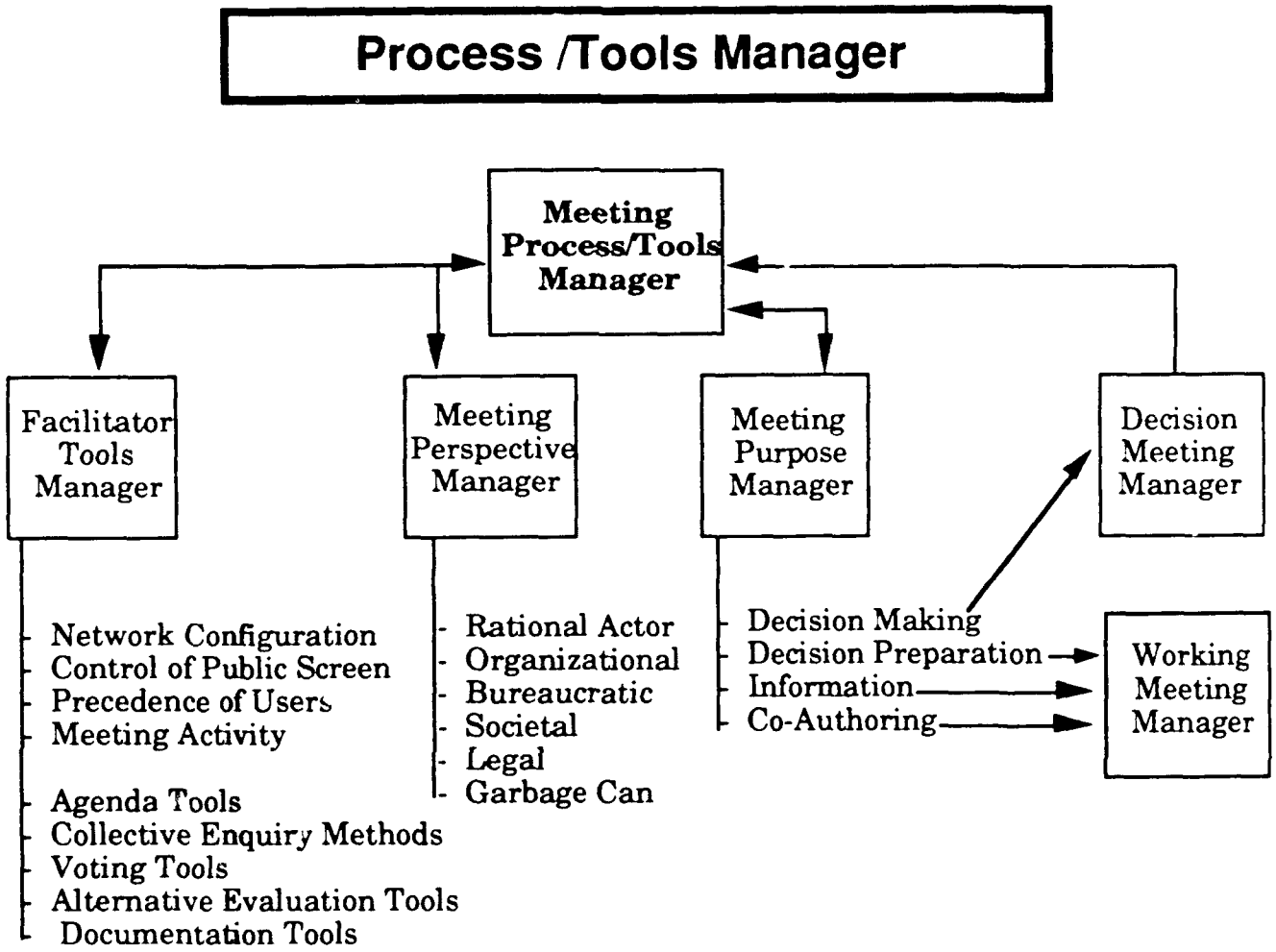


Figure 4. Process Tools Manager Functions

The process/tools manager activities depicted in Figure 4 indicate controls on the nature and character of the meeting. The facilitator has command of a number of configurations that are available for the presentation mode. These include the network configuration control, control of the public screen, agenda, and method of evaluation. The other aspects of the meeting process/tools manager are intended to keep records of the actual conduct of the meeting (or session) for later use in the understanding of how decisions were reached and the process dynamics of the meeting. For example, if the meeting was billed as a decisionmaking meeting and was to be conducted in a rational actor mode, but moved to an information sharing meeting with an organizational perspective, this information would be recorded and the outcome of the meeting analyzed to determine if the will of the group was recognized or compromised or in what way the perspective may have affected the outcomes.

Moving to an additional level of detail, Figure 5 indicates the functions of the Decision Meeting Manager.

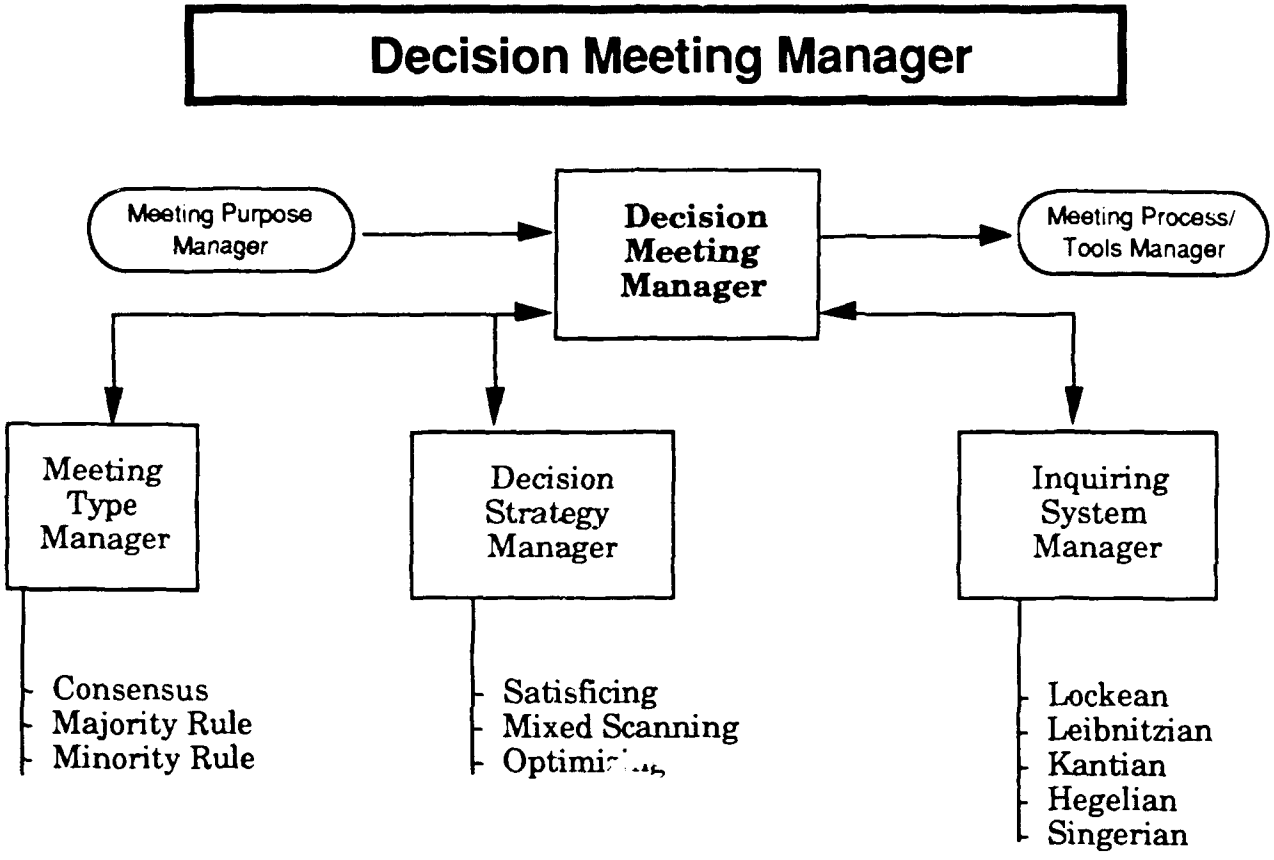


Figure 5. Decision Meeting Manager Functions

The Decision Meeting Manager receives input from the Meeting Purpose Manager. In this component we track the type of meeting, the decision strategy actually used, and the inquiring[1] system employed by the users, and this information is returned to the Meeting Process/Tools Manager.

---

[1] The Inquiring Systems include the following [Churchill, 1971]:

Lockean:	Data Driven, complementary alternatives
Leibnitzian:	Theory Driven, complementary alternatives
Kantian:	Data and Theory Driven, complementary alternatives
Heglian:	Data and Theory Driven, conflicting alternatives
Singerian:	Data and Theory Driven, complementary and conflicting alternatives

Within the environment, all information is converted to objects managed by an Object Management System. The architecture for the Object Management System is shown in Figure 6.

The Object Management System consists of two divisions, an Object Manager and the Object Database. The Object Manager contains the facilitator's window, template screens for ease of producing additional objects, and a menu bar for ease of navigation. It is capabable of creating and modifying objects, creating and maintaining version control, browsing through objects in any order or sequence, dynamic presentation, and annotation. The Object Database contains all multimedia objects, various application programs, the Resource Information System, and the Systems Level Software Requirements System.

This architecture provides us with the necessary systems for deployment and operation of the CSCW platform. The hardware for the platform is a MacIntosh fx with a significant number of peripherals. The hardware in use presently for the CSCW environment is shown in Figure 7. It shows the basic components to implement the architecture. The workstation was constructed with off-the-shelf components. Software was specifically written to support the four basic requirements engineering functions that are necessary for the environment. The basic approach that we have taken is to support highly interactive activities that involve users and designers in all appropriate activities.

Capabilities include the use of video boards for analog video signals, an audio system controlled by the computer, various standard input capabilities including a keyboard, mouse, scanner, and digitizing tablet. Video signals are controlled through use of an Optical Disk Recorder/Player (Write once, Read Many) system. Monitors include a large 37 inch color monitor and a 19 inch color monitor. Other monitors include the standard MAC monitor and a 19 inch black and white monitor. Outputs may be sent to a color printer, standard postscript printer, tape backup unit, floppy disks, or permanent and portable hard disks.

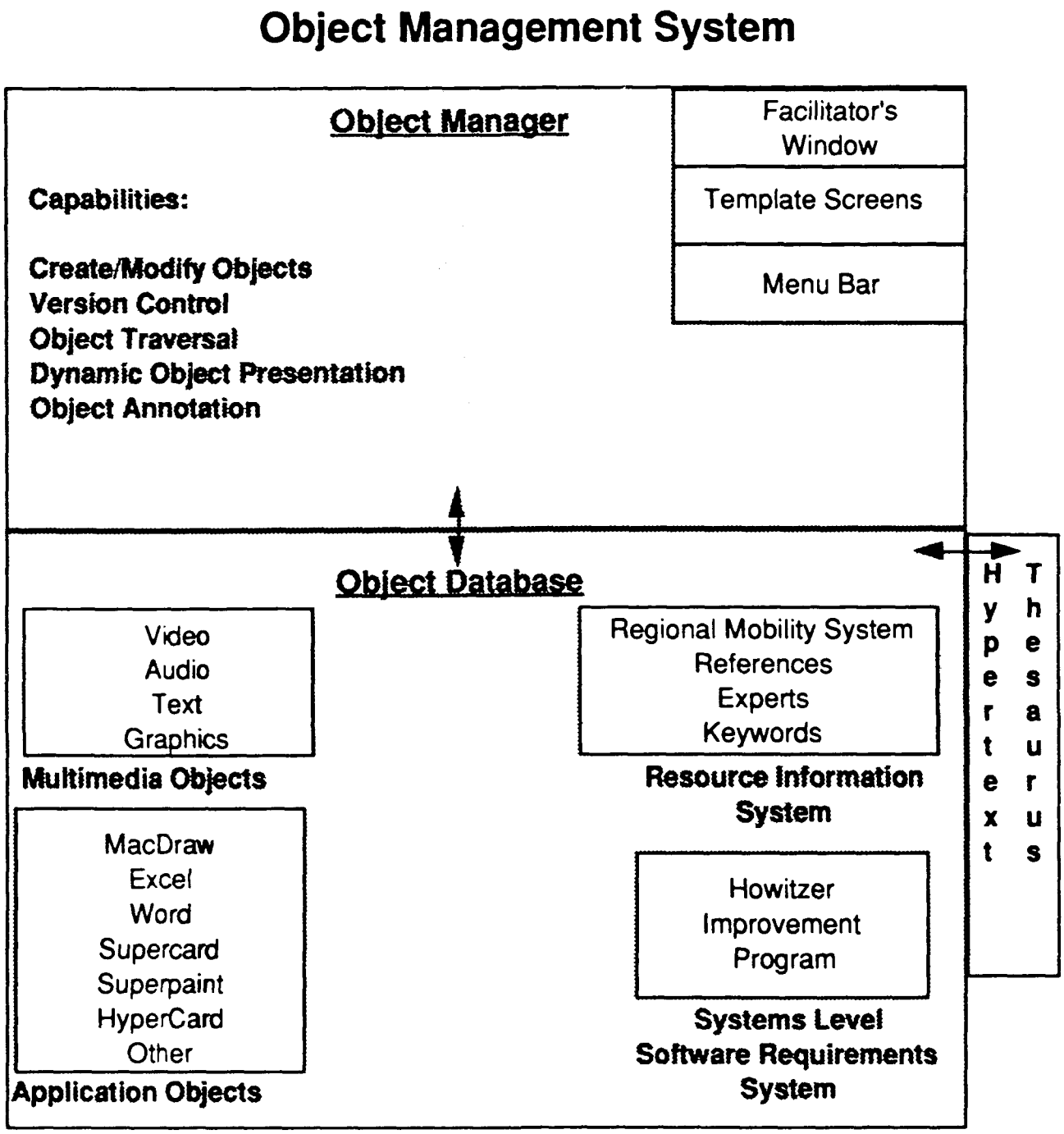


Figure 6. Platform Object Management System

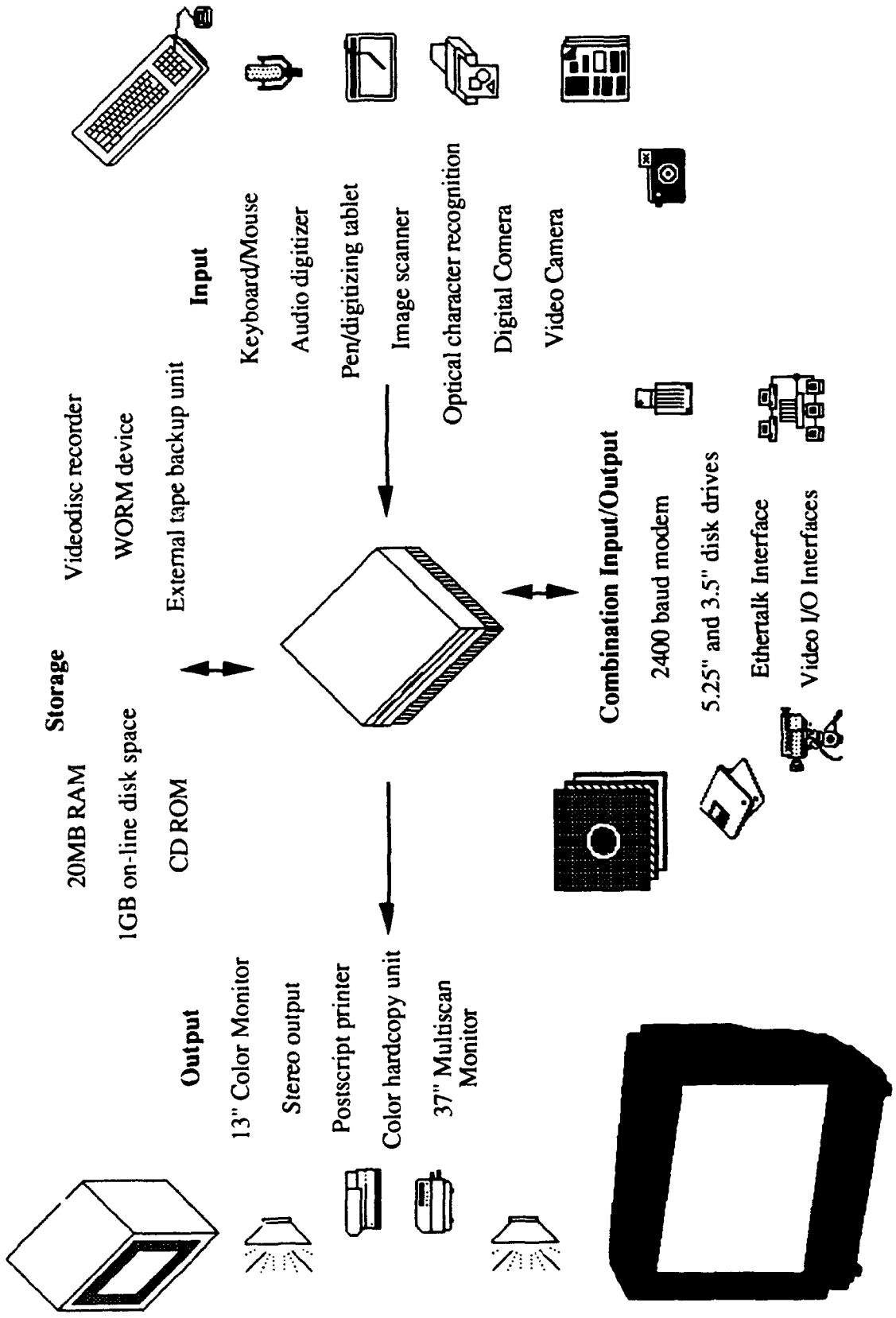


Figure 6. Hardware Configuration for the CSCW Platform

Applications

Applications of the CSCW environment have been limited to research and demonstration efforts at this time. We have applied the system to existing requirements to ascertain if problems in these existing requirements remained after completion of the requirements engineering phase of the software development life cycle and whether or not we would have been able to detect any such problems and correct them. We will review the requirements of a very large-scale system that was prepared by the U.S. Army. Then we will look at our present research work concerning the development of a CSCW environment for purposes of providing a group decision support system for assistance in the resolution of problems related to traffic mobility and congestion.

Software Requirements Analysis of the U.S. Army Howitzer Improvement Program (HIP)

The hypermedia workstation concept was utilized to examine, analyze, and manage software requirements developed for the HIP program. The requirements supplied include the ROC, A-Specifications, B-Specifications, and C-Specifications. Software requirements for the HIP program were developed in the standard format utilizing text and graphic presentation formats. The requirements process was conducted in the traditional way of users getting together to prepare a set of software requirements based on the perceived need of the ultimate user, in this instance the U.S. Army Artillery. For purposes of analyzing the requirements statements for presence of imprecision, ambiguity, conflict, and overspecification, we worked in the CSCW environment to capture the requirements and then organize and synthesize them for purposes of reduction and management of risk associated with statements that contained problems.

We were able to demonstrate that there were many instances of imprecision, ambiguity, overspecification, and conflict remaining in the specifications after these had been completed [Samson, 1989]. While the analysis involved only text and graphics information, the users could have had access to multimedia information that would have aided in the spatial visualization of the ways in which the improved system would have functioned. It is this added dimension, plus the additional analytic tools available in the CSCW environment that leads us to the conclusion that requirements could have been generated that would have had fewer flaws than did the existing ones.

CSCW Environment Utilization in the Resolution of Issues in Mobility and Congestion Problems

Mobility problems plague all major metropolitan areas of the United States (and those of the rest of the world as well). These problems are characterized by long and slow commuting times from residences to places of work and from residences to locations to transact business. While the problems appear to be greatest during commuting hours, the acceleration of the problem during the middle of the day around shopping centers and commercial sites continues to grow.

It has been said that at the end of nearly every trip in an automobile, an economic transaction takes place. This could come in the form of a purchase at a shopping mall, buying lunch, or delivering a pizza. The estimated economic loss due to congestion has been placed at billions of dollars per year. While no hard data exists on this particular figure, the importance of the loss of economic activity certainly places this problem near the top of the agenda for the U.S. Department of Transportation. Thus, there are many groups, constituents, and agencies that have a stake in the resolution of congestion issues and better ways must be found to assist in the resolution of these problems that includes all interested groups.

The resolution of these problems has proved to be extremely difficult, tedious, and time consuming. Analyses of potential solutions of complex problems may take well over six months to develop and present, and, when the analysis is complete, particpants may find that a new cast of players is now interested and that the information has changed. This means that there is an immediate need for a CSCW environment that is able to provide solution information in a timely manner so that real consensus can by built around a given decision path.

The approach that we have taken to assist in the resolution of these problems has been to propose a CSCW workstation that will aid in

- consensus building
- resource allocation
- crisis management
- policy setting, planing
- resource acquisition
- implementation
- design
- education

By definition it is our position that CSCW activities encompass group decision support technologies. These group decision support technologies include

- inter- and intra-group communications
- information sharing
- negotiation
- conflict management and resolution

Each of these factors are present in the congestion problem formulation and each must be addressed for successful problem resolution.

The approach that we have taken after receiving the definition of a specific problem domain has been to build a representative scenario about the problem. For example, we obtain maps of the area under consideration and digitize these for storage and access in the environment, utilize video cameras to record the current congestion situation as a function of time of day and day of the week, obtain all planning materials that are related to the problem area and its nearby areas, obtain all pertinent traffic data, and get any existing forecasts as to any anticipated changes that may be expected to occur. This information is digitized and placed in the scenario database for organization, synthesis, presentation, and retrieval. If there are existing simulations that have been run, we get these and as much data as possible to operate accepted simulation models. In sum, we attempt to obtain all the information that we can possibly get within the time constraints available and enter this information in the workstation. It is then ready for the group meeting that will be conducted to examine alternatives and work to reach a consensus concerning possible solutions to the problems.

Once information gathering and assessment has been accomplished, the concerned parties are invited to attend working sessions that are conducted by a professional facilitator who is skilled in conflict resolution techniques. The facilitator manages the use of the CSCW environment for purposes of bringing all participants the same level of information concerning the problem. The sessions begin with the meeting chair introducing the problem as it is presently understood and additional information is requested from the participants. A review of existing information is conducted and all are asked to verify that this information is correct and representative of the situation. Any modifications must be agreed to by group prior to modification of the databases. Following this introductory activity, the facilitator begins the process of moving the group toward consensus and problem resolution.

This process may involve all of the information that is stored relevant to the current situation or may require restructuring and synthesis of information to provide differing views. It may also require that real-time simulations be run to clarify specific potential consequences of proposed solutions. Whatever the requirement, the facilitator is able to bring to the group any of the information or

the tools to manipulate this information that is available in the system to assist the group to reach a consensus on the problem solution.

An overview of the way the CSCW system is utilized for an information query in the Transportation System Regional Mobility Domain is shown graphically in Figure 8.

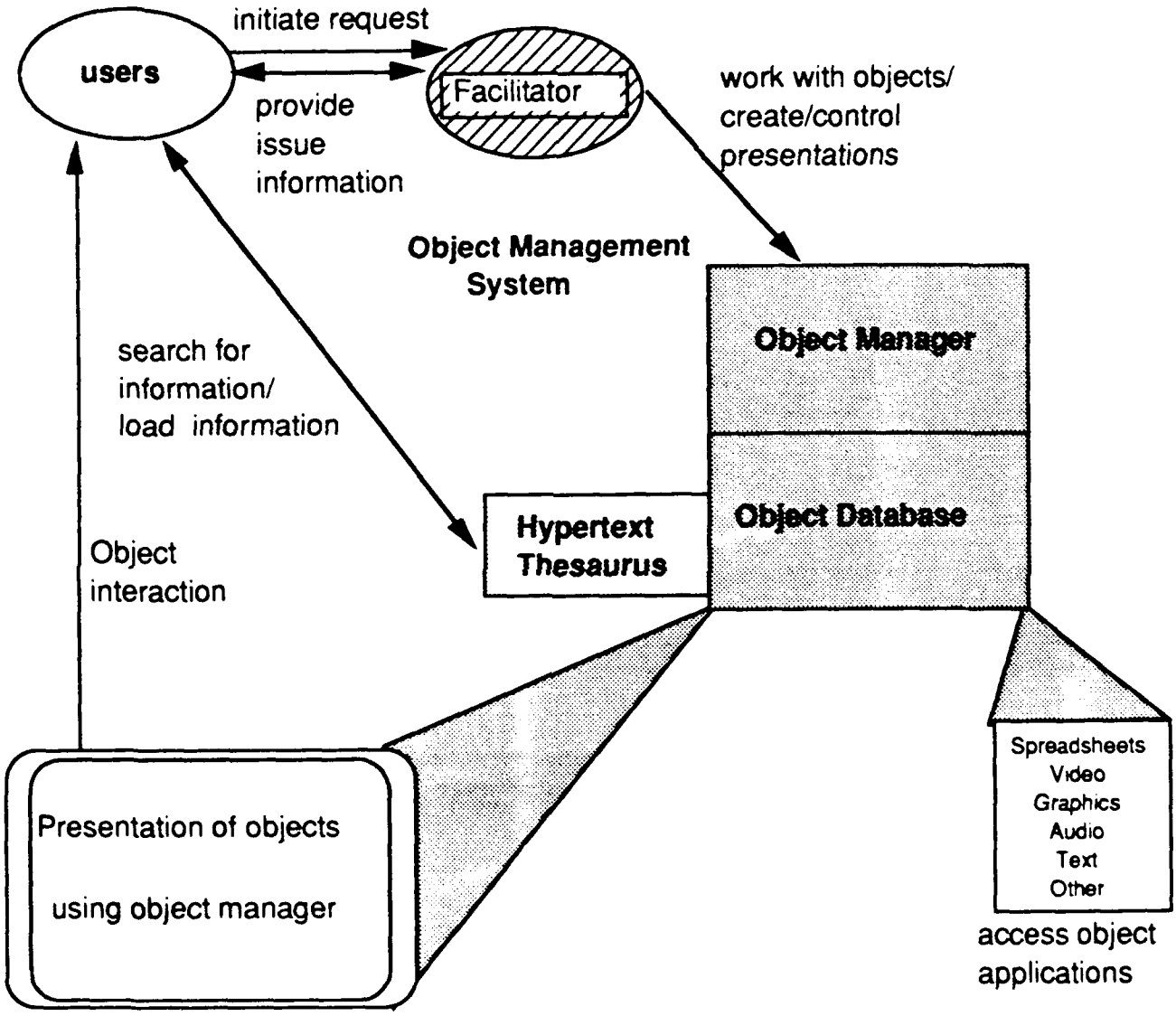


Figure 8. Resource Information Center Query in the Transportation System Regional Mobility Domain

The process shows that a user makes an input or forms a query to a facilitator. The facilitator utilizes an electronic thesaurus to point to the information location. Through an iterative process the facilitator continues to refine the query until the information sought is determined to meet the needs of the user. Following this the

facilitator develops the presentation interactively and iteratively with the user until the final form is accepted by the user. Then, depending upon the way in which the user intends to utilize the information, a presentation is developed and given to the audience selected by the user. This may be in the form of a bibliographic listing, a list of available experts in a specific field, combinations of these, or as a formal presentation to a group meeting for CSCW purposes.

A more detailed view of the functioning of the Resource Information Center processes is shown in Figure 9.

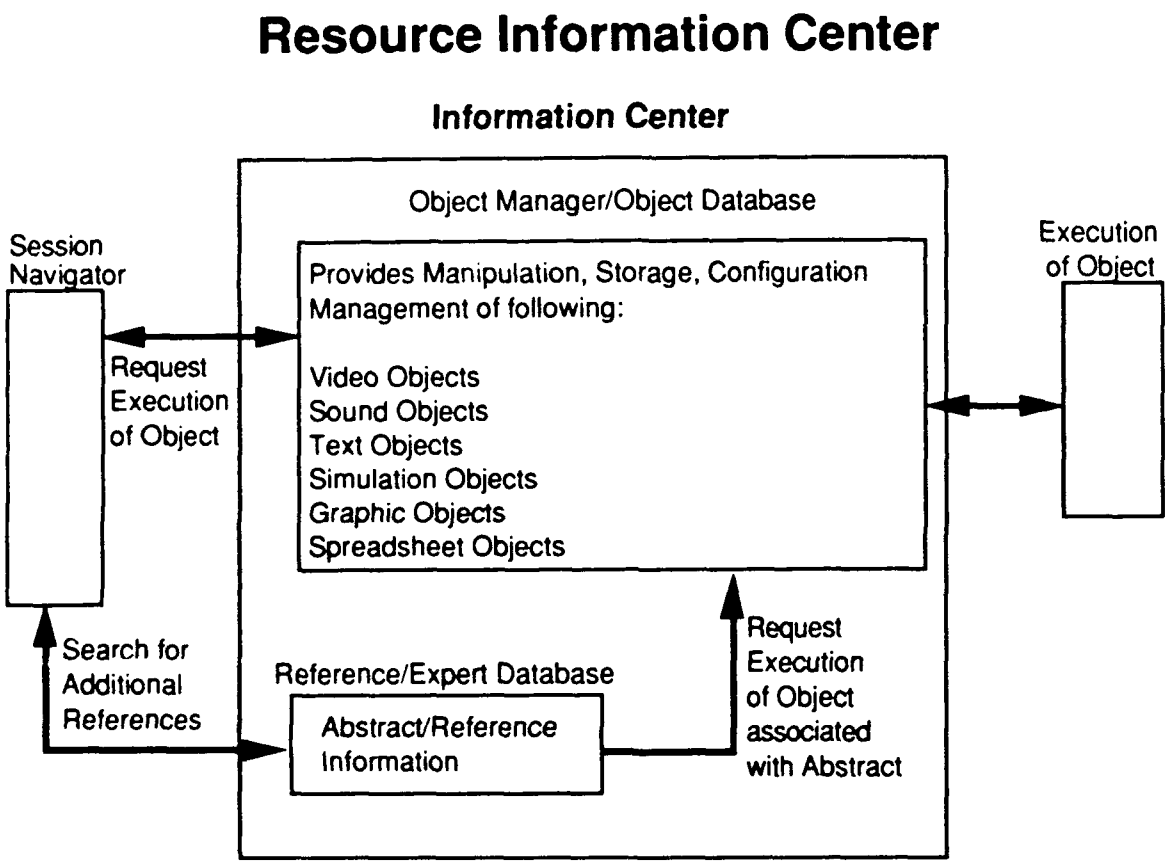


Figure 9. Overview of the Resource Information Center Functions

In this view of the operation the CSCW environment, the objects in the Object Manager are all related to the regional mobility domain. The facilitator interacts with the system via the sessions navigator to select objects, review these with the user, search for additional references, keywords, or experts in the domain and then present this collection of objects to the user. Other domains are constructed as needed to support the CSCW environment.

Conclusions

A Computer Supported Cooperative Work environment has been developed that is able to incorporate the important aspects of hypermedia capability. We have been able to deal successfully with very large amounts of highly complex data from a variety of media sources. We have been able to incorporate video, audio, text, and graphics into a single workstation and provide for the capture, organization, synthesis, and presentation of this material. We are able to take advantage of the model based management system features of the environment to run simulation models, analytic models, and group results together for presentation. We have been able to provide assistance for a facilitator in carrying out the process of consensus building.

Sessions with groups working on relatively well formulated problems have shown that the CSCW environment is a powerful adjunct to the facilitator in attempts to reach consensus. It appears that the fact that all parties are dealing with the same information and have equal ability to have the information organized, reorganized, synthesized, and re-synthesized on demand facilitates the task of moving a group toward consensus. Clearly, the CSCW environment is not a substitute for the facilitator nor will it be any better than the inherent quality and quantity of relevant information. However, it does appear to be able to enhance and extend the basic tools used by the facilitator in guiding the group through the consensus building process.

The tests that have been run on the CSCW environment to date have not been adequate to reach any definitive quantitative conclusions concerning improvements in either software requirements engineering processes or for providing assistance to a facilitator for purposes of consensus building. Our research and development goals and objectives over the next several months are to develop several more robust scenarios with regard to mobility and congestion and place the CSCW environment to test as to its efficacy in assisting facilitators.

References

Aiken, P., 1989, *A Hypermedia Workstation for Requirements Engineering* (Ph.D. dissertation), George Mason University, 1989.

Andriole, S. J., 1986, *Handbook for the Design, Development, Evaluation, and Application of Interactive Military Decision Support Systems*, Marshall VA: International Information Systems, 1986.

HyperCard User's Guide, 1988, Apple Computer, 1988

Aseltine, J., Beam, W.R., Palmer, J.D., Sage, A.P., 1989, *Introduction To Computer Systems: Analysis, Design and Application*, John Wiley & Sons, Inc., New York, 1989.

Beam, W.R., Palmer, J.D., and Sage, A.P., 1987, "Systems Engineering for Software Productivity," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 17, No. 2, March/April 1987.

Boar, Bernard H. 1984, *Application Prototyping: A Requirements Definition Strategy for the 80s*, New York: Wiley-Interscience, 1984.

Bolt, R. A, 1984, *The Human Interface: Where people and computers meet*, Belmont, California: Lifetime Learning Publications, 1984.

Brooks, F. P., 1988, "Grasping Reality Through Illusion - Interactive Graphics Serving Science," *Proceedings: ACM/SIGCHI Conference on Human Factors in Computing Systems* May 1988, pp. 1-10.

Bush, V., 1945, "As We May Think," *Atlantic Monthly* July 1945, pp. 101-108.

Churchman, C., 1971, *The Design of Inquiring Systems: Basic Concepts of Systems and Organization*, Basic Books, Inc., New York, 1971.

Conklin, J., 1987, "Hypertext: An Introduction and Survey," *IEEE Computer* September 1987, 20(9):17-41.

Engelbart, D., 1962, *Augmenting Human Intellect: A Conceptual Framework* Summary Report, Stanford Research Institute, on Contract AF 49(638)-1024, October 1962, 134 pp.

Goodman, D., 1987, *The Complete HyperCard Handbook*, New York: Bantam Books, 1987.

Gookin, D., 1989, *The Complete SuperCard Handbook*, Compute! Publications Inc., Rador Pennsylvania, 1989

Gray, S. and Shasha, D., 198[illegible] "[illegible] .npirical guidance for the design of nonlinear text systems," *Behavior Research Me[illegible]ds, Instruments, and Computer* 1989 21(2):326-333.

Halasz, F. G., 1987, "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems" *Hypertext '87 Papers*, pp. 345-366.

Hogan, T. and Swaine, M., 1988, "The Great HyperCard Debate," *Bay Area Computer Currents* October 18-October 31, 1988, 6(11):38-43.

Martin, J. and W.T. Tsai, 1990, "N-Fold Inspection: A Requirements Analysis Technique," *Communications of the ACM* February 1990, 33(2):225-232.

Nelson, T. H., 1987, *Computer Lib*, Redmond, WA: Microsoft Press, 1987

Palmer, J.D., 1987, "Expert Systems Use in Software Productivity," Proceedings, *1987 IEEE Conference on Systems, Man, and Cybernetics*, Washington, D.C., October 1987.

Palmer, J.D., 1987, "Uncertainty in Software Requirements," *Large Scale Systems*, Vol. 12,1987, pp. 257-270.

Palmer, J.D., 1988, "Impact of Requirements Uncertainty on Software Productivity," *Proceedings, Twenty-seventh Annual Technical Symposium*, Washington D.C. Chapter Association for Computing Machinery, June 1988, pp. 15-85.

Palmer, J.D., and Myers, M., 1988, "Knowledge-based Systems Application to Reduce Risk in Software Requirements," *Proceedings, Uncertainty and Intelligent Systems, 2nd International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Urbino, Italy, July, 1988, pp. 351-358.

Palmer, J.D., Sage, A.P., 1988, "Cognitive Models and User Interfaces for an Advanced Software Systems Engineering Development," *Revue Intenationale de Systemique*, Vol. 2, No. 2,1988, pp. 195-214.

Palmer, J.D., Samson, D., Myers, M., 1988, "Resolving Risk in Software Requirements Definition," Proceedings, *National Conference on Software Reliability and Testing*, Crystal City, Virginia, November 1988, pp. Gl - G28.

Palmer, J.D., 1990, "Information Systems and Software Productivity," *Concise Encyclopedia of Information Processing*, McGraw-Hill Publishing Co., Inc., New York, 1990.

Palmer, J.D., 1990, "Software System Requirements Engineering for Command and Control," in *Advanced Technologies for Command and Control Systems Engineering*, S. Andriole, Ed., AFCEA International Press, Fairfax, VA, 1990.

Palmer, J.D., 1990, "System Level Requirements Specifications for Command & Control: An Approach to Risk Management & Volatility Reduction," in *Advanced Technologies for Command and Control Systems Engineering*, S. Andriole, Ed., AFCEA International Press, Fairfax, VA, 1990.

Palmer, J.D. and Aiken, P. H., 1990, "A Hypermedia Environment Supporting the Development of Software Requirements," submitted to IEEE *Software*, 1990.

Pozeshy, M. T. and Mann, M. K., 1989, "The US Air Traffic Control System Architecture," *Proceedings of the IEEE*, November 1989, 77(11):1605-1617.

Sage, A.P., and Palmer, J.D., 1990, *Software Systems Engineering*, John Wiley & Sons, Inc., New York, 1990.

Samson, D., 1988,*A Knowledge-based Assistant for Software Requirements Analysis*, PhD Dissertation, GMU, 1988.

Samson, D., 1989, "EXTEND: Automated Assistance for Test Plan Generation," Sonex Enterprises, 1989.

SuperCard User Manual, 1989, Silicon Beach Software, 1989.

Webster, D. E., 1988, "Mapping the Design Information Representation Terrain," *IEEE Computer*, December 1988, 21(12):8-23.

Woods, D. A., 1984, "Visual momentum: A Concept to Improve the Cognitive Coupling of Person and Compter," *International Journal of Man-Machine Studies* 21:229-244.

WHITE PAPER

An Informal Approach to Developing an Environment for Requirements Capture and Refinement

Elizabeth S. Kean
Rome Laboratory
Griffiss AFB NY 13441-5700

I. Introduction

The definition of a requirement is "something that is required" or a "necessity". In the system and software engineering world, requirements specifications are precise statements of need intended to convey some understanding about a desired result. They describe external, user visible characteristics of the needs as opposed to internal construction of the solutions to those needs. They also specify the constraints placed on what is needed. For example, performance, reliability, safety, cost and schedule are typical constraints.

Requirements specifications of large command, control, communications and intelligence (C3I) systems are often ambiguous, inconsistent and incomplete. The verification and validation of the requirements for these systems is a manual process. Studies have indicated that as many as fifty five percent (55%) of the errors reported during the life of a C3I system occur in the requirements phase. If the errors go undetected until later phases, the cost to correct them increases significantly. As a result, individuals involved in the development of system and software requirements have identified the importance of careful requirements analysis activities and various disciplines for requirements specification, verification and validation have emerged. Rome Air Development Center (RADC) has been conducting a research and development program in requirements engineering since 1980. Its goals are to develop methodologies and supporting tools for problem analysis, specification and validation.

II. The Model

One of the first efforts of the RADC program was to convene a panel of requirements engineers and researchers with the charter to develop a long range plan for work in this area. The panel developed a process model (see Figure 1) which goes beyond existing models in its characterization of requirements engineering. It provides a detailed description of the fundamental activities occurring during requirements engineering, but avoids prescribing specific methods for accomplishing them. It does not separate the analysis, specification and validation of requirements into discrete operations, but instead elaborates on how analysis and specification are used together from the very outset of requirements engineering, on how static analysis and validation cause the entire process to be highly iterative and on how validation is related to design activities, naturally occurring before the commitment to detailed design and implementation activities. The model recognizes three basic activities being performed: eliciting requirements from the various individual sources of a system's requirements, specifying them and insuring their consistency; insuring that the needs of all users are self-consistent and feasible; and validating that the requirements so derived are an accurate reflection of user needs.

The model begins with a description of requirements analysis activities. Requirements originate in the minds of the people who will eventually use the system they are trying to characterize. Initial requirements are usually thought of as "wish lists", because they may be totally or partially infeasible, they typically exhibit the undesirable properties of being inconsistent, incomplete, ambiguous, untestable, and they are undocumented. Therefore, the first activity in the requirements engineering process is to document and refine each wish list. This implies that all relevant viewpoints (i.e., users of the target system, other systems which must interface with the target system or environmental factors) must be identified and organized by function. All of the important activities and data of each viewpoint must be described to a satisfactory level of detail. Finally, all of this information must be analyzed for self-consistency. For example, if the requirements are being represented as data flows, then all process inputs must be produced somewhere, all naming conflicts must be resolved, levels must be balanced, etc.

The results of these efforts are called "goals" and are the first documented statement of user's needs. They typically are high-level in nature, specific to the relevant problem domain and stated in the user's own terminology. Goals are still relative to the user viewpoint which they represent; they may still be infeasible; and they probably conflict with the goals represented by other viewpoints. However, within each viewpoint goals are self-consistent and unambiguous as a result of several iterative analyses.

Once consistency of the goals within each viewpoint is established and verified, the next step is to integrate the goals across the various user viewpoints thereby resolving the conflicts among them. User scenarios, whcih can be thought of as dataflow threads throughout the system, are typically relied on to identify the activities and data which are the basis for integrating goals which involve several viewpoints. The ensuing integration analysis demands that all involved viewpoints agree on which viewpoint will accept responsibility for an activity and which will provide the data necessary to carry it out. Consistency checking is an important part of this process. Finally, the non-functional requirements, such as performance and reliability issues must also be considered and stated. The resulting requirements then, are an unambiguous, consistent, feasible subset of the goals which describe a number of possible solutions. The question now remaining is whether or not they are what the user actually intended.

The requirements engineering process model provides several options for verifying that the documented requirements meet the user's needs. One is dynamic analysis of the requirements themselves. The intent is to "animate" the requirements by providing sufficient functionality to enable the requirements engineer to perform a "walkthrough" of requirements with the user. The walkthrough is an exercise conducive to exposing misunderstandings, omissions and interface problems. The dynamic analysis utilizes the activities derived during the second step of the requirements analysis and activates them with actual data which is representative of classes of system input. An activity is "animated" by utilizing algebraic and logical functions to represent the activity or by actually programming the data transformation which it represents. The walkthrough typically produces an interchange between user and requirements engineer which results in the elicitation of valuable user commentary and the validation of requirements.

The remainder of the options prescribed by the process model involve design of critical system components. The resulting partial system designs are then made the subject of either analytical studies or of prototyping. The analyses focus on mathematical modeling of system performance and reliability. Various models may also be utilized to study the cost and schedule risks associated with a particular system development. Prototyping involves the construction of user interfaces, actual system functions and discrete event simulation models. Behaviors exhibited by the resulting user interface and functional prototypes are validated through actual usage by the user.

Performance modeling results are usually analyzed by individuals skilled in this discipline.

Regardless of which path through the model is followed or if all are pursued simultaneously, the outcome is a new appreciation of user needs which invariably leads to a re-evaluation of the requirements as previously stated. This activity in the process model leads to a new set of requirements which once again is the subject of analysis and validation activities. This process is repeated until users, requirements engineers and designers are satisfied that the requirements and partial designs have reached a maturity which permits development of the target system to proceed.

III. Process Model Tools and Methodologies

The process model, by itself, is a step forward in the specification, verification and validation of requirements. However, tools and techniques that will automate this process are required. There are two approaches that can be taken to develop methods and tools to support the process model. The first approach would be to develop a single formal language for expressing goals, requirements, and solution architectures, all which can reference their domain model. Within this approach, there will be the ability to formally interpret the goals and requirements as predicates against the solutions and their behaviors.

The other path, more informal than the first, involves gathering and integrating existing tools and methods to support the requirements engineering process model. Within this approach, there are partially formal characterizations of the problem domain, functional and non-functional requirements, interface protocols, and solution architectures. There is support for organizing the problem domain, developing a functional description of the system, building executable models, interface models, and performance models.

IV. The Informal Approach - Requirements Engineering Environment

In order to provide tools and aids to support the model in the near term, RADC is in the process of developing an environment that follows the informal approach with the plan to evolve the formal language approach over the next several years. The approach to developing this environment, called the Requirements Engineering Environment (REE), is to integrate existing methodologies and tools into a single unified methodology for requirements analysis, specification and validation. The major effort here is the design and implementation of a common data repository and user interface for the tools. The common data base will utilize an object manager to store all information which is not exclusively used by the tools, thereby allowing direct sharing of information among all tools. The user interface will provide uniform, object oriented access mechanisms to the tools. The initial tools to reside in the REE are tools in which RADC has been involved in the development and enhancement of, and which support a subset of the activities described in the process model including requirements analysis, specification, prototyping and validation. Figure 2 identifies the activities supported by the initial methodologies and corresponding toolset and the following is a brief description of the functionality of each methodology/tool to demonstrate how it supports the process model.

Controlled Requirements Expression (CORE) is a requirements definition and analysis method whose procedures explicitly support the notion that requirements originate from several, diverse viewpoints of how the same needed capability will be used. CORE organizes these viewpoints as a functional hierarchy. CORE procedures prescribe data flow techniques to elaborate the viewpoints. Logical data relationships are described using a notation similar to that used in the Jackson System Design. Self-consistency of each viewpoint is a goal of the CORE method and is supported by specific checking procedures. Transaction analysis is used to resolve conflicts among the various

viewpoints. Constraints analysis (e.g., performance, reliability) is also an explicit step in the CORE method.

The Analyst, a proprietary tool developed by Systems Designers, Inc. and enhanced by RADC, is an expert system for requirements analysis and specification applying the CORE method. It provides documentation and analysis tools which support and enforce the viewpoint hierarchy and data flow rules of the CORE method. Capabilities include diagram construction, management and consistency checking in support of CORE analysis activities and word processor support for textually oriented aspects, e.g., describing project objectives, performance constraints, etc. Analyst provides an intelligent "help facility" which understands project progress in terms of CORE objects (e.g., diagrams) and strategically guides the user toward project completion in accordance with method rules. Analyst also implements dynamic analysis of requirements providing tools which support the requirements engineer in conducting a user "walkthrough" of typical system transactions.

The Rapid Prototyping System (RPS), developed by Martin Marietta, is a collection of methodologies and supporting tools which support the activities of building, executing and analyzing prototypes of computer based systems for the purpose of improving understanding of the requirements for those systems. The RPS consists of user interface and computer system modeling capabilities which collectively provide an integrated environment for rapidly prototyping critical C3I functions. These capabilities focus on prototyping the high-risk, high payoff aspects of C3I systems such as the user interface, system communications, database management, operator/analyst work flow and system performance.

The RPS user interface modeling tools provide the user the capability to quickly generate a demonstration or prototype of the user interface of a C3I system. A high level graphical editor provides the capability to draw the graphics for the user interface and menu driven templates, allow for functionality to be tied to the graphics. The RPS tools generate the source code automatically, provide for the compilation and linking of the source code into an executable demonstration, and a mechanism for invoking the demonstration. The prototype developer may add any additional capabilities to the demonstration by simply coding and linking into the automatically generated source code. Once the demonstration is developed and presented to the user, comments and feedback can be easily incorporated into the demonstration until the demonstration represents the requirements for the user interface of the system. Demonstrations can be developed that provide both dynamic and static user interface displays.

The RPS provides a template-driven interface to a performance prediction capability which enables the prototype developer to rapidly construct discrete event simulation models to assess the following system performance drivers: computer hardware and software configuration; operator/analyst terminal work flows; allocation of system functions to system resources; and computer network configuration. Lastly, the RPS provides a graphical data modeling component to allow the prototype developer to quickly construct different logical views of a relational data base and to assess their potential for satisfying user query/update needs and system response requirements.

The Very High Level System Prototyping Tool (PROTO), developed by International Software Systems, Inc (ISSI), is a high-level specification and prototyping language with integrated tools supporting specification manipulation and a library of reusable specification components. PROTO includes a methodology for functional prototyping, in which the logical capabilities of a proposed system are modeled. PROTO is an executable specification language and so complete or partial specifications can be validated through interpretation (i.e., provided with inputs so that output

behavior may be examined). PROTO implements a data flow specification model. The specifier creates a graphical representation of a target system or subsystem. This data flow graph defines the system's processing activities, its inputs and their origins, the outputs which are produced and which processing activities utilize those outputs. The algorithmic details occurring within any processing activity may be specified at that level or may be postponed to some lower level of abstraction, thereby providing the specifier with an iterative, step-wise refinement methodology. Finally, at some level in the specification an activity's processing must be detailed. For this the specifier has three options. A behavior language internal to PROTO may be used to specify the algorithms which transform and pass data from input ports to output ports. If the processing becomes complex, a programming language, such as C, may be employed. However, both of these programming approaches are time-consuming and costly. Hence, the PROTO library of reusable components may be browsed, using keyword searching techniques, to locate components which are appropriate to the current process development. The located component may be better understood through the use of documentation maintained internally by PROTO and by executing the component. Finally the component may be integrated into the evolving specification using the same graphical interface being used to develop the specification.

The REE common database will provide an object oriented data repository for information that will be used throughout the process model activities. The purpose of the common database is twofold. First, the data generated by developing the CORE, RPS, and PROTO prototypes will be stored in a centralized location. This will allow other tools the ability to extract the information for document preparation, requirements traceability, etc. Secondly, the three existing tools have some overlapping capabilities. For example, the CORE Analyst has the capability to animate a set of transactions of the system under development, while PROTO provides a more sophisticated mechanism for executing specifications. The REE database will provide a mechanism to store the information generated by the CORE Analyst in the database in a format acceptable to the PROTO tool. Similarly, all data that is determined to be sharable between the three tools and potentially by new tools will reside in the REE common database and mechanisms for sharing will be developed.

The REE database is designed to be extensible, such that new classes can be added to the data base as new tools are integrated into the environment. For example, one of the major areas of the process model in which the REE does not contain a methodology or set of tools is in the area of performance, reliability, cost and risk analysis. Work is in preparation to enhance the REE to include a set of knowledge based tools to model requirements using several different forms of representation and store them in a knowledge base whose self-consistency is maintained by supporting tools.

V. Conclusions

The Requirements Engineering Process Model provides a mechanism for generating validated and verified requirements that are unambiguous, consistent and complete. The Requirements Engineering Environment provides us with a near term solution which provides an informal approach to the generation of these validated requirements using the methodology of the process model. Further research is needed in the formal language approach. An initial RADC effort, which sought to exploit the features of an existing logic programming language, Prolog, to capture requirement semantics in a conceptual model, began in the fall of 1987. Its objectives were to determine the feasibility of modeling a system's operational requirements within the context of its application domain and of analyzing the resulting knowledge bases to determine the impact of various real world scenarios on the system model. These results are useful for making early assessments of system design options. The Prolog language proved to be sufficiently powerful to model important aspects of an air defense domain thereby establishing the feasibility of this

approach. However, an immature Prolog development environment made model implementation difficult, especially the construction of a user interface. Future work will further develop these kind of conceptual modeling techniques.

REFERENCES

[1] B. W. Boehm, "Software Engineering," TRW Defense Systems Group, Redondo Beach, CA, Tech. Rep. TRW-SS-76-08, Oct. 1976.

[2] P. Daley, et al., "Rapid Prototyping System - Users Manual", Rome Air Development Center, Griffiss AFB NY, Tech Rep., June 1988.

[3] A. Ege et al., "Requirements Engineering Environment - System Specification", Rome Air Development Center, Griffiss AFB NY, Tech Rep., January 1990.

[4] M. Jackson, System Development. New York, NY: Prentice-Hall International Series in Computer Science, 1983.

[5] E. Kean, et al., "Rapid Prototyping - A Methodological Approach for Prototyping User Interface Requirements", Rome Air Development Center, Griffiss AFB, NY, Tech. Rep. RADC-TR-89-234, October 1989.

[6] M. Konrad et al., "RADC System/Software Requirements Engineering Testbed R&D Program," Rome Air Development Center, Griffiss AFB, NY, Tech. Rep. RADC-TR-88-75, 1988.

[7] M. Konrad et al., "VHLL System Prototyping Tool - Users Manual," International Software Systems, Inc., Austin, TX, Tech. Rep., June 1987.

[8] J. Kramer et al., "TARA: Tool Assisted Requirements Analysis," Rome Air Development Center, Griffiss AFB, NY, Tech. Rep. RADC-TR-88-28, July 1988.

[9] G. P. Mullery, "CORE - A Method for Controlled Requirement Expression," Systems Designers Limited, Camberley UK, Tech. Rep., Feb. 1979.

[10] W. Rzepka, "A Requirements Engineering Testbed: Concept, Status and First Results" in Proceedings 22th Hawaii International Conference on System Sciences , Jan. 1986, pp. 339-347.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S CMU/SEI-91-TR-30	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-TR-91-30
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute 6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/AVS Hanscom Air Force Base, MA 01731
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office 8b. OFFICE SYMBOL (if applicable) ESD/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO 63756E; PROJECT NO. N/A; TASK NO N/A; WORK UNIT NO. N/A
11. TITLE (Include Security Classification) Requirements Engineering and Analysis	
12. PERSONAL AUTHOR(S) SEI Requirements Engineering Project	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM TO	14. DATE OF REPORT (Yr., Mo., Day) December 1991 15. PAGE COUNT 202
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Software requirements, requirements engineering, mission-critical systems

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Inadequate, incomplete, erroneous, and ambiguous system and software requirements are a major and ongoing source of problems in systems development. These problems manifest themselves in missed schedules, budget excesses, and systems that are to varying degrees unresponsive to the true needs of the sponsor. These difficulties are often attributed to the poorly defined and ill-understood processes used to elicit, specify, analyze, and validate requirements.

The Software Engineering Institute (SEI) hosted the Requirements Engineering and Analysis Workshop in Pittsburgh, Pennsylvania, on March 12-14, 1991. The intention of the workshop was to focus

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution
22a NAME OF RESPONSIBLE INDIVIDUAL John S. Herman, Capt, USAF	22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7631 22c. OFFICE SYMBOL ESD/AVS (SEI)

ABSTRACT —continued from page one, block 19

discussion on issues and activities that could help the Department of Defense (DoD) to deal more effectively with the requirements of mission-critical systems. The SEI workshop built upon work performed previously at the Requirements Engineering and Rapid Prototyping Workshop held by the U.S. Army Communications-Electronics Command (CECOM) Center for Software Engineering in Eatontown, New Jersey, on November 14-16, 1989.

The workshop participants were divided into four working groups: Requirements Engineering Process and Products, Requirements Volatility, Requirements Elicitation, and Requirements Engineering Techniques and Tools. A summary of the findings of each working group follows.