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

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 afternoon 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 workshop [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 process 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 every 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).

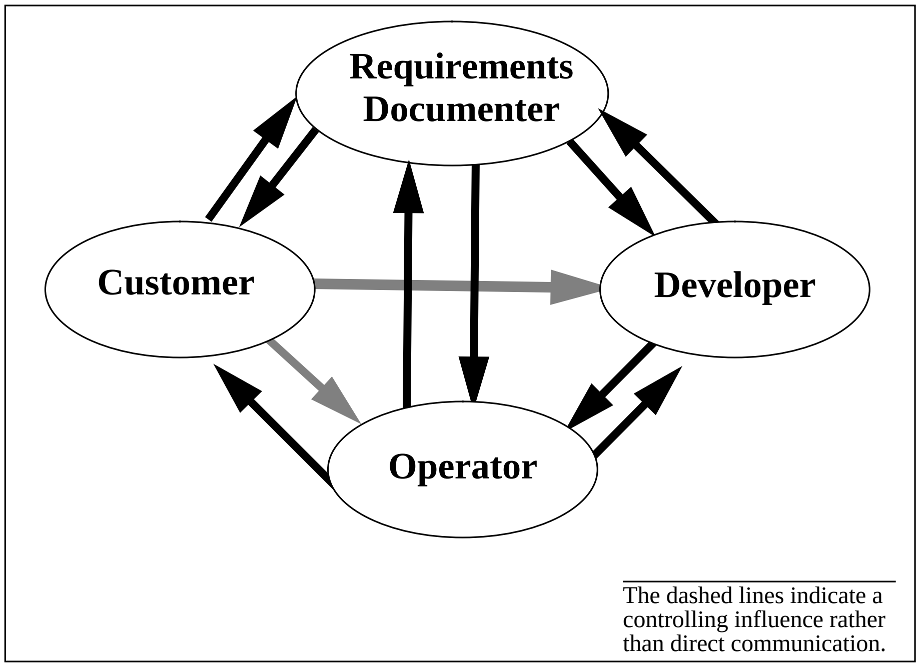
Requirements Documenter
Customer
Developer
Operator
The dashed lines indicate a controlling influence rather than direct communication.

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.

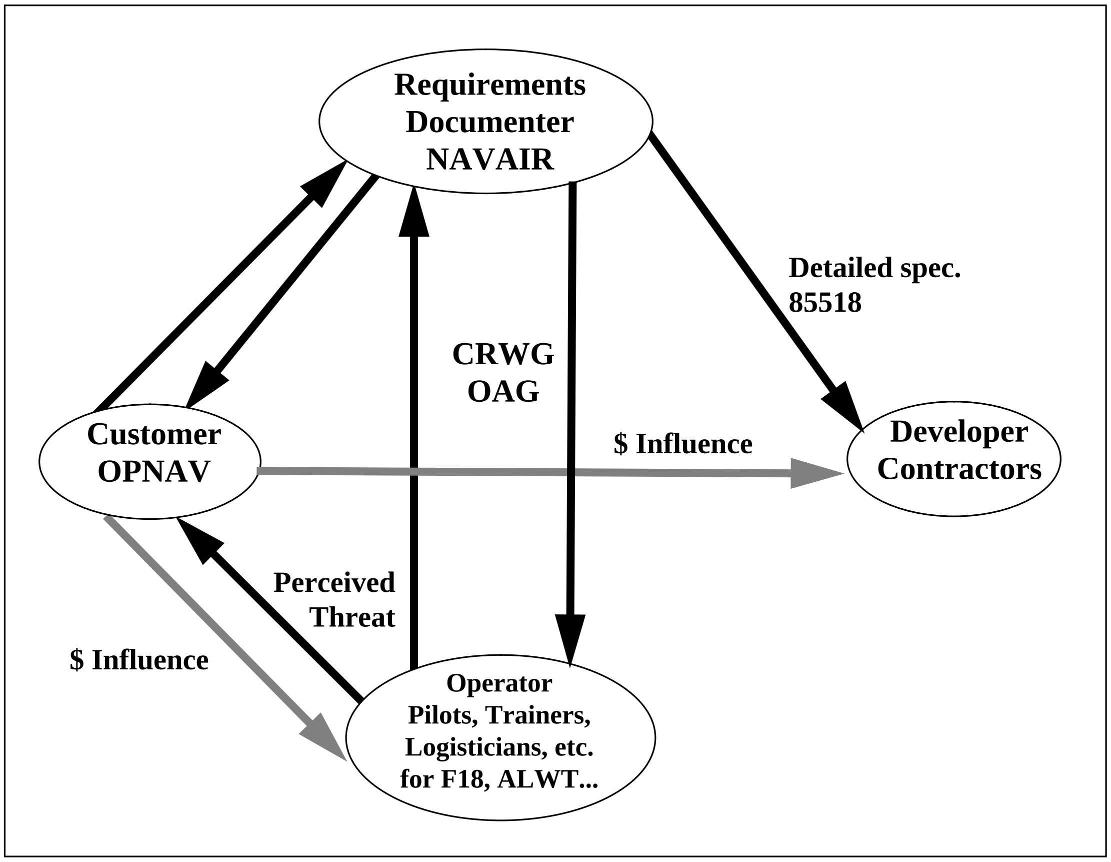

Figure 5-1 AYK-14 SAC Example of Requirements Elicitation

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 presented 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 requirements 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 requirements 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 positions 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. S**imple 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. B**usiness 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. T**he 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. A**lthough 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*

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

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	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Requirements Engineering and Analysis

12. PERSONAL AUTHOR(S)
SEI Requirements Engineering Project

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM TO	December 1991	202

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
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	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
John S. Herman, Capt, USAF	(412) 268-7631	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.