②

NAVSWC TR 91-584

A METHODOLOGY FOR SYSTEMS REQUIREMENTS SPECIFICATION AND TRACEABILITY FOR LARGE REAL-TIME COMPLEX SYSTEMS

BY MICHAEL EDWARDS AND STEVEN L. HOWELL

UNDERWATER SYSTEMS DEPARTMENT

27 SEPTEMBER 1991

Approved for public release; distribution is unlimited.

NAVAL SURFACE WARFARE CENTER

Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

418546 6208

NAVSWC TR 91-584

A METHODOLOGY FOR SYSTEMS REQUIREMENTS SPECIFICATION AND TRACEABILITY FOR LARGE REAL-TIME COMPLEX SYSTEMS

BY MICHAEL EDWARDS AND STEVEN L. HOWELL
UNDERWATER SYSTEMS DEPARTMENT

Accesion For	
NTIS CRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

27 SEPTEMBER 1991

NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

FOREWORD

This document describes the beginnings of a methodology for requirements specification and traceability of real-time, large-scale, complex computer-intensive systems. The method is aimed at better understanding the top-level system requirements and how they relate to the system under design.

The ability to formally capture and understand the top-level requirements, before beginning the analysis and design of the system, is critical in creating systems which correctly reflect the user's needs and the avoidance of costly redesign. Being able to formally and efficiently relate the requirements to the system design and implementation, through traceability techniques, is essential to guarantee that the design completely meets the requirements. Traceability also is needed to properly maintain a complex system.

The authors would like to thank their sponsor, the Office of Naval Technology, especially Cmdr. Jane Van Fossen, USN Ret., and Elizabeth Wald. The authors would also like to thank Phil Hwang and all who provided technical support to refine this report, and Adrien Meskin for her editorial support.

Approved by:
C. A. Kalivretenos
C. A. KALIVRETENOS, Deputy Head
Underwater Systems Department

ABSTRACT

This document describes the beginnings of a methodology for requirements specification and traceability of real-time, large-scale, complex computer-intensive systems. The method is aimed at better understanding the top-level system requirements and how they relate to the system under design.
The methodology will cover the requirements aspects of system development over the entire system development life cycle, beginning with the specification of the requirements and tracing those requirements to the design and final implementation.

CONTENTS

CONTENTS (Cont.)

ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

The purpose of this document is to provide a detailed explanation of a Requirements Specification and Traceability (RESPECT) methodology which is being developed under the Engineering of Complex Systems (ECS) Block program. The methodology will cover the requirements aspects of system development over the entire system development life cycle, beginning with the specification of the requirements and tracing those requirements to the design and final implementation.

1.1 BACKGROUND

The results presented in this paper are part of the Office of Naval Technology (ONT) ECS effort. This program was developed to integrate systems engineering capabilities for developing large-scale, real-time, complex, computer intensive systems. The goal of the ECS effort is to improve the way in which the Navy currently creates, maintains and improves its systems by incorporating state-of-the-art technology and supplying new technology where holes in present methods exist. The ECS block is divided into four projects: Systems Design Synthesis Technology, Systems Evaluation and Assessment Technology, Systems Reengineering Technology, and Engineering Application Prototype. These projects work closely together to incorporate new technology across the entire system development life cycle.

The Requirements Specification and Traceability Task is within the Systems Design Synthesis Technology Project. The project looks at the forward engineering aspects of the system life cycle. The task will work closely with the Systems Design Capture and Analysis Task (within the same project) to ensure that the requirements provide all the information for the systems engineers to fully capture the design.

Typically, the requirements for real-time, complex Navy systems are developed by technical and operational Navy experts. These experts need to specify the requirements in a manner in which they are comfortable. Usually, this means a combination of English text and diagrams. Formal specification methods which are highly mathematical are less intuitive than English text; therefore, it may be more difficult for the experts to validate that the formal requirements represent their intentions.

The requirements for large-scale, real-time, complex, computer-intensive systems may be defined in thousands of pages of documents, usually specified in a rather informal manner by several different experts. These requirements must completely, consistently, unambiguously and correctly achieve the goals that the requirements developers have in mind. A representation needs to be created to specify these requirements in a manner that can be easily checked to ensure the presence of these characteristics, even as it goes through modifications and changes.

Currently, there are formal specification languages that allow for some checking of requirements. These languages tend to specify only certain pieces of a complex, mission critical system (i.e., hardware requirements, software requirements, etc.). These languages also tend to be limited to specifying only some aspects of a complex system (i.e., data requirements, behavior requirements, functional requirements, etc.). Existing requirements specification checking techniques are usually restricted to simple consistency checks and do not fully support completeness and correctness checking.

In addition to checking the requirements, a method needs to be developed to ensure that the design meets these specified requirements. As part of the system development and maintenance process, many decisions and trade-offs must be made in the design of the system. In large systems, even the requirements themselves go through many changes during the development phase. It is necessary that with all these changes, the design still meets the specified (latest) requirements. To guarantee this, requirements traceability throughout the systems engineering process is imperative; if the development process cannot be traced back to the requirements, errors will occur. Similarly, all system components throughout each level of the development process must be able to be linked back to the requirements. These links must be bidirectional to allow requirements tracing forward, from requirements to system components, and backward, from system components to requirements. Traceability must be maintained through all levels of the systems engineering process, from the problem as stated (or contracted) by the customer, through analysis, design, coding and testing to the final product. The theoretical background that allows the representation and linking of system requirements, at the highest level, through progressive phases of system hardware, software and humanware development needs to be formally developed. These techniques will provide the basis for an automated system for requirements traceability.

1.2 ROADMAP

The remainder of this paper describes the RESPECT methodology along with justifications. Chapter 2 describes the motivation for the approach that the research effort is taking. Chapter 3 describes the details of the RESPECT methodology. Chapter 4 looks at the feasibility of the approach based on existing technology. Finally, Chapter 5 looks at future plans for this research effort.

CHAPTER 2

MOTIVATION

The RESPECT methodology has evolved from a preliminary examination of how the Navy currently specifies and traces requirements and the current state-of-the-art technology. The motivation for this effort is broken into three sections: specifying correct and consistent system requirements, tracing the requirements through the design, and examining the shortfalls in the state-of-the-art technology.

2.1 SPECIFYING CORRECT AND CONSISTENT SYSTEM REQUIREMENTS

Building correct and long lasting large-scale, real-time, complex, computer-intensive systems starts with specifying correct and consistent system requirements. If the requirements do not correctly reflect the intentions of the domain experts, then the system will not be built according to their needs. Important decisions which eliminate inconsistencies and clarify ambiguities in the requirements are often made by the system designers when they should be made by the domain experts. For most Navy systems the requirements are developed by the Navy and passed on to contractors for analysis and system design. All the information concerning what the system should do needs to be specified at this time. In order to ensure that the system will function as required by the Navy, these requirements must be specified completely, unambiguously and correctly, so that the contractors can develop the system without having to interpret the requirements.

Currently, the Navy specifies their requirements to contractors in documents. These documents are usually thousands of pages in length and are developed by several different domain experts. They contain all the information regarding what the system should do and any implementation constraints that fit the Navy's specific needs. The functionality of the system is fully described. All critical behavior and timing of the system is specified. Physical properties such as size and weight of the system are also included. The Navy also specifies certain constraints that limit the freedom that the contractor has on designing the system. These constraints include the use of specific hardware, software languages, operating systems and windowing environments. Sometimes it is necessary to constrain the analysis or design of parts of the system. For example, some systems might require the use of a specific functional breakdown for a part of the system.

The present method of specifying Navy requirements is very informal. It consists mostly of English narrative with some diagrams and supporting charts. There is no systematic method of checking for consistency of information, or for identifying ambiguities, leaving their resolution to the contractors.

These inconsistencies and ambiguities are inevitable since the requirements are developed by more then one expert using an imprecise language (English). A formal method would allow the requirements developers to resolve these issues by checking the requirements for consistency and ambiguities before the contractors performed any detailed requirements analysis or design.

2.2 MAINTAINING CONSISTENCY BETWEEN REQUIREMENTS AND DESIGN

Maintaining consistency between the requirements and the design is one of the key issues in developing a correct and long lasting system. This includes both determining if the design and implementation initially meets the requirements, and maintaining requirements, design, and implementation consistency throughout the system development life cycle. Since the Navy typically relies on contractors to design and build large, complex real-time computer intensive systems, having a systematic way of validating that every requirement is met by the design is important, not only to ensure that the system performs correctly, but also to determine whether contractual obligations have been met.

Post deployment support is essential to the development life cycle of Navy systems. Systems introduced into the Navy's fleet tend to remain for long periods of time due to the amount of cost and time it takes to develop new systems. This is especially true during the current environment of declining defense dollars. Fixing problems and developing enhancements to existing systems are critical in support of the fleet.

Any change to a complex system can have effects which may not be obvious to the maintainer of the system. An example of this happened July 1991 when the C & P telephone company had a large part of their switching network shut down by a bug in their computer software. According to the Washington Post, the bug was caused by a "minor" software change made by DSC Communications Corporation. A Vice President of DSC stated that "The software was sent out without major testing because DSC judged that the changes were too small to require it."[1] These "small" changes shut down telephone services in three states for over a six-hour period. Although the shutting down of the phone company had a major effect across the country, it does not compare with the disastrous consequence that unforeseen effects of a "small" change can have in a sonar or weapons system especially during wartime.

These types of "bugs" might be avoided with proper traceability and system maintenance techniques. These techniques show a relationship between each element of the design and the requirements. Any change to the design can be analyzed to determine if the system still meets every requirement. Since every requirement that is affected by a part of the design is linked through the traceability techniques, any effects on requirements that the system maintainer may not have considered will be examined. This cuts down on the possibility of requirements that were initially met by a part of the design not being met due to a change in a piece of the design.

2.3 SHORTFALLS IN CURRENT TECHNIQUES

In order to implement a methodology such as RESPECT, the technology must be mature enough to support it. The current state-of-the-art for both requirements specification and traceability techniques are not robust enough to fully meet all the Navy needs. This section describes some of the shortfalls in today's technology, which is one of the reasons for continuing to research this area.

2.3.1 Shortfalls in Current Requirements Specification Techniques

Some of the current problems with requirements specification languages were listed in the Requirements Engineering and Rapid Prototyping Workshop Proceedings, which was sponsored by the US Army CECOM Center for Software Engineering. According to the findings of this workshop, current requirements specification languages lack the following:

* They do not capture requirements information effectively to support the system evolution;

* They do not specify nonfunctional requirements;

* There is no automated way to reflect changes to the systems in the requirements or vice versa;

* They do not represent diverse viewpoints;

* There are too many gaps in the formalisms that represent system requirements;

* There are too many facts that have to be known before any requirements specification languages can be used.[2]

Current requirements specification and traceability tools (i.e., Teamwork/RQT, R-trace, etc.) allow for a manual parsing and grouping of functional requirements. There is no systematic method, either syntactic or semantic, in which requirements can be grouped. There is also no automated checking of the requirements for consistency or ambiguity. Requirements which are grouped together can be manually labeled as ambiguous or inconsistent, but these designations are based strictly on the individual's analysis of the requirements and are not systematic or repeatable.

2.3.2 Shortfalls in Current Requirements Traceability Techniques

The current commercial state-of-the-art requirements traceability techniques (i.e., Teamwork/RQT, R-trace, RDD-100) simply link requirements to pieces of the design and implementation. They relate parts of requirements documents to a database which represents the pieces of system design and implementation. These techniques do not capture how the requirement is satisfied by the design, just the fact that some relationship exists.

Another shortfall with today's traceability tools is that they lack the ability to trace back from the actual pieces of design and implementation to the requirements. Although some tools such as Teamwork/RQT and R-trace allow the user to trace from requirements analysis tools such as Teamwork and Software Through Pictures, they do not have a method for tracing from a particular piece of hardware or humanware back to the requirements. This capability would be extremely useful in performing systems maintenance.

CHAPTER 3

APPROACH

The RESPECT methodology approach is divided into three parts. The first part looks at natural interfaces for requirements specification. The second part develops a formal requirements representation. The last part looks at techniques for maintaining consistency between requirements and design. This chapter gives an overview of the RESPECT methodology and then gives a detailed explanation of all three parts.

3.1 OVERVIEW OF METHODOLOGY

The RESPECT methodology examines the requirements specification and traceability challenge in its entirety. It examines the problem in the abstract and does not limit itself to today's technology. The approach looks towards the long-term and incorporates ideas which makes use of tomorrow's technology as well as today's.

It is often said that one person's design is another person's requirement and vice versa. It is therefore necessary to define these terms in the context of the RESPECT method before further explaining the approach. The term "requirements" refers to the top-level system requirements that are specified by the operational personnel at the initial stage of a system's development. The term "design" is used to represent every stage in the system development life cycle after the specification of requirements. This includes what is traditionally known as requirements analysis, design, detailed design, testing and maintenance. The term "domain expert" refers to a person who originates requirements and is an expert in a particular application area; it is used interchangeably with the term "requirements originator."

The methodology divides the requirements aspect of system development into three parts. The first part is the conceptual view which represents the intentions of the domain experts who did the initial development. The second part is the formal representation which ensures that the requirements are captured both completely and consistently. The final part is the design elements to which the requirements are traced. The method uses the term "design element" to denote any part of the system design or implementation (i.e., data flow diagrams, code, pieces of hardware, humanware, structure charts etc.).

It is important that the requirements and design remain consistent throughout the entire system development process. The conceptual view of the requirements must be completely and correctly presented to the designers through the formal representation and interfaces. The designers must then be able to demonstrate that the final design meets the intentions of the requirements originators through traceability and interfaces. This methodology allows for a formal representation of the conceptual view of the requirements in a manner which ensures the integrity of the requirements and presents the requirements clearly to both the requirements originators and the designers.

Figure 3-1 gives a pictorial view of the approach taken by this research effort showing all three parts and how they are related. The cloud figure represents the conceptual requirements which the requirements originators have in their minds. These ideas are captured through interfaces, which might be natural language or graphical, into a formal representation. The requirements are then checked for consistency, ambiguity and completeness through the formal representation. The requirements originators, through the use of the interfaces, check the formal structure for correctness. Finally the requirements are traced from the formal representation to the system design elements.

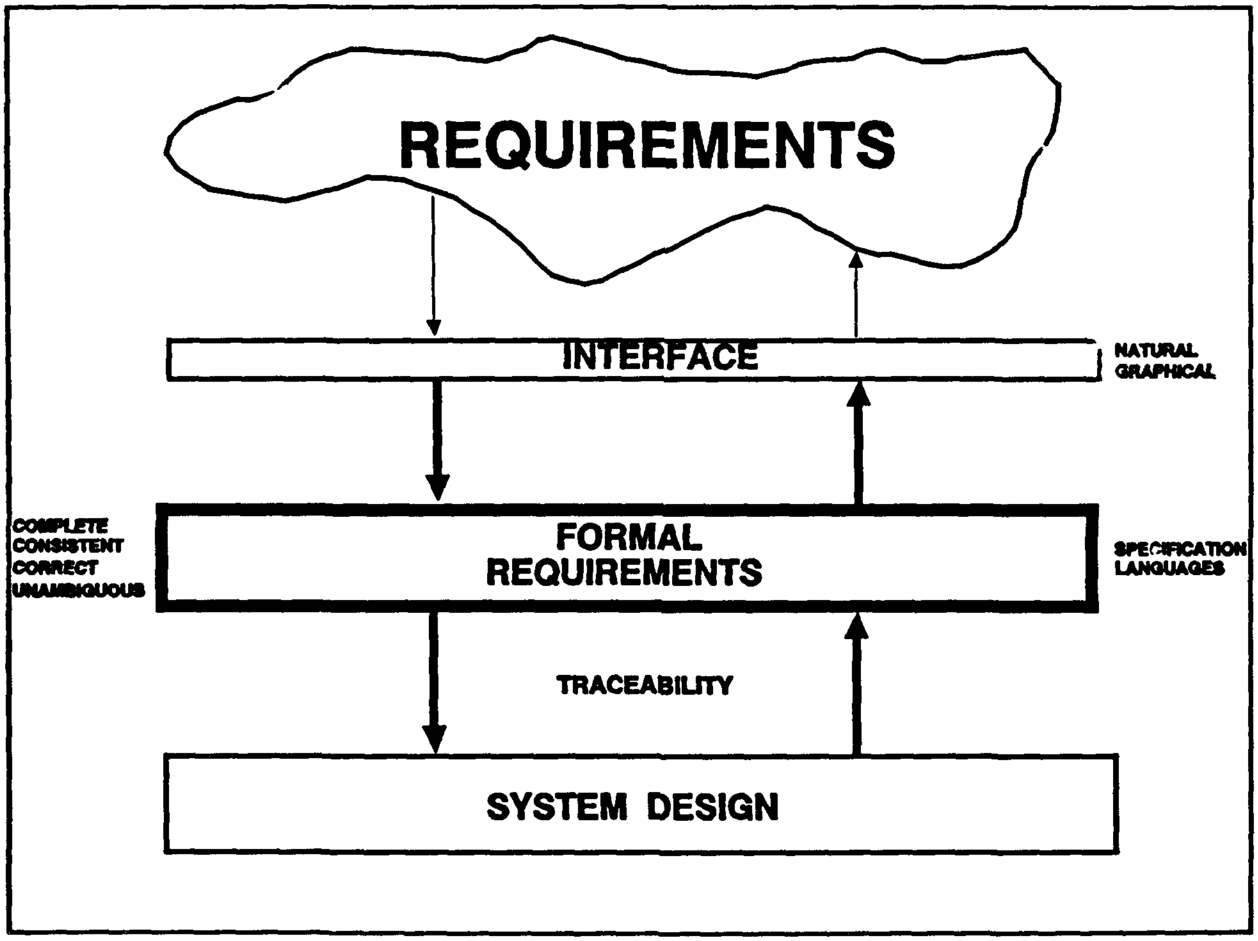

FIGURE 3-1. OVERALL VIEW OF THE METHOD

The methodology will specify the requirements for building interfaces which ensure that the formal requirements meet the original intentions of the domain experts. This will be done by presenting the requirements to the domain experts in a way which is complete, unambiguous and natural to them. In this way, the requirements originators can check the correctness of the requirements after they have been formally captured without having to understand the mathematical formalisms which comprise the formal representation.

Within RESPECT, the methodology will create a syntax for a formal requirements representation. This representation will allow for an assurance of the requirements integrity without being biased toward a particular view or perspective of the system. The representation will be mathematical and logical in nature and will require different interfaces to allow the requirements originators and the designers to view them properly.

Also, systematic techniques for ensuring consistency between the requirements representation and the design will be created. This will be done through interfaces, traceability techniques, and by automating the process of transforming the requirements representation to present design methods. These techniques will confirm that the final design and implementation meet the requirements as presented in the requirements representation.

3.2 NATURAL EXTRACTION OF REQUIREMENTS INFORMATION

One major step in the requirements process is presenting the formal requirements to the domain experts using natural techniques, i.e., English and graphical. This is important to prevent errors during the initial specification of the requirements as well as for the process of reviewing the requirements for correctness.

Typically, several different experts with widely varying backgrounds are involved in developing requirements for large complex systems. These experts are only responsible for specifying certain areas of the system and consequently are only interested in viewing information which relates directly, or indirectly, to those areas of the system. It is therefore important that each domain expert be able to view the full set of requirements through their specific perspective.

3.2.1 Different Domain Perspectives

Figure 3-2 shows how the different system domains fit into the RESPECT methodology. The requirements are broken up to represent the different domain experts who develop requirements. Each expert has his own perspective of the system. The perspectives are integrated by the interface to form the formal representation. The interfaces also display the requirements according to each expert's perspective.

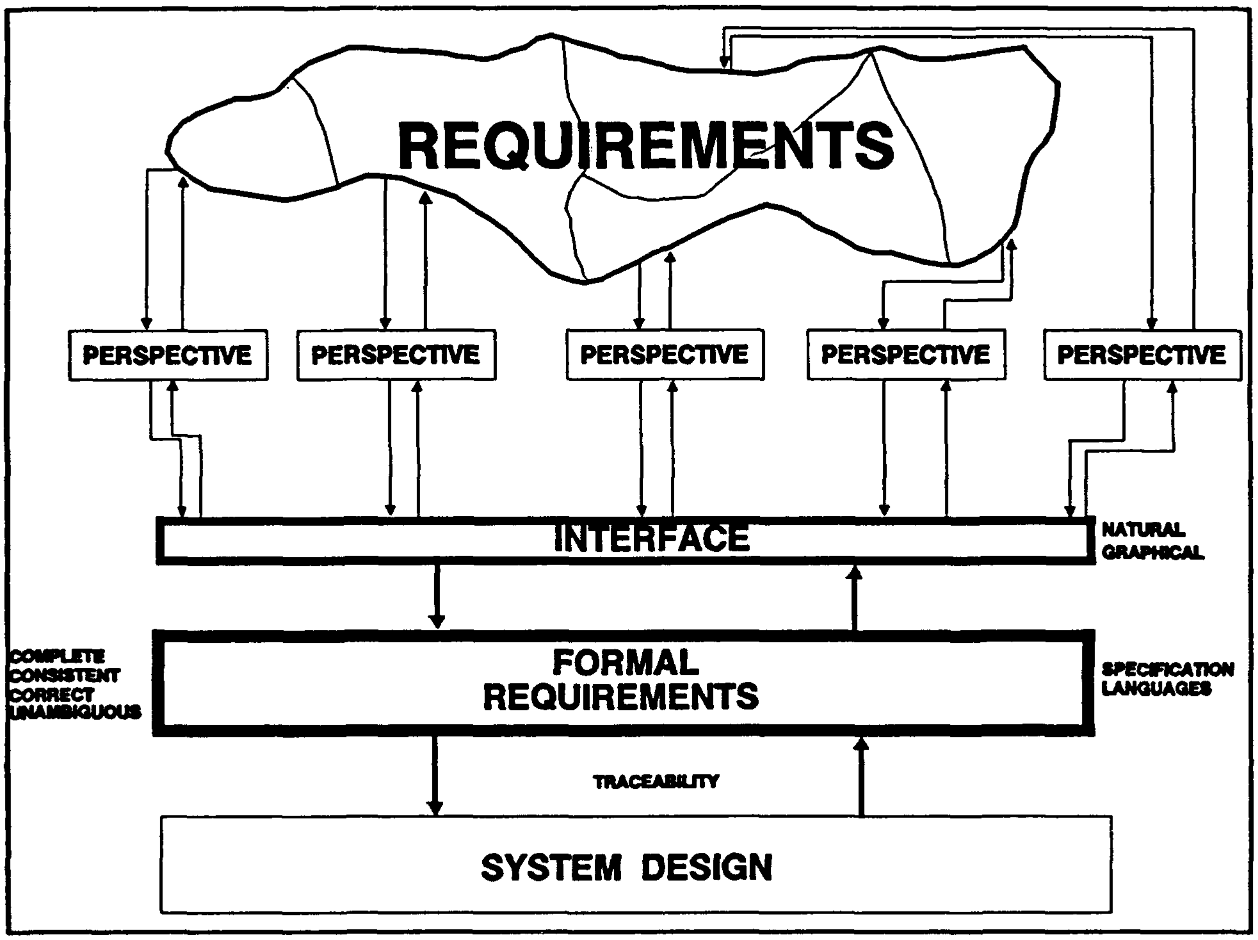

FIGURE 3-2. PERSPECTIVES

Each domain perspective contains information that overlaps the different domains. A domain expert will probably need to view all the requirements information relating to his particular domain, but only some of the information in the other domains. It is important that the domain experts can limit the information viewed. For example, an expert creating the requirements for a hydrophone or receiver of an active sonar system needs to know all the information of the hydrophone including its frequency, weight and environmental constraints. The hydrophone expert also probably needs to know the frequency of the projector or transmitter, but might not need to know the weight constraints or environmental conditions of the projector. He needs to be able to view the projector's requirements which are pertinent to the hydrophone without being distracted by all the other requirements of the projector. This is extremely important in large systems, because the number of requirements are numerous and their relationships are complex.

3.2.2 Natural Interfaces

In order to allow the domain experts to specify and review the formally represented requirements, natural interfaces need to be built. There are two natural methods of specification: graphical interfaces and English interfaces. A combination of the two approaches is likely to be the most effective

3.2.2.1 Graphical Interfaces. Graphical interfaces represent the requirements in a form other than text. This is useful for presenting the requirements in a manner which is natural to the eye. They are also useful for displaying states, control, relationships and hierarchies.

One way to adapt to graphical interfaces is through the use of an expert system shell to create an interactive environment. The expert system shell acts as a guide, asking the requirements generators leading questions to help set up their environment. This is extremely useful when an ambiguous textual description of the requirements already exists and the transition needs to be made to a more graphical form.

3.2.2.2 English Interfaces. The English language is extremely imprecise. This makes creating an automatable process for converting English text into a formal structure very difficult. Parsing text is the first step in converting from English text to a more formal language. Parsing is the partitioning of the requirements document into parts in order to better understand the requirements. Parsing can be done manually or systematically according to syntax or semantics.

3.2.2.2.1 MANUAL PARSING. Manual parsing requires the knowledge of domain experts or system designers to partition the requirements. These experts look through the requirements and extract the necessary information to create the formal requirements. Manual parsing is neither systematic nor repeatable; but given the imprecise nature of the English language, some manual parsing will probably be necessary when converting from English to the formal representation.

3.2.2.2.2 SYSTEMATIC PARSING (AUTOMATABLE). A systematic method of parsing a requirements document is preferable to a manual one. A systematic method divides the requirements according to rules. These rules can either be generic to all requirements or domain specific to the system under design. A systematic method for parsing requirements is both repeatable and automatable. There are two types of systematic parsing: syntactic and semantic.

Syntactic parsing is breaking the requirements down according to specific words and structures. This is a first step in organizing the requirements in a formal manner. For example, a syntactic parser can extract all the statements in a document following the word "shall." This would allow for easy identification of statements of what the system must do. A parser could also break the paragraphs into individual sentences. Then each sentence could be identified as a different requirement. Syntactic parsing can also include grouping all the sentences which contain a specific keyword (i.e., threat, hydrophone, projector, etc.).

Semantic parsing is breaking the requirements down according to specific meaning. This process is difficult to do with the informality of the English language. Semantic parsing techniques may be developed for specific domains because the same words and phrases are often used. Some examples of semantic parsing are grouping phrases or sentences which contain related keywords (i.e., targets, submarines and ships) and grouping paragraphs or sections that are related either within or across documents.

3.3 FORMAL REPRESENTATION

The formal requirements representation will be used to fully capture the requirements. It will act as a buffer between the people who generate the requirements and the designers of the systems. The representation will allow for the requirements to be verified before any partitioning of the requirements or designing of the system takes place. The representation will consist of formal specification languages which will be used to capture and analyze the requirements. These languages check the requirements for consistency and can prove certain assertions about the system. The representation will also be open and flexible enough to allow for interfaces and traceability techniques to be built around the representation.

3.3.1 Using Several Specification Languages

The Systems Design Capture and Analysis Task determined that to fully capture the design of a system, several views of the system need to be examined.[3,4] The requirements for a system must contain all the information necessary to capture all the views during system design. Also, the types of requirements information is diverse. The requirements contain hardware, humanware and software constraints, functional requirements, nonfunctional requirements, behavior, and real-time attributes as well as testing and documentation procedures. The formal representation must fully capture all the requirements of the system. This may require the use of more than one specification language in the requirements representation. Present day specification techniques seem to capture part of a system very well. Some languages, such as Modechart,[5] capture behavior while other techniques, such as entity relationship diagrams, capture the required objects of the system and how they relate. Combining a number of these languages and developing a method to check consistency across these languages will allow full capture of all the requirements of a system.

3.3.2 Criteria of Requirements Specification Techniques

In order to determine the best notation for fully capturing the formal requirements, an evaluation of state-of-the-art requirements specification languages is being performed. As a first step to evaluating the requirements specification languages, a number of criteria, against which the specification languages will be judged, has been created. These criteria include properties for formally capturing a complete system (i.e., scope), formal methods for checking a system (i.e., consistency, completeness, etc.), properties specific to specifying large-scale, complex systems (i.e., real-time, scalability, etc.), and properties concerning the implementation of the method (i.e., how well the language can interface with other specification languages, ease of use of interface, etc.). A detailed description is presented in Technical Memorandum 5530-100: Evaluation Criteria for Real-Time Specifications Languages, published by the Naval Research Laboratory (NRL) and included as Appendix A to this document.

3.4 CONSISTENCY BETWEEN FORMAL REQUIREMENTS AND THE DESIGN

Creating and maintaining a correct and consistent set of requirements does not guarantee a properly operational final system. The designed system must also meet the specified requirements. Maintaining consistency between requirements and design can be done in two ways: through traceability techniques and by direct transformation from the requirements representation to design.

3.4.1 Traceability

Traceability provides a relationship between the requirements, the design, and the final implementation of the system. Showing these relationships aids both the designers and testers in many areas. It allows the designers to demonstrate that their design meets the requirements and also allows for easy recognition of those requirements which have not yet been met by the design. Some of the implications of a requirements change during or after the development of the design can be determined before system redesign takes place. One of the greatest advantages of traceability is evident in the post deployment phase of the system life cycle. By effectively relating each requirement to a specific design element, the testers and designers can determine the effects of changing the design on the requirements.

Traceability can be implemented at two levels: on-line and through reports. On-line traceability allows a direct interactive means of referencing a design element back to the original requirement(s) (i.e., clicking on a bubble of a data flow diagram and being able to see the requirement(s) that the bubble satisfied, or tracing a particular sub-routine of code back to its initial requirement(s) or functional representation). This type of traceability is advantageous when trying to determine the effects due to a change in the design or a change in the requirement after the system is already developed.

Traceability through reports allows the designer to see which requirements are satisfied through a matrix of tables. These reports can give information based on a variety of keys; such as requirements, design elements, unsatisfied requirements, etc. This type of traceability is especially useful in determining the completeness of the design. Both on-line and report traceability are necessary to completely maintain a system.

3.4.1.1 Traceability Domains. In order for a traceability technique to be effective it must allow the designer to trace the requirements through each stage of the design. It is important that traceability is consistent at every level of the design and that there is direct traceability between the design levels as well as between the requirements and the design. This ensures that changes made in later design stages get reflected back to earlier stages in the system's development (i.e., changes made during detail design need to be reflected in the design and requirements analysis phases). Figure 3-3 demonstrates some of the different design stages and how the relationships between requirements, design and implementation need to be documented.

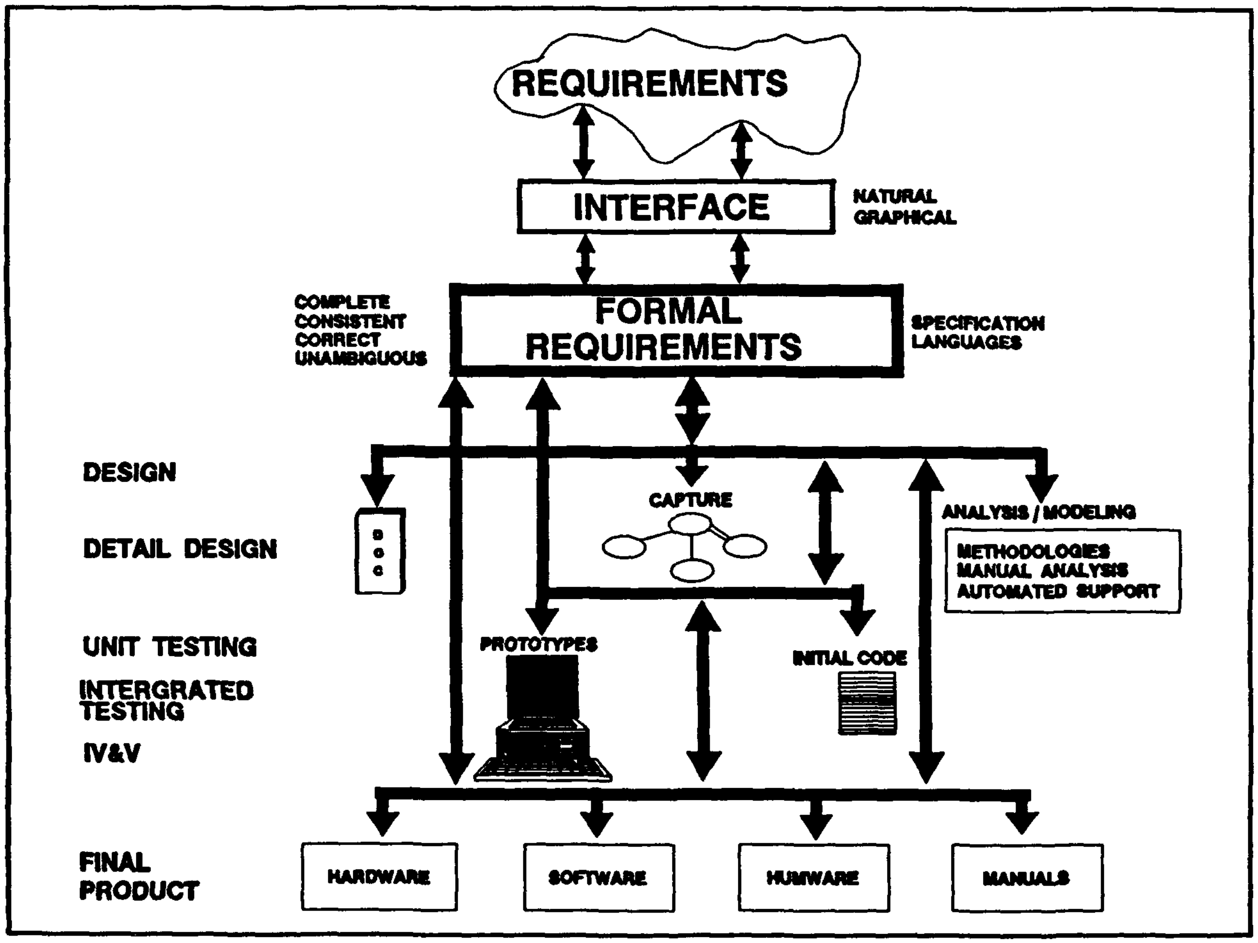

FIGURE 3-3. TRACEABILITY DOMAINS

The stages in system development range from the original English requirements through the internal representation, requirements analysis and top level design, down to detailed design and implementation. Each stage has its own set of different notations and methodology. These notations require different methods of relating their parts of the system back to the requirements and to the other stages of design. It is imperative that every design stage is linked to every other design stage so that the effects of a change to a particular stage is reflected throughout the entire design.

One of the most difficult issues involved in traceability concerns linking requirements to the implementation of the system. Since the implementation of systems involves hardware, software and humanware, more robust tracing methods may apply. Typically, tracing techniques lend themselves to design objects which are stored in a computer database (i.e., data flow diagrams, pieces of code, requirements documents, etc.) since it is relatively easy to attach comments and notes to any design element which is stored in a computer. Tracing pieces of hardware and humanware back to the requirements is more difficult because there is no technique for attaching the information directly to the design element. A computer representation of the design elements can be developed (i.e., Teamwork/RQT) but that does not allow direct traceability from the design element back to the requirement.

Within each stage in the system development life cycle, the system may be viewed in different ways. Traceability must be maintained within these separate views which are defined by the Design Capture and Analysis Task of the ECS block.[3] These views are Functional, Informational, Environmental, Behavioral and Implementation. Each view contains different information but some information overlaps from one view to another. Traceability is important between these views so that consistency is maintained. Each view of the system is captured using different methods so different linking techniques may need to be developed to trace to each view.

3.4.1.2 Linking Technology. Linking is one method of relating the requirements to the design elements. For large systems these links need to be automated through a computer database or Hypertext system.

Present methods allow some traceability by using simple linking techniques to relate requirements to design. These methods do not annotate the types of relationships that exist between the requirements and design. Complex linking techniques, which would show the specific relationships between the requirements and the design, will allow the designer to better understand the system and the effects that changes to the design will have with respect to the requirements.

Figure 3-4 shows the manner in which most traceability tools link requirements to design. The links are simple in that they do not reflect any meaning behind the relationships between requirements and design. In this case, the links show that requirement A is somehow satisfied by design elements one, two and three. Figure 3-5 shows one complex method of linking which uses combinatorial logic to display how the design elements satisfy the requirements. An example of this is shown in Figure 3-6. In this example, the requirement "target sub" is being satisfied by the design elements "passive sonar," "active sonar" and "display." The combinatorial logic method shows a specific relationship between "target sub" and the three design elements.

"Target sub" is being satisfied in the system by the "display" AND "active sonar" OR "display" AND "passive sonar." This representation describes the relationship between the three components and the requirement.

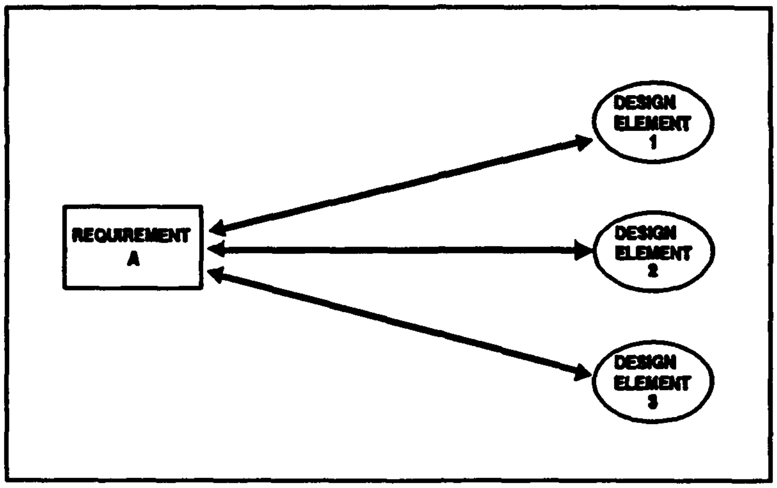

FIGURE 3-4. SIMPLE LINKING

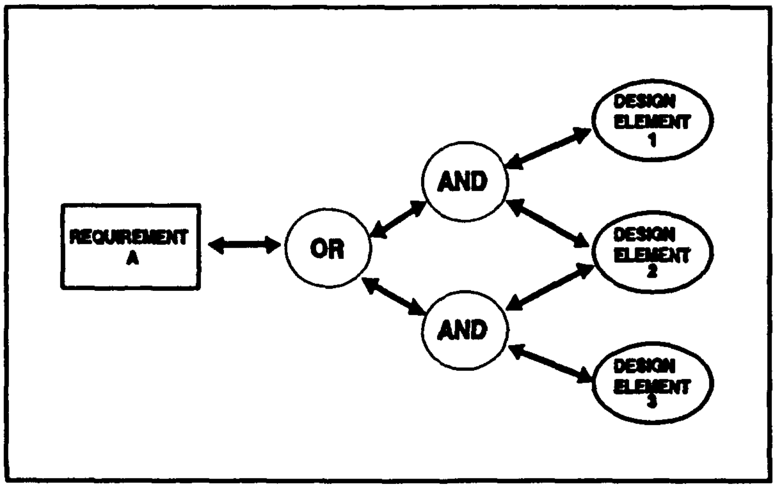

FIGURE 3-5. COMPLEX LINKING

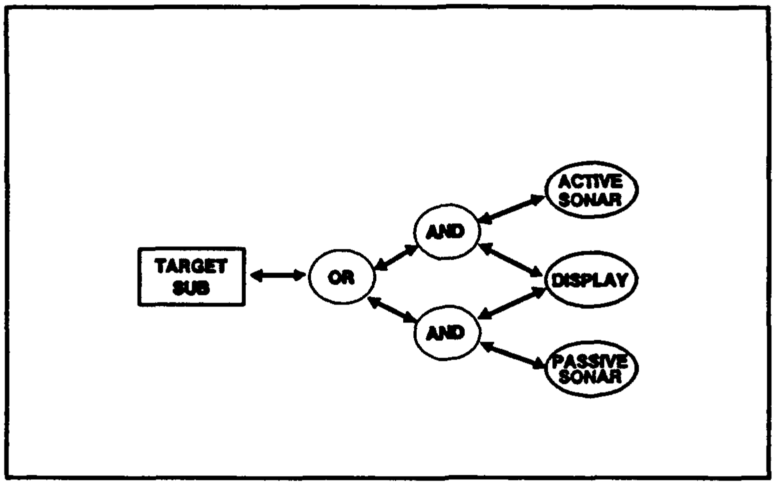

FIGURE 3-6. COMPLEX LINKING (EXAMPLE)

Another method of complex linking captures the conditions under which a design element satisfies a requirement. Using the above example, the "target sub" requirement can be satisfied by "active sonar" under the condition that the system is already detected by the targeted sub. If the system is not already detected, then the requirement is satisfied by the design elements "passive sonar" and "display."

Another method of specifying a relationship between requirements and design elements is by capturing design decisions. By documenting the assumptions, options and reasons why a system designer makes certain decisions, the manner in which a design element meets a requirement(s) is also captured. It is also important that these design decisions are captured formally so that a change in an assumption can produce an automated change in design.[6]

3.4.2 Automated Transition from Representation to Design

One method of ensuring consistency between requirements and design is by using automated techniques to transfer the information from the requirements representation to the parts of the design. If similar methods are used to capture the requirements information and the design information, then automated transitioning from one representation to another is possible. This transitioning would guarantee consistency because there is no change in the information. The problem with this method is that it needs to be back annotated. If a change is made to the design, a method must also exist to reflect those changes back to the requirements.

3.4.3 Marking Satisfied Requirements

One important consequence of traceability is the ability to determine if the requirements are satisfied by the design. This feature is especially essential for DOD work since the designing and constructing of the system is typically performed by contractors. The major issues concerned with marking requirements include showing that the requirements are completely satisfied by the design, denoting when a requirement is partially satisfied by the design, denoting under what conditions requirements are satisfied and determining if nonfunctional requirements have been met by the design.

3.4.3.1 Completely Satisfied Requirements. The RESPECT methodology will create a technique for consistently marking when a requirement is completely satisfied by the design. This marking system will be maintainable so that checking is done whenever design changes are made to ensure that the requirements are still met by the design. Initially, any requirement which is traced to a design element, and then changed, will be flagged, and the designer will be forced to check that all requirements relating to that changed design element are still met. Eventually a systematic technique will be developed, based on the formal capture of complex linking techniques, to selectively flag the requirements that are affected by the design change.

3.4.3.2 Partially Satisfied Requirements. Requirements of large, real-time, complex, computer-intensive systems tend to be met by several design elements. At a particular stage in a system's development, a requirement might only be partially met by the design. A method for marking these requirements as being partially satisfied will be developed. These markings will not only indicate that the requirement is partially satisfied, but give additional information as to how it is satisfied and what is needed to completely satisfy the requirement.

3.4.3.3 Conditions of Satisfied Requirements. In order to fully understand if the requirements are satisfied by the design, it is important to capture the conditions under which certain requirements are satisfied. Some designs will fulfill certain requirements, based on the fulfillment of other requirements, design decisions or external events. It is necessary to capture and monitor these conditions to ensure that the requirements are satisfied under all operating conditions.

3.4.3.4 Nonfunctional Requirements. Certain requirements are not linked to any particular combination of design elements but are affected by the system as a whole. These requirements tend to be nonfunctional requirements. Some examples of these are dependability, security, and timing. The methodology will create a method for determining if these requirements are satisfied by the system and a manner for marking these requirements as satisfied. Since these requirements are affected by any change to the design or implementation, keeping track of how they are satisfied is important to ensuring that these requirements are met.

CHAPTER 4

FEASIBILITY

Some of the ideas presented in the RESPECT methodology are feasible using today's technology. Other ideas will require the advancement of new technology before they can be successfully implemented. The methodology is addressing the requirements problem as a whole and is attempting to provide a complete solution. In building the methodology and associate prototypes the project does not want to rule out the possibilities that can be created by advancing technologies. The following describes the feasibility of each section of the RESPECT methodology. The emphasis is on describing the areas of research which are currently being explored, and the research efforts which will be explored in the future as technology matures. Also some major issues concerning the development and implementation of the RESPECT methodology are discussed.

4.1 FEASIBILITY OF NATURAL INTERFACES

The ideas presented in this paper referring to natural interfaces will require an advancement in the state-of-the-art technology for natural specification development. The research effort will work with the Natural Specification and Generation Task (scheduled to start in FY93) to help make advances in this area.

One area that is currently being researched is using an expert system shell to aid users in parsing English requirements and converting them to a graphical interface. This method is interactive with the user. The expert shell asks questions of the user to guide the user in forming the graphical representation. This method is being prototyped by Trident Systems, Inc.

4.2 FEASIBILITY OF FORMAL REPRESENTATION

The research area of formal specification is fairly well developed. There are several requirements specification languages and techniques available (i.e., Modechart, Van Schouwen (Modified SCR), ASTRAL, Hierarchical Multi-State (HMS) Machines, Statemate) and others which are still under development. The main problem with the specification languages is their limited scope. Most cannot handle all the information that needs to be specified by a system. The difficulty in combining these languages is that most of the languages are propriety and therefore their structure is not open. This makes the implementation of this representation difficult to develop.

4.3 FEASIBILITY OF TRACEABILITY TECHNIQUES

The traceability issue is one which is still being researched today. Presently there is no standardized methodology for vendors who produce CASE tools involving requirements traceability. There is some research going on in the areas of traceability and the capturing of design decisions.

The Naval Postgraduate School is currently looking at a conceptual model and prototype for formally capturing design decisions. This model allows the designer of the system to formally capture and analyze the arguments and assumptions which lead to design decisions. This procedure may be tailorable to capture requirements traceability across large, complex systems.

The area of requirements traceability has been researched by commercial vendors (e.g., Cadre's Teamwork/RQT and Ascent Logic's RDD-100) and there are some CASE Tools on the market with requirements traceability capabilities. These tools allow for a simple linking of requirements to other requirements and design elements. These tools hold some information about the links using keywords and attributes. The linking techniques are very informal in nature and do not allow for any automatic consistency or correctness checking of the requirements to design links.

The idea of complex linking is one of the most challenging to implement. One of the most complicated parts of this effort is determining exactly what type of formal information needs to be captured concerning how requirements link to the design elements. This involves capturing both the engineering knowledge concerning the design and what specific types of relationships exist between requirements and the design elements. Once this information is discovered, creating a nomenclature for capturing this combination of knowledge and relationships can be developed.

CHAPTER 5

FUTURE PLANS

The Requirements Specification and Traceability Task's plans include the continuation of research in both the specification and traceability areas. All the information compiled will be documented in two Technical Reports (A Methodology for System Specification and Traceability of Large Complex Real-Time Systems (updated) and Design View Traceability Techniques). The following paragraphs describe in some detail the future plans of this research effort.

A full evaluation of several requirements specification languages will be performed. Based on this evaluation, one or more specification languages will be chosen to be included in the formal structure for the RESPECT methodology. The beginnings of the notation for combining these specification languages and methods for consistency checking between these languages will also be developed.

A detailed investigation into the types of information that need to be captured by complex linking techniques (i.e., what information is important in linking requirements to design) will be performed. This will include defining the proper questions to ask the domain experts, who presently specify requirements, and system designers, followed by a report of their answers.

The research effort will look at the Naval Postgraduate School's conceptual model of capturing design decisions.[6] The emphasis will be on improving it to effectively capture large, real-time complex systems. This will include capturing design decisions through all design phases and consistency checking and validation.

The research effort will start to develop complex linking techniques based on the needs of the domain experts. These techniques will capture the information that the Naval Postgraduate School's work does not cover.

This research effort will look at how existing commercial traceability tools can fit into the RESPECT methodology. The tools that will be examined include, but are not limited to, Teamwork/Rqt, R-Trace and RDD-100.

A mock-up demonstration of an interactive entry environment for requirements specification will be developed. This environment will be used for demonstration purposes using entry into an Information Modelling (IM) method. The IM method is one of the main views used in design capture, and the method is also applicable to requirements capture.

REFERENCES

1. Burgess, John, "Tiny 'Bug' Caused Phone Blackouts," The Washington Post, 19 Jul 1991.

2. Requirements Engineering and Rapid Prototyping Workshop Proceedings, U.S. Army Communication-Electronics Command Center for Software Engineering, Eatontown, NJ, Nov 1989, p. 53.

3. Hoang, Ngocdung T., "The Essential Views of System Development," Proceedings of the 1991 Systems Design Synthesis Technology Workshop, Silver Spring, MD, Sep 1991, pp 3-9.

4. Karangelen, Nicholas E., "Multi-Domain Real-Time System Design, Capture and Analysis," Proceedings of the 1991 Systems Design Synthesis Technology Workshop, Silver Spring, MD, Sep 1991, pp 11-22.

5. Jahanian, Farnam, Raymond Lee and Aloysisu K. Mok, "Semantics of Modechart in Real Time Logic," Proceedings of the 21st Annual Hawaii International Conference on System Sciences, Vol. II, IEEE Comput. Soc. Press, Washington, DC, 1988, pp. 479-489.

6. Ramesh, Balasubramaniam, "Capturing and Reasoning with Process Knowledge in Large-Scale Systems Design and Maintenance," Proceedings of the 1991 Systems Design Synthesis Technology Workshop, Silver Spring, MD, Sep 1991, pp 97-106.

BIBLIOGRAPHY

Berzins, Valdis and Luqi, Software Engineering with Abstractions, Addison-Wesley Publishing Company Inc., Reading Massachusetts, 1991.

Burgess, John, "Tiny 'Bug' Caused Phone Blackouts," The Washington Post, 19 Jul 1991.

Davis, Alan M., Software Requirements Analysis and Specification, Prentice-Hall Inc., Englewood Cliffs, NJ, 1990.

Gabrielian, A., "HMS Machines: A Unified Framework for Specification, Verification, and Reasoning for Real-Time Systems," Proceedings of 3rd ONR Workshop on the Foundations of Real-Time Computing, Washington, DC, Oct 1990, pp. 359-372.

Hoang, Ngocdung T., "The Essential Views of System Development," Proceedings of the 1991 Systems Design Synthesis Technology Workshop, Silver Spring, MD, Sep 1991, pp 3-9.

Jaffe, Matthew S. and Nancy G. Leveson, "Completeness, Robustness, and Safety in Real-Time Software Requirements Specification," Proceedings, 11th International Conference on Software Engineering, Pittsburgh, PA, IEEE Comput. Soc. Press, Washington, DC, May 1989, pp. 302-311.

Jahanian, Farnam, Raymond Lee and Aloysisu K. Mok, "Semantics of Modechart in Real Time Logic," Proceedings of the 21st Annual Hawaii International Conference on System Sciences, Vol. II, IEEE Comput. Soc. Press, Washington, DC, 1988, pp. 479-489.

Karangelen, Nicholas E., "Multi-Domain Real-Time System Design, Capture and Analysis," Proceedings of the 1991 Systems Design Synthesis Technology Workshop, Silver Spring, MD, Sep 1991, pp 11-22.

Lee, Insup, Susan Davidson and Richard Gerber, "Communicating Shared Resources: A Paradigm for Integrating Real-Time Specification and Implementation," Proceedings of 3rd ONR WORKSHOP on the Foundations of Real-Time Computing, Washington, DC, Oct 1990, pp. 359-372.

Levine, A., "An Investigation of Requirements Specification Languages: Theory and Practice," Computer (USA) Vol. 15, No. 5, pp. 50-59.

BIBLIOGRAPHY (Cont.)

Ramamoorthy, C. V., A. Bhide, T. Yamaura and V. Garg, "Software Quality and Requirement Specification," Proceedings IEEE Computer Society 1986 International Conference on Computer Languages, Miami, FL, IEEE Comput. Soc. Press, Washington, DC, 1986, pp. 75-83.

Ramesh, Balasubramaniam, "Capturing and Reasoning with Process Knowledge in Large-Scale Systems Design and Maintenance," Proceedings of the 1991 Systems Design Synthesis Technology Workshop, Silver Spring, MD, Sep 1991, pp 97-106.

Requirements Engineering and Rapid Prototyping Workshop Proceedings, U.S. Army Communication-Electronics Command Center for Software Engineering, Eatontown, NJ, Nov 1989, p. 53.

Roman, Gruia-Catalin, "A Taxonomy of Current Issues in Requirements Engineering," COMPUTER, IEEE Computer Society, Apr 1985, pp. 14-23.

Saeki, Motoshi, Hisayuki Horai and Hajime Enomoto, "Software Development Process from Natural Language Specification," Proceedings, 11th International Conference on Software Engineering, Pittsburgh, PA, IEEE Comput. Soc. Press, Washington, DC, May 1989, pp 64-73.

APPENDIX A

EVALUATION CRITERIA FOR REAL-TIME SPECIFICATIONS LANGUAGES

Evaluation Criteria for Real-Time Specification Languages

PAUL C. CLEMENTS

Human-Computer Interaction Laboratory
Information Technology Division
and
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

CAROLYN E. GASARCH

Human-Computer Interaction Laboratory
Information Technology Division

RALPH D. JEFFORDS

Locus, Inc.
Alexandria, VA 22303

CONTENTS

EVALUATION CRITERIA FOR REAL-TIME SPECIFICATION LANGUAGES

1. Introduction

This report proposes a set of evaluation criteria for languages designed to specify the requirements of real-time systems. It is intended for a reader who is beginning a real-time development project and considering a method or language for capturing the system's requirements. We assume the reader is familiar with at least some real-time specification languages and with the characteristics that distinguish real-time systems from others. Specification languages, for the purpose of this study, include both highly formal languages (having at least a formal syntax) as well as informal ones. We include technical criteria we believe may be formally evaluated, such as the ability to verify timing properties; we also include criteria of a more subjective nature, such as readability and ease of use. For each we include a list of key questions that a developer may use to help evaluate a candidate language.

Not surprisingly, the criteria are not unrelated, although how they affect each other varies from case to case. For example, increasing the formality of a language may increase or decrease the readability of the specification. A language with a strong conceptual construct and high applicability to real-time systems probably produces a very concise specification, which may be easier to modify, but an overly concise specification (or an overly verbose one) may have very poor readability.

We do not provide value rankings for the criteria, because the value varies with projects. For example, ease of learning would be more important to a project staffed with unskilled or inexperienced personnel than one with seasoned veterans; sophisticated support tools may be irrelevant to a project without the computing resources to exploit them.

We do not at this time provide objective measurement procedures for the criteria. For some criteria, derivation of measurements represents an obvious continuation of this work and is beyond the scope of the current effort. For others, it isn't clear that finding an objective measure is feasible. In any case, we believe that the identification of the criteria is useful in its own right. It should motivate the project manager to think about long-term issues and provide a justification framework for choosing a particular language and rejecting others.

The evaluation criteria are presented in two sections. Section 2 suggests language features that support desirable properties of the finished *product*—the requirements specification. Section 3 proposes language features that facilitate the *process* by which the specification is produced. A brief summary appears in Section 4. A bibliography and glossary relevant to real-time system specification conclude this report.

2. Product-Oriented Criteria

The product under consideration is the requirements specification. The purpose of a requirements specification language is to establish a syntactic and semantic context in which to develop that product. The purpose of a requirements specification is to define all acceptable implementations of a system and to specify any constraints on its implementation [Heitmeyer and McLean 1983]. We take as axiomatic that a requirements specification should be unambiguous, complete, verifiable, consistent, easy to change, traceable, and usable during development, operation, and maintenance [ANSI/IEEE Std 830-1984]. A specification language must at least permit such properties in a requirements specification. It might guarantee them (by making it impossible to produce a product without them); it might simply encourage them (by providing features designed to ease their rendering). Note that a language that guarantees a good product need not be the best choice; it might, for instance, be prohibitively hard to use.

In the following subsections, we suggest evaluation criteria related to the production of high-quality requirements specifications in the context of real-time systems.

2.1. Applicability to Real-Time Systems

Since real-time systems, by definition, must respond to events under some timing constraints, it is imperative that the language for rendering real-time specifications be able to express such timing requirements. The existence of a model for timing in the requirements specification language, and the notation for expressing timing constraints is the primary issue that sets real-time and non-real-time specification languages apart. The timing model may be based upon either continuous or discrete time. Furthermore, soft real-time systems deal with stochastic performance models; their requirements are written in terms of a some minimum number of times that a real-time deadline must be met. By contrast, hard real-time systems use deterministic models; their minimum required rate for satisfying a deadline is 100%. Some systems, such as the Space Shuttle, combine aspects of both soft and hard real-time: a set of primary or high-priority tasks must always meet their deadlines, whereas it is permissible for less important tasks to fail to complete from time to time.

It is not sufficient that requirements for real-time systems express only an ordering of events, system responses, etc.; they must also express absolute and relative time intervals from a fixed starting point. Timing constraints should be stated only in terms of events that are externally visible at the system level. To achieve this goal the model of the system environment embraced by the language must be complete and well-defined.

Some specification languages suitable for real-time systems may have semantics for parallelism, which some real-time applications may need. The semantics of parallelism in the specification language may use either a maximal parallelism model or an interleaving model. Maximal parallelism allows any number of events to occur simultaneously, as in the real world. The interleaving model, on the other hand, forces simultaneous events to be sequentialized artificially. Interleaving is considered inadequate by some for handling certain situations involving simultaneity in a meaningful manner [Mok 1991]. Others find that it is possible to incorporate time into an interleaving model to represent real-time adequately [Ostroff 1989].

Key questions about the applicability of the language to real-time systems, then, include:

(1) Can the language express absolute and relative timing constraints?

(2) Is the language's model of time discrete or continuous?

(3) Can timing constraints be expressed only in terms of events observable to the system in its operating environment?

(4) Can the language express stochastic requirements for deadline satisfaction?

(5) Can the language express what is required to occur if a timing constraint is missed?

(6) Can the language express parallelism? Does it use the true or interleaved model of parallelism?

2.2. Representing the Conceptual Construct

A major concern in representing a set of requirements is capturing the essential properties or *conceptual construct* [Brooks 1987] of a system while leaving unspecified those details that do not affect validity. The specification should serve as an abstraction representing exactly the set of all valid implementations, and neither overspecify (provide details that are not requirements) nor underspecify (omit details that are requirements). A specification should say *what* is required of the system and not *how* that system is implemented, i.e., it should represent a "black box" with only the externally observable behavior specified [Parnas 1979].

A difficulty in representing the conceptual construct is that inessential artifacts of a specification may be misconstrued as part of the conceptual construct. Such is often the case with specifications that use operational definitions whose details may be misinterpreted as design or implementation constraints. For example, a specification may include parallelism as a conceptual construct, but it would be a premature design-level decision to interpret this as requiring either a parallel system or a distributed system architecture (unless mandated as a constraint). Also undesirable are specification languages that use design-level concepts such as data flow models, because the specifications resulting from such techniques usually imply that a particular component architecture is required. Even though the ideal conceptual construct for a system can at best be subjectively evaluated, it is desirable that a specification language support models and notations that minimize confusion about which constructs are required properties (i.e., external behavior) and which are artifacts of the specification.

Legitimate design and implementation constraints, which tend to decrease the number of potentially valid implementations by limiting choices for designers and implementors, must be handled with care. Different notations may be appropriate for such constraints, since it is important that true constraints not be confused with similar appearing constructs that are only artifacts of the specification.

Families of systems arise anytime when a requirements specification leaves a choice open to the designer or implementor. While implicit choices left to the designer or implementor give rise to families of different implementations, we emphasize explicit requirements constructs that provide additional family concepts:

- Nondeterminism allows for different choices based upon alternate behaviors that are equally suitable.
- Abstract input or output devices define families of systems with common functionality but choice of hardware devices.
- Generic system parameters (analogous to those of the Ada programming language) give rise to families of systems that differ in the values of those system parameters.

Such concepts in a specification tend to increase the number of potentially valid implementations and may make a specification reusable or more easily modified. Language support for families of systems should be flexible enough to include the full range from narrowly defined single systems with many constraints to general families of systems.

Part of a system's conceptual construct is its operating environment. Developing a real-time system may require extensive modeling of the external environment (such as a timing model for it) in order to describe or analyze the requirements properly. For further discussion of this concern see [Heitmeyer and McLean 1983].

Key questions about capturing a system's conceptual construct, then, include:

(1) Does the language lend itself to specifying only what is required, or is it burdened by the need to include irrelevant details or other artifacts of using that language in the specification?

(2) Does the language facilitate the representation of program families by allowing abstraction of parameter values, hardware devices, and nondeterminism?

(3) Does the language provide for modeling the system's operating environment?

2.3. Formality

As with many of these criteria, formality is a spectrum quality rather than an absolute. A formal specification language has at least a precise, rigorously defined syntax. That means that it is possible to test unambiguously whether a specification is a member of the language or not. In addition, some formal languages have precisely defined semantics. A specification written in such a language has mathematical properties that can be analyzed. More to the point, it can be shown that any system that meets that specification will have certain properties. For example, desired invariants can be derived for most real-time systems; an example invariant is "the valve will always be closed within two seconds of detection of sensor temperature exceeding 212 degrees." Proving that desired invariants are implied by the specification is an indispensable exercise in making sure that the specification is valid.

Formal specifications also have the potential to be processed mechanically; for example, a correctness proof can be checked automatically, even if the proof could not be automatically derived [Liskov and Berzins 1979]. Completeness and consistency checks can be automated because there is a formal definition of both. By contrast, natural language specifications can be processed mechanically only at the most superficial levels, such as simply manipulating various blocks of text. However, the role of natural language commentary should not be overlooked for clarifying the major points of a formal specification or providing background and motivation for the decisions embodied by the formalisms.

Formal specification languages can be judged by the ease with which their specifications can be checked for a range of properties. These properties include completeness, consistency, lack of ambiguity, and verifiability, each of which is discussed below.

2.3.1. Completeness

A requirements specification is complete if it has all the information needed to define at least one system that is acceptable to the customer. This also includes support for any attributes that contribute to completeness, such as robustness—the ability to handle any possible input conditions including errors.

The specification language must tolerate incompleteness in a given requirements specification during development, although the language must also aid in detecting incompleteness so that it can eventually be eliminated. Certain constructs are needed, such as TBD's, that allow for reasonable analysis of an incomplete specification.

Key questions:

(1) Does the language include rules that define what a complete requirements specification is?

(2) Does the language tolerate incompleteness during development?

(3) Does the language facilitate rapid identification of areas of incompleteness in the specification?

2.3.2. Consistency

Consistency means that no contradiction can be derived from a set of facts. Inconsistent requirements specifications have no systems that satisfy all the requirements. A requirements specification must be internally consistent; that is, no contradiction can be deduced from within the specification. Specification languages should provide some form of internal consistency checking. A requirements specification must also be externally consistent with other products of development, such as the design, implementation, etc. Traceability (see Section 3.3.) can provide some support for external consistency checking.

If formal reasoning is associated with a specification language, then it is desirable that the underlying formal logic system have been shown to be sound. That is, any theorem derived from the specification must be true in the specification model. Although first order predicate logic is sound, special-purpose logics need to be shown to be sound also [Berg et al. 1982]. Formal reasoning is necessary to precisely define and to automate consistency checking.

Key questions:

(1) Does the language contains rules from which mechanical self-consistency checks can be derived?

(2) Does the language provide a means to perform external consistency checks with other products of the development?

2.3.3. Lack of Ambiguity

Ambiguity in a specification leads to more than one meaning, when only one is intended. A specification language should not have ambiguity at either the syntactic or semantic levels. Formal syntax and formal semantics are solutions, since formal constructs usually have unambiguous definitions. However, even the lack of ambiguity found in formal specifications may not prevent misunderstandings, if the reader does not have the appropriate background and experience in the language [Parnas 1979].

Key questions:

(1) Does the language have a formal syntax by which syntactic correctness of a specification can be unambiguously judged?

(2) Is the language semantically ambiguous?

2.3.4. Verifiability

The quality of verifiability refers to the ability to prove that some set of properties holds for a given specification. The ability to verify properties of a specification is one of the most important reasons for using formal specification languages.

Specification languages that can be verified are usually built upon some type of logic. To make verification feasible, it must be automated. Complete automation of verification requires a decidable language, i.e., one whose underlying logic is decidable. Decidable languages, however, may not be able to express all desired requirements concepts that could be expressed via an undecidable language. A compromise using some restricted subset of an undecidable language is generally sought in order to provide the requisite expressiveness. The ultimate goal is to provide an efficient automation of proofs, so that specifiers and verifiers need not be experts in proof techniques.

Key questions:

(1) Does the language have a formal semantics that will allow proofs of invariant correctness?

(2) Is the language based on a logic that has been shown to be decidable?

(3) What is the computational complexity of the automatic proof techniques, if any are provided?

2.4. Constructibility, Expressiveness, and Conciseness

A language is said to be constructible if it is able to express application domain concepts. A special case of constructibility for real-time systems was discussed in Section 2.1. Other concepts may include special language constructs for vehicle position and attitude, chemical reactions, fluid dynamics quantities, etc. The language is said to have high constructibility if (1) the way the specifiers think about the problem domain is reflected in the available constructs and in the way these constructs are combined; or (2) the specification language is expressive. Expressiveness refers to the ability to make statements about many types of properties, and the ability to describe varied functionality. So, for example, even if a particular language is not constructible with respect to both avionics and chemical processing domains because it does not have built-in concepts specific to each one, it may still be expressive enough so that specifiers can easily build suitable specifications for both domains by using more general (domain-independent) built-in features of the language.

Assuming that the language is able to express a construct at all, as discussed above, conciseness refers to its ability to express the construct with a minimum of redundant or irrelevant information. Important features common to many real-time systems, e.g., timing properties, should be expressible directly via a small number of primitives, rather than indirectly in terms of many primitives.

Alternately viewed, conciseness measures the lack of repetition (either actual or conceptual) necessary in a document. The expressive primitives are powerful because they take the bulk of common information from the specification and move it into the semantics of the language. Support for some form of abbreviations (such as macros) can also aid in concise specifications by factoring out repetitive parts of a specification. Avoiding such repetition also promotes consistency within the specification by maintaining a single definition of a concept.

Key questions:

(1) What features of the language are specifically relevant to the problem domain under consideration?

(2) What features of the language are relevant across application domains?

(3) To what degree must information (whether detailed requirements, conceptual background, or semantic constructs) be repeated in a specification written in the language under consideration?

(4) How compact is the expression of information in the language under consideration, compared to that of other languages?

2.5. Scalability

It is important that the specification language can handle scaling up from small, toy problems to production real-time systems. A major difficulty in scalability is that the complexity of large systems increases nonlinearly with the size of the system [Brooks 1987]. The best evidence of language scalability is the existence of previous application of the language to large production-quality real-time systems, along with documented evaluation of the language's tool and methodology support. Lacking such *a posteriori* evidence, the following *a priori* criteria can be used:

- The language should support vertical decomposition of a specification from the top level (most abstract) through refinement to additional more detailed levels. Consistency must be maintained among multiple vertical levels that comprise a specification.
- At each level of the vertical decomposition, there should be constructs for partitioning the specification into more manageable work assignments (horizontal decomposition).

Key questions:

(1) Is there testimonial evidence of application of the language and its methodology to production systems?

(2) Does the language support vertical or horizontal decomposition of the system?

2.6. Modifiability

Modifiability is the quality that makes a specification easy to change. Requirements for real-time systems will likely change many times during the evolution of a project due to such factors as changing environment and changing customer needs under complex technological, legal, political, and social pressures [Brooks 1987]. The language must support ease of change throughout the system's evolution.

In general, readability factors, such as indices and cross-referencing or their automated equivalents, contribute to ease of change. Additionally, structuring to facilitate anticipated changes may be beneficial. However, structuring criteria for modifiability may conflict with those for readability and scalability, and require a compromise.

Key questions:

(1) Does the language support browsing facilities (hardcopy or online) that facilitate locating related sections during modification?

(2) Does the language support the documentation of anticipated changes?

2.7. Readability

Readability is a quality that enables individuals in different roles (specifiers, customers, users, verifiers, and implementors) to understand a specification without undue difficulty. Each role has its own perspectives and assumptions, and requires different educational backgrounds, knowledge, and experience. Those portions of the specification relevant to each role should be clearly understandable to each person serving in that role.

The structure of the specification also affects readability. It is preferable that the specification language aid in separating normal processing from error processing. Including indices and cross-references (or online retrieval equivalents) also promotes understanding, as well as modifiability.

Key questions:

(1) Does the language provide for the needs of readers in different roles?

(2) Can specifications be structured for readability (e.g., normal vs. error processing)?

(3) Does the language support browsing (hardcopy or online)?

3. Process-Oriented Criteria

Development of a good specification requires the basic processes of creation, modification, and analysis. The process of creation should be supported by a method that provides guidance to the specifier. Analysis of a specification takes two basic forms. Verification (definition 1 in Glossary) and testing apply to properties such as consistency, timing, security, and reliability that are sufficiently formal to permit objective evaluation. Properties such as readability, maintainability, and suitability of the system to customer needs require a subjective evaluation or validation. Modification and analysis must normally be iterated until there is agreement with the customer that the specification is satisfactory.

At later stages of the life cycle, verification (definition 2) and testing of designs and implementations with respect to the requirements specification will occur. Traceability, as a complement to analysis, provides limited assurance that all requirements have been covered both during specification development and at later stages.

For building large real-time systems, automation of these processes in terms of tools and environment is an overriding concern, as the complexity of such systems is generally unmanageable without tool support.

3.1. Method for Specification Creation and Modification

A requirements specification language either implies or explicitly provides a method by which the language is used to create and modify a specification. The method may consist of a sequence of steps (i.e., suggestions of what to do next), procedures or heuristics for executing each step, rules for evaluating the results, etc. Guidance should be in terms of when to apply various constructs of the language, as well as how to apply these constructs effectively, especially when there may be a choice of applicable constructs. Finally, the guidance should not be so rigid as to encumber the creativity or productivity of the specifier [STARTS 1987].

Ease of use of the method should be evaluated and considered. Ideally, the method should require little formal training and mastery of few unfamiliar or complicated concepts. Of course, the benefits of the method must be weighed against its start-up cost; one might be willing to invest in a long training course if the method seemed likely to deliver significant long-term benefits. A valuable characteristic of a method is to allow decomposition of the specification task into small work assignments so that a team of specifiers can cooperate and at the same time work relatively independently.

The maturity of the method, whether potential, rudimentary, or fully mature, should be an important factor in evaluating the method associated with a specification language [Zave 1991]. Similarly, the level of method support, ranging from no support for a "bare" specification language to a specification language embedded in a methodology that addresses the entire system life cycle, should also be considered. The specification method should also be compatible with other methods used during the system life cycle.

Key questions:

(1) Is guidance or heuristics provided for creating and modifying specifications?

(2) How difficult or expensive is training in the method?

(3) Can the method support division of the specification process into independent work assignments?

(4) How mature is the method?

(5) How does the method integrate with others in the system life cycle?

3.2. Verification and Testing

A verification technique guarantees that a specification satisfies some property for all states of the system, in contrast to testing, which can only show the satisfaction of that property for some states. In evaluating a specification language, one should consider which verification and test procedures have been established, and the level of support via methods and tools to aid in such analyses.

The primary concern is verification and testing techniques that apply directly to the requirements specification; for example, a formal specification may lead to inexpensive automatic test case generation. Verification or testing of designs and implementations versus a specification will occur at later stages of system development, e.g., correctness (definition 1) of an implementation with respect to its specification. Verification and testing techniques at the design and implementation levels should also be factors in evaluating a specification language.

Key questions:

(1) Which verification and test procedures are available at requirements level?

(2) Which verification and test techniques are available during later design and implementation?

3.3. Traceability

Traceability provides for relating objects at one stage of development to objects at the next stage. Tracing between two development steps provides two major forms of compliancy checks: (1) coverage of the former system by the latter (each former object corresponds to one or more latter objects) , and (2) necessity of the latter objects (each latter object corresponds to one or more former objects).

For requirements specification there are two important forms of traceability that should be compatible with a specification language, with traceability links both forward (linking the requirements specification to another work product) and backward (linking that work product back to the requirements specification) [Davis 1990]:

- The requirements specification language should provide a means of tracing each requirement to its manifestation in the design and implementation as a rudimentary form of verification or testing. This should be supplemented by verification and testing for more complete analysis.
- The requirements specification language should provide a means of tracing each requirement to its informal expression by the customer as an aid in validation.

Key questions:

(1) Is traceability from informal customer requirements to the requirements specification supported?

(2) Is tracing from the requirements specification to design or implementation supported?

3.4. Validation

Validation that a specification satisfies the customer's needs is at present an informal process involving the specifier, the customer, and the requirements specification. Language support for specification properties, such as readability and traceability, can help in this process, as well as support for techniques such as prototyping, scenarios, and specification execution (e.g., step by step execution of a STATEMATE specification that provides visual highlighting of the currently active state [Harel et al. 1990]). Verification or testing of properties, such as timing and safety, provide additional input to the validation process.

Key questions:

(1) Which properties (e.g., readability) related to the language aid in validation?

(2) Which techniques (e.g., prototyping, specification execution) related to the language aid in validation?

3.5. Tools and Environment

Manual introduction of methods, analyses, and traceability can only provide limited support. To scale up to production systems ultimately requires well-integrated tool support, as could be provided by a CASE environment, simply to cope with the amount of data and its different uses by the people involved in the requirements specification process.

Various quality factors will affect the acceptance of automation in place of manual techniques. Tools and a supporting environment must be cost-effective. Tools should be robust, easy to learn, and easy to use. Furthermore, simple tools (such as syntax-checking editors) may suffice when the major concern is with recording the specification rather than extensive analyses [Place et al. 1990].

Some basic features of tools and environments that should be integrated with a specification language and its related method and analyses include the following:

- The environment should provide a common repository for all requirements and various useful relationships among different types of information. Special-purpose editors (e.g., structured, syntax-directed, graphical) should provide for the efficient and correct entry of requirements data in multi-user mode.
- Open, non-proprietary data formats and interfaces should be standardized to promote interoperability among tools.
- The environment should provide configuration management and version control for the various work products of the requirements process.
- The organization of the environment and the requirements data should support various analysis tools.

Key questions:

(1) Are the available tools supporting the language cost-effective, robust, easy to learn, and easy to use?

(2) Are these tools interoperable with the development environment?

4. Summary

We have developed a set of general evaluation criteria for real-time requirements specification languages. These criteria cover important properties of a specification (applicability to real-time systems, capturing the *conceptual construct, etc.*) that should be supported by a specification language, as well as techniques for analyzing those properties (verification, traceability, etc.). These general criteria are intended as a guide to the development of more detailed criteria during actual evaluations of specification languages.

We close by reminding the reader that choice of language plays only a limited role in the success of a development effort. Although a language may facilitate sound engineering practices, it is still incumbent on the engineering staff and project management to enforce those practices.

5. Acknowledgements

We would like to thank Connie Heitmeyer, Bruce Labaw, and Preston Mullen of the Naval Research Laboratory for their many useful suggestions for improving earlier versions of this document. We also gratefully acknowledge the support of and comments from Steven Howell and Michael Edwards of the Naval Surface Warfare Center.

REFERENCES AND BIBLIOGRAPHY

ANSI/IEEE Std 830-1984.

Software Engineering Tech. Committee of the IEEE Computer Soc., "IEEE guide to software requirements specification," ANSI/IEEE Std 830-1984, The Institute of Electrical and Electronics Engineers, New York, NY, 1984.

STARTS 1987.

U.K. Dept. of Trade and Industry, *The STARTS Guide, Vol. 1 (2nd ed.)*, National Computing Centre, Manchester, UNITED KINGDOM, 1987. (Excerpt reprinted in Dorfman and Thayer, *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, pp. 320-367.)

British Standards Institute 1989.

"VDM Specification Language (Draft)," (BSI IST/5/50), British Standards Institute, April 1989.

ISO 1989.

"LOTOS—a formal description technique based on the temporal ordering of observational behavior," (ISO 8807), International Organization for Standardization, 1989.

i-Logix 1989.

"STATEMATE: system documentation," version 3.0, i-Logix, Inc., Burlington, MA, 1989.

ACM/IEEE-CS 1991.

ACM/IEEE-CS Joint Curriculum Task Force, "Computing curricula 1991," *Comm. ACM 34*(6), pp. 68-84, June 1991.

Alford 1977.

M. W. Alford, "A requirements engineering methodology for real-time processing requirements," *IEEE Trans. Softw. Eng. SE-3*(1), pp. 60-68, Jan. 1977.

Alspaugh et al.

T. A. Alspaugh, S. R. Faulk, K. H. Britton, R. A. Parker, D. L. Parnas, and J. E. Shore, "Software requirements for the A-7E aircraft (Release 3)," NRL Report 9194, Naval Research Laboratory, Washington, DC, (to appear).

Auernheimer and Kemmerer 1985.

B. Auernheimer and R. A. Kemmerer, "ASLAN user's manual," TRCS84-10, Dept. of Computer Science, Univ. of Cal. Santa Barbara, Santa Barbara, CA, Mar. 1985.

Auernheimer and Kemmerer 1986.

B. Auernheimer and R. A. Kemmerer, "RT-ASLAN: a specification language for real-time systems," *IEEE Trans. Softw. Eng. SE-12*(9), pp. 879-889, Sep. 1986.

Balzer and Goldman 1979.

R. Balzer and N. Goldman, "Principles of good software specification and their implications for specification languages," in *Proc. IEEE Conf. on Specifications of Reliable Software*, IEEE Press, pp. 58-67, 1979. (Reprinted in Gehani and McGettrick, *Software Specification Techniques*, pp. 25-39.)

Berg et al. 1982.

H. K. Berg, W. E. Boebert, W. R. Franta, and T. G. Moher, *Formal Methods of Program Verification and Specification*, Prentice Hall, Englewood Cliffs, NJ, 1982.

Berry et al. 1983.
G. Berry, S. Moisan, and J. P. Rigault, "ESTEREL: towards a synchronous and semantically sound high level language for real-time applications," in *IEEE Real-Time Systems Symp.*, IEEE Computer Society Press, 1983.

Bjorner and Jones 1982.
D. Bjorner and C. Jones, *Formal Specification and Software Development*, Prentice Hall Intl., Englewood Cliffs, NJ, 1982.

Boehm 1984.
B. W. Boehm, "Verifying and validating software requirements and design specifications," *IEEE Software 1*(1), pp. 75-88, Jan. 1984. (Reprinted in Thayer and Dorfman, *System and Software Req. Engineering*, pp. 471-484.)

Brooks 1987.
F. P. Brooks, Jr., "No silver bullet: essence and accidents of software engineering," *IEEE Computer 20*(4), pp. 10-19, Apr. 1987.

Bruns et al. 1986.
G. R. Bruns et al., "Design technology assessment: the Statecharts approach," Tech. Report MCC STP-107-86, Microelectronics and Computer Technology Corp., Austin, TX, Mar. 1986.

Clarke et al. 1985.
E. M. Clarke, M. C. Browne, E. A. Emerson, and A. P. Sistla, "Using temporal logic for automatic verification of finite state systems," in *Logics and Models of Concurrent Systems*, K. R. Apt, ed., Springer-Verlag, New York, NY, pp. 3-26, 1985.

Clarke and Grumberg 1987.
E. M. Clarke and O. Grumberg, "Research on automatic verification of finite state concurrent systems," in *Annual Review of Computer Science*, 1987.

Coleman et al. 1990.
G. L. Coleman et al., "Experience in modeling a concurrent software system using STATEMATE," in *Proc. Intl. Conf. Comput. Syst. Softw. Eng. (COMPEURO '90)*, IEEE Computer Society Press, Los Alamitos, CA, pp. 104-108, 1990.

Davis 1988.
A. M. Davis, "A comparison of techniques for the specification of external system behavior," *Comm. ACM 31*(9), pp. 1098-1115, Sept. 1988.

Davis 1990.
A. M. Davis, "The analysis and specification of systems and software requirements," in *System and Software Requirements Engineering*, R. H. Thayer and M. Dorfman, ed., IEEE Computer Society Press, Washington, DC, pp. 119-144, 1990.

Dijkstra 1977.
E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

Dorfman and Thayer 1990a.
M. Dorfman and R. H. Thayer, *System and Software Requirements Engineering*, IEEE Computer Society Press, Washington, DC, 1990.

Dorfman and Thayer 1990b.
M. Dorfman and R. H. Thayer, *Standards, Guidelines, and Examples on System and*

Software Requirements Engineering, IEEE Computer Society Press, Washington, DC, 1990.

Faulk and Parnas 1988.
S. R. Faulk and D. L. Parnas, "On synchronization in hard real-time systems," *Comm. ACM 31*(3), pp. 274-287, March 1988.

Firth et al. 1987.
R. Firth, W. G. Wood, R. Pethia, L. Roberts, V. Mosley, and T. Dolce, "A classification scheme for software development methods," CMU/SEI-87-TR-41, Softw. Eng. Inst., Pittsburgh, PA, Nov. 1987.

Franklin and Gabrielian 1989.
M. K. Franklin and A. Gabrielian, "A transformation method for verifying safety properties in real-time systems," in *Proc. 10th IEEE Real-Time Systems Symp.*, IEEE Computer Society Press, Santa Monica, CA, pp. 112-123, Dec. 5-7, 1989.

Gabrielian and Franklin 1988.
A. Gabrielian and M. K. Franklin, "State-based specification of complex real-time systems," in *Proc. 9th IEEE Real-Time Systems Symp.*, IEEE Computer Society Press, Huntsville, AL, pp. 2-11, 1988.

Gabrielian and Franklin 1991.
A. Gabrielian and M. K. Franklin, "Multilevel specification of real-time systems," *Comm. ACM 34*(5), pp. 51-60, May 1991.

Gabrielian 1991.
A. Gabrielian, "HMS Machines: a unified framework for specification, verification and reasoning for real-time systems," in *Foundations of Real-Time Computing: Formal Specifications and Methods*, A. M. van Tilborg and Gary M. Koob, ed., Kluwer, Boston, MA, pp. 139-166, 1991.

Gehani and McGettrick 1986.
N. Gehani and A. McGettrick, *Software Specification Techniques,* Addison-Wesley, Reading, MA, 1986.

Ghezzi et al. 1987.
C. Ghezzi et al., "A general way to put time in Petri nets," in *Proc. 4th Intl. Workshop on Softw. Design and Spec.*, Monterey, CA, Apr. 3-4, 1987.

Ghezzi and Kemmerer 1990.
C. Ghezzi and R. A. Kemmerer, "ASTRAL: an assertion language for specifying real-time systems," Tech. Report TRCS 90-25, Dept. of Computer Science, Univ. of Cal. Santa Barbara, Santa Barbara, CA, Nov. 19, 1990. (to appear *Proc. Third European Softw. Eng. Conf.*, Milan, Italy, Oct. 1991).

Ghezzi and Mandrioli 1990.
C. Ghezzi, D. Mandrioli, and A. Morzenti, "TRIO: a logic language for executable specifications of real-time systems," *J. of Systems and Softw.*, June 1990.

Gudmundsson et al. 1989.
O. Gudmundsson, D. Mosse, A. K. Agrawala, and S. Tripathi, "MARUTI: an environment for hard real-time applications," UMIACS-TR-89-98, Univ. MD Inst. for Advanced Computer Studies, College Park, MD, 1989.

Gunter 1991.
C. Gunter, *Formal Models of Concurrency,* University of Pennsylvania, Presentation given at the Naval Research Laboratory, May 31, 1991.

Harel 1987.
D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming 8*(3), pp. 231-274, June 1987.

Harel 1988.
D. Harel, "On Visual Formalisms," *Comm. ACM 31*(5), pp. 514-530, May 1988.

Harel et al. 1990.
D. Harel et al., "STATEMATE: a working environment for the development of complex reactive systems," *IEEE Trans. Softw. Eng. 16*(4), pp. 403-413, Apr. 1990.

Harel 1991.
D. Harel, "Biting the silver bullet: towards a brighter future for system development," Tech. Report CS90-08, Dept. of Applied Math. and Computer Science, Weizmann Inst. of Science, Rehovot, Israel, Jan. 17, 1991. (to appear *IEEE Computer*).

Heitmeyer and Labaw 1991.
C. Heitmeyer and B. Labaw, "Requirements specification of hard real-time systems: experience with a language and a verifier," in *Foundations of Real-Time Computing: Formal Specifications and Methods,* A. M. van Tilborg and G. M. Koob, ed., Kluwer, Boston, MA, pp. 291-314, 1991.

Heitmeyer and McLean 1983.
C. L. Heitmeyer and J. D. McLean, "Abstract requirements specification: a new approach and its application," *IEEE Trans. Softw. Eng. SE-9*(5), pp. 580-589, Sep. 1983.

Heninger et al. 1978.
K. L. Heninger et al., "Software requirements for the A-7E aircraft (Release 1)," NRL Report 3876, Naval Research Laboratory, Washington, DC, Nov. 1978.

Heninger 1980.
K. L. Heninger, "Specifying software requirements for complex systems new techniques and their application," *IEEE Trans. Softw. Eng. SE-6*(1), pp. 2-13, Jan. 1980.

Hoare 1985.
C. A. R. Hoare, *Communicating Sequential Processes,* Prentice-Hall, Englewood Cliffs, NJ, 1985.

Jaffe et al. 1991.
M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart, "Software requirements analysis for real-time process-control systems," *IEEE Trans. Softw. Eng. 17*(3), pp. 241-258, Mar. 1991.

Jahanian and Mok.
F. Jahanian and A. K. Mok, "Modechart: a specification language for real-time systems," *IEEE Trans. Softw. Eng.,* (to appear).

Jahanian and Mok 1986.
F. Jahanian and A. K. Mok, "Safety analysis of timing properties in real-time systems," *IEEE Trans. Softw. Eng. SE-12*(9), pp. 890-904, Sep. 1986.

Jahanian and Mok 1987.
F. Jahanian and A. K. Mok, "A graph-theoretic approach for timing analysis and its implementation," *IEEE Trans. Computers C-36*(8), Aug. 1987.

Jahanian and Stuart 1988.
F. Jahanian and D. A. Stuart, "A method for verifying properties of Modechart specifications," in *Proc. 9th Real-Time Systems Symp.*, IEEE Computer Society Press, Huntsville, AL, pp. 12-21, Dec. 6-8, 1988.

Jahanian et al. 1988.
F. Jahanian, R. S. Lee, and A. K. Mok, "Semantics of Modechart in Real-Time Logic," in *Proc. 21st Hawaii Intl. Conf. on System Sciences*, Jan. 1988.

Jahanian 1989.
F. Jahanian, "Verifying properties of systems with variable timing constraints," in *Proc. 10th Real-Time Systems Symp.*, IEEE Computer Society Press, Santa Monica, CA, pp. 319-328, Dec. 5-7, 1989.

Koymans et al. 1987.
R. Koymans, R. Kuiper, and E. Zijlstra, "Specifying message passing and real-time systems with real-time temporal logic," in *ESPRIT '87 Impact and Achievement*, North Holland, Amsterdam, NETHERLANDS, 1987.

Lee and Nyberg 1988.
J. A. N. Lee and K. A. Nyberg, "Strategies for introducing formal methods into the Ada life cycle," SPC-TR-88-002, Software Productivity Consortium, Herndon, VA, Jan. 1988.

Leite 1987.
J. Leite, "A survey on requirements analysis," Tech. Report RTP 071, Univ. of Cal. Irvine, Irvine, CA, June 1987.

Leite 1988.
J. Leite, "Viewpoint resolution in requirements elicitation," Ph.D. Dissertation, Univ. of Cal. Irvine, Irvine, CA, 1988.

Levene and Mullery 1982.
A. A. Levene and G. P. Mullery, "An investigation of requirements specification languages: theory and practice," *IEEE Computer 15*(5), pp. 50-59, May 1982.

Liskov and Berzins 1979.
B. H. Liskov and V. Berzins, "An Appraisal of Program Specifications," in *Research Directions in Software Technology*, Peter Wegner, ed., MIT Press, Cambridge, MA, pp. 276-301, 1979. (Reprinted in Gehani and McGettrick, *Software Specification Techniques*, pp. 3-23)

Matelski and McKim 1989.
J. P. Matelski and J. C. McKim, "A continuous time, categorical approach to software system definition," Tech. Report, The Hartford Graduate Center, Hartford, Conn., Oct. 23, 1989.

Melhart et al. 1988.
B. E. Melhart, N. G. Leveson, and M. S. Jaffe, "Analysis capabilities for requirements in Statecharts," Technical Report, Department of Information and Computer Science, University of California Irvine, Irvine, CA, Sep. 1988.

Milner 1989.
R. Milner, *Communication and Concurrency,* Prentice-Hall, Englewood Cliffs, NJ, 1989.

Mok 1991.
A. K. Mok, "Towards mechanization of real-time system design," in *Foundations of Real-Time Computing: Formal Specifications and Methods,* A. M. van Tilborg and G. M. Koob, ed., Kluwer, Boston, MA, pp. 1-38, 1991.

Ostroff 1989.
J. S. Ostroff, *Temporal Logic for Real-Time Systems,* Research Studies Press Ltd. (distributed by John Wiley and Sons), England, 1989.

Ostroff 1991.
J. S. Ostroff, *Survey of formal methods for the specification and design of real-time systems,* Dept. of Computer Science, York University, North York, Ontario, Sept. 1991. (To appear in IEEE Press book *Tutorial on Specification of Time.*)

Parnas 1979.
D. L. Parnas, "The role of program specifications," in *Research Directions in Software Technology,* Peter Wegner, ed., MIT Press, Cambridge, MA, pp. 364-370, 1979.

Parnas et al. 1988.
D. L. Parnas, A. J. van Schouwen, and S. P. Kwan, "Evaluation standards for safety critical software," Tech. Report TR88-220, Department of Computer Science, Queen's University, Kingston, Ontario, May 1988.

Parnas and Wang 1989.
D. L. Parnas and Y. Wang, "The trace assertion method of module interface specification," Tech. Report 89-261, Dept. of Comput. and Inform. Science, Queen's University, Kingston, Ontario, Oct. 1989.

Pedersen and Klein 1988.
J. Pedersen and M. Klein, "Using the Vienna Development Method (VDM) to formalize a communication protocol," CMU/SEI-88-TR-26, Softw. Eng. Inst., Pittsburgh, PA, Nov. 1988.

Place et al. 1990.
P. Place, W. Wood, and M. Tudball, "Survey of formal specification techniques for reactive systems," Tech. Report CMU/SEI-90-TR-5, Software Engineering Institute, Pittsburgh, PA, May 1990.

Pnueli 1985.
A. Pnueli, "Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends," in *Current Trends in Concurrency,* Springer-Verlag, New York, NY, pp. 510-584, 1985.

Pnueli and Harel 1988.
A. Pnueli and D. Harel, "Applications of temporal logic to the specification of real-time systems," in *Formal Techniques in Real-Time and Fault Tolerant Systems,* M. Joseph, ed., Springer-Verlag, New York, NY, 1988.

Ramamoorthy and So 1978.
C. V. Ramamoorthy and H. H. So, "Software requirements and specifications: status and perspectives," in *Tutorial: Software Methodology,* C. V. Ramamoorthy and R. T. Yeh, ed., IEEE Computer Society Press, Washington, DC, pp. 43-164, 1978.

Reisig 1985.
W. Reisig, *Petri Nets, An Introduction*, Springer-Verlag, New York, NY, 1985.

Roman 1985.
G. Roman, "A Taxonomy of Current Issues in Requirements Engineering," *IEEE Computer 18*(4), pp. 14-22, Apr. 1985.

Rose et al. 1991.
A. Rose, C. Heitmeyer, and B. Labaw, "Languages for requirements specification: experience with a tabular approach," in *Proc. 1991 Systems Design Synthesis Technology Workshop*, Naval Surface Warfare Center, Silver Spring, MD, pp. 107-113, Sep. 10-13, 1991.

Sannella 1988.
D. Sannella, "A survey of formal software development methods," ECS-LFCS-88-56, Edinburgh Univ., Edinburgh, SCOTLAND, July 1988.

Shooman 1983.
M. L. Shooman, *Software Engineering: Design, Reliability, and Management*, McGraw-Hill, New York, NY, 1983.

Smith and Gerhart 1988.
S. L. Smith and S. L. Gerhart, "STATEMATE and cruise control: a case study," in *Proc. COMPAC '88, 12th Intl. IEEE Comput. Softw. and Application Conf.*, IEEE Press, New York, NY, pp. 49-56, 1988.

Stankovic and Ramamritham 1988.
J. A. Stankovic and K. Ramamritham, "Chapter 1: Introduction," in *Tutorial: Hard Real-Time Systems*, J. A. Stankovic and K. Ramamritham, ed., IEEE Computer Society Press, Washington, DC, pp. 1-11, 1988.

Stuart 1990.
D. A. Stuart, "Implementing a verifier for real-time systems," in *Proc. 11th Real-Time Systems Symp.*, IEEE Computer Society Press, Lake Buena Vista, FL, pp. 62-71, Dec. 5-7, 1990.

Thayer and Thayer 1990.
R. H. Thayer and M. C. Thayer, "Glossary," in *System and Software Requirements Engineering*, R. H. Thayer and M. Dorfman, ed., IEEE Computer Society Press, Washington, DC, pp. 605-676, 1990.

Turski and Maibaum 1987.
W. M. Turski and T. S. E. Maibaum, *The Specification of Computer Programs*, Addison-Wesley, Reading, MA, 1987.

van Schouwen 1990.
A. J. van Schouwen, "The A-7 requirements model: re-examination for real-time systems and an application to monitoring systems," Tech. Report 90-276, Queen's Univ., Kingston, Ontario, May 1990.

White 1987.
S. White, "A pragmatic formal method for computer system definition," Ph.D. Dissertation, Polytechnic Inst. of New York, New York, NY, June 1987.

Wing 1990.
J. M. Wing, "A specifier's introduction to formal methods," *IEEE Computer 23*(9), pp. 8-24, Sep. 1990.

Wood and Wood 1989.
D. P. Wood and W. G. Wood, "Comparative evaluations of four specification methods for real-time systems," Tech. Report CMU/SEI-89-TR-36, Software Engineering Institute, Pittsburgh, PA, Dec. 1989.

Yeh et al. 1984.
R. T. Yeh, P. Zave, A. P. Conn, and G. E. Cole, Jr., "Software Requirements: New Directions and Perspectives," in *Handbook of Software Engineering*, C. R. Vick and C. V. Ramamoorthy, ed., Van Nostrand Reinhold, New York, NY, pp. 519-543, 1984.

Zave and Yeh 1981.
P. Zave and R. T. Yeh, "Executable requirements for embedded systems," *Proc. Fifth Intl. Conf. Softw. Eng.*, pp. 295-304, March 1981.

Zave 1990.
P. Zave, "A comparison of the major approaches to software specification and design," in *System and Software Requirements Engineering*, R. H. Thayer and M. Dorfman, ed., IEEE Computer Society Press, Washington, DC, pp. 197-199, 1990.

Zave 1991.
P. Zave, "An insider's evaluation of PAISley," *IEEE Trans. Softw. Engin. 17*(3), pp. 212-225, Mar. 1991.

GLOSSARY

acceptable	Satisfying the customer's "real" requirements for a system. The customer's requirements may not be the same as those actually in the requirements specification, since the requirements specification may not correctly capture the "real" requirements. Synonym for valid.
abstraction	The process and product of choosing only certain attributes from many that exist of an object or concept. The chosen attributes are important with respect to some goal.
ambiguity	Lack of precision (fuzziness), which allows multiple interpretations of a given aspect of a specification, at least one of which would lead to an unacceptable implementation.
analysis	1. Process of testing or verifying that a system has certain properties, for example, those described in its requirements specification. 2. Process of determining if the requirements in the specification are consistent with the "real" customer requirements (i.e., validation).
completeness	Quality that all relevant information for developing an acceptable implementation has been included in the specifications. With incomplete specifications it is possible to develop at least one unacceptable implementation.
conceptual construct	The essential properties of a specified system. This represents what is needed to specify the valid implementations of the system—no more (overspecification) and no less (underspecification).
conciseness	Compact expression of a concept.
consistency	1. (Internal consistency) Quality of a requirements specification such that there are no contradictions or conflicts among any of its parts. 2. (External consistency) Quality of a requirements specification that there are no contradictions or conflicts between the specification and another product of the development process.
constraint	Any decision that limits the set of valid implementations of a system. These may be hardware constraints (e.g., computer X must be used), software constraints (e.g., database package D must be used), or non-functional properties such as performance

	and reliability.
correctness	1. Quality of having consistency between the requirements specification and any of the other development products (for example, design specification or the implementation). 2. Quality of having consistency between the customer's "real" requirements for a system and the final system.
customer	The person(s) who contracts and pays for the development of a system. The customer usually (but not necessarily) defines the system requirements [ANSI/IEEE Std 830-1984].
distributed system	A system operating with multiple processors, and one or more of those processors shares no common memory with the others. This makes message passing for communication a requirement.
environment	1. (External) environment: The external conditions and interfaces under which a system operates. 2. (Software engineering) environment (SEE): The collection of computers, support software, procedures and facilities that make the tools and methodologies used by software developers easily available [Thayer and Thayer 1990].
formal specification	A formal specification is one that has an effective procedure to tell whether a specification has a particular property of interest [Gunter 1991]. This type of specification tends to be mathematically oriented, and it requires more knowledge and experience than the informal type to understand. Informal explanatory comments may elucidate formal specifications.
formal verification	Verification in which the proof could be recognized mechanically to be a proof, regardless of whether the proof was developed by hand or (partially) automatically generated. (Also see verification.)
hard real-time system	System in which deadlines for critical tasks (e.g., start or completion times) must be met to prevent some catastrophe, or to reduce the probability that a catastrophe would result.
informal specification	A specification that is written largely using natural language rather than formal mathematical notations. The syntax and notation may be largely ad hoc, rather than consistent in the manner of formal specifications. An informal specification may contain free-form commentary as part of the specification itself (as contrasted to informal explanatory comments used with formal specifications). An informal specification is amenable to

	very limited machine manipulation.
informal verification	Verification in which the proof cannot be recognized mechanically to be a proof. This type of verification tends to be ad hoc and to be expressed in natural language, or informal notations (e.g., ones invented ad hoc, or mixtures of various notations). (Also see verification.)
maintenance	The process of fixing errors encountered in a system once it is in operational use, or changing it to satisfy new requirements.
model	An abstraction that includes all essential properties of the process or object being modeled, but does not include any irrelevant properties. Relevance is determined by how the model will be used. A specification is a model of the system to be developed.
nondeterminism	Property that an observer cannot tell which of several behaviors should be chosen, since more than one is acceptable. It can be a desirable characteristic in a specification. In addition, however, repeatability may also be desired in some subset of these cases, so that once a particular behavior is chosen later in the development process, it may be a requirement to use that one alone wherever this nondeterministic requirement appears [Parnas and Wang 1989].
notation	The means of expressing the structure (i.e., syntax) of a given language unit (e.g., sentences in English, or propositions in logic) via the composition of symbols. The notation for expressing syntax is not the same as the syntax.
overspecify	1. To put extra information into the specification of a software system to the extent that it excludes at least one acceptable implementation (due to the extra information being inconsistent, being design information not appropriate to describing the externally visible behavior, etc.). 2. To put undesirably redundant information into the specification of a software system [Place et al. 1990].
parallel system	A system with multiple processors that communicate via shared memory.
precise	Well-defined. Precision is a major requirement for automated processing.
real-time system	Any system that must operate under some form of timing constraints. (See soft and hard real-time systems).

requirements specification	Product that defines the acceptable implementations for a software system subject to constraints (which may include performance, operating, interface, and economic constraints) [Heitmeyer and McLean 1983, Roman 1985]. These constraints should be minimal so that no useful implementations are precluded [Zave and Yeh 1981].
semantics	The meaning of a construct as opposed to its syntax.
simulation	Process of testing important system properties by executing a model of the proposed system. The model may have artificial (simulated) components for any of the computer hardware, environment, or software functionality [Shooman 1983].
soft real-time system	A real-time system that has stochastic (rather than deterministic) timing constraints.
specification	Description of the essential properties of an object (system, software, program, etc.). Specifications may be informal or formal.
specification language	Any method providing the basic concepts and relations for expressing specifications. Specification languages include formal languages as well as less formal constructs.
syntax	The structure of a construct as implied by the rules for composing it from subcomponents (as opposed to its semantics). The syntax rules may manifest themselves through different notations.
testing	Any process (e.g., regression testing) for establishing confidence in the truth of some property of a specification, design, implementation, etc. by checking or executing some subset of the total possible outcomes. In contrast to verification, testing can never absolutely verify (definition 1) some property (except exhaustive testing of all possibilities).
traceability	Identification and recording of the links between requirements and the manifestation of those requirements in other products of the software development process (informal customer requirements, design, implementation, test plans, etc.)
underspecify	To specify without enough detail (i.e., incompletely, ambiguously, etc.) such that the specification admits at least one unacceptable implementation [Place et al. 1990].

user	Person(s) who interacts directly with a system. Users and customers are not necessarily the same people [ANSI/IEEE Std 830-1984].
valid	Synonym for acceptable.
validation	Process of determining that a software system satisfies the customer's needs. "Am I building the right product [Boehm 1984]?"
verification	1. Process of proving that a specification satisfies some property (for all situations). 2. Process of determining if products of one phase of software development satisfy requirements of previous phase. "Am I building the product right [Boehm 1984]?"

DISTRIBUTION

	Copies
Defense Technical	
Information Center	
Cameron Station	
Alexandria, VA 22304-6145	12
Library of Congress	
Attn: Gift and Exchange Division	4
Washington, DC 20540	
Naval Air Development Center	
Attn: Code 7033	
(Dr. C. Schmiedekamp)	1
(P. Zombori)	1
Warminster, PA 18974-5000	
Office of Naval Technology	
Attn: Code 227	
(Elizabeth Wald)	1
800 N. Quincy Street	
Arlington, VA 22217-5000	
Center for Naval Analyses	
4401 Fort Avenue	
P.O. Box 16268	
Alexandria, VA 22302-0268	2
Naval Research Laboratory	
Attn: Code 5534	
(Connie Heitmeyer)	1
(Bruce Labaw)	1
Washington, DC 20375	
Naval Postgraduate School	
Attn: Code AS/RA	
(Balasubramaniam Ramesh)	1
Administrative Sciences Department	
Monterey, CA 93943	
United States Army	
CECOM, C2NVEO	
Attn: AMSEL/RD/VNT/TST	
(H. Nguyen)	1
Fort Belvoir, VA 22060	

	Copies
Advanced Technology &	
Research Corp.	
Attn: Adrien J. Meskin	5
George Stathopoulos	1
14900 Sweitzer Lane	
Laurel, MD 20707	
General Electric—Aerospace	
Attn: Steven Sietz	1
Mail Stop 127-333	
199 Borton Landing Road	
Moorestown, NJ 08057-3075	
Bell Laboratories	
Attn: Suzanne Edwards	1
Room 3B-326	
101 Crawfords Corner Road	
Holmdel, NJ 07733	
Internal Distribution:	
D4	1
E231	2
E232	3
E342 (GIDEP)	1
F01	1
F31 (W. Laposata)	1
G07 (F. Moore)	1
G42 (T. Dumoulin)	1
G42 (A. Farsaie)	1
G42 (J. Moscar)	1
G42 (E. Ogata)	1
G42 (J. Youngblood)	1
G70 (D. Dorsey)	1
G72 (H. Parks)	1
H32 (J. Miller)	1
K02	1
K10 (J. Sloop)	1
K12 (J. O'Toole)	1
K13 (D. Parks)	1
K14 (D. Clark)	1
K41 (L. Gross)	1
K51 (J. Smith)	1
K52 (G. Brooks)	1
K52 (W. Farr)	1

DISTRIBUTION (Cont.)

	Copies
K52 (H. Huber)	1
N15 (M. Wilson)	1
N33 (J. Sizemore)	1
N35 (M. Masters)	1
N35 (F. Riedl)	1
N35 (M. Zarin)	1
R44 (E. Cohen)	1
R44 (H. Szu)	1
R44 (J. Zien)	1
U	1
U02	1
U042	1
U10	1
U20	1
U23 (W. Dence)	1
U23 (R. Fitzgerald)	1
U23 (J. Horner)	1
U23 (M. Richards)	1
U23 (P. Winters)	1
U25	1
U25 (D. Bergstein)	1
U25 (E. Hein)	1
U30	1
U33	1
U302 (P. Hwang)	20
U33 (D. Choi)	1
U33 (M. Edwards)	20
U33 (K. Murphy)	1
U33 (N. Hoang)	1
U33 (S. Howell)	10
U33 (M. Jenkins)	1
U33 (T. Moore)	1
U33 (C. Nguyen)	1
U33 (T. Park)	1
U33 (H. Roth)	1
U33 (M. Trinh)	1
U33 (P. Wallenberger)	1
U40	1

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
27 September 1991

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
A Methodology for Systems Requirements Specification and Traceability for Large Real-time Complex Systems

5. FUNDING NUMBERS

6. AUTHOR(S)
Michael Edwards and Steven L. Howell

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Surface Warfare Center (Code U33)
10901 New Hampshire Avenue
Silver Spring, MD 20903-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NAVSWC TR 91-584

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document describes the beginnings of a methodology for requirements specification and traceability of real-time, large-scale, complex computer-intensive systems. The method is aimed at better understanding the top-level system requirements and how they relate to the system under design. The methodology will cover the requirements aspects of system development over the entire system development life cycle, beginning with the specification of the requirements and tracing those requirements to the design and final implementation.

14. SUBJECT TERMS
Requirements Specification; Traceability; Natural Interfaces; Formal Requirements; Formal Methods; Complex Linking; Large, Complex Systems; Requirements Engineering

15. NUMBER OF PAGES
62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102