A METHODOLOGY-INDEPENDENT REQUIREMENTS
TRACEABILITY PROTOTYPE

THESIS

Presented to the Graduate Council of the
North Texas State University in Partial
Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

By

Charles Warren Moseley, B.A.
Denton, Texas
December, 1984

The production of quality software is difficult to control with the growing complexity and size of the systems being constructed. This places a new emphasis on the support systems, tools, and environments that support the software development. There have been efforts to build requirements traceability into systems, but the requirements were defined as a part of a problem-specific methodology.

The purpose of this investigation was to try and determine if it was possible and feasible to build a methodology-independent system to trace requirements throughout the development life cycle.
A METHODOLOGY-INDEPENDENT REQUIREMENTS
TRACEABILITY PROTOTYPE

THESIS

Presented to the Graduate Council of the
North Texas State University in Partial
Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

By

Charles Warren Moseley, B.A.
Denton, Texas
December, 1984
# TABLE OF CONTENTS

**LIST OF ILLUSTRATIONS** ........................................ iv

**Chapter**

I. OVERVIEW OF REQUIREMENTS ENGINEERING ..................... 1

II. A RESPONSE TO THE STARS PROGRAM ........................... 9

III. REQUIREMENTS TRACEABILITY PROTOTYPE ..................... 13

IV. DEFINING THE DEVELOPMENT SYSTEM ........................... 14

V. ORIGINATING REQUIREMENTS .................................... 18

VI. WORKING REQUIREMENTS SPECIFICATIONS ........................ 35

VII. THE REQUIREMENTS SPECIFICATIONS DOCUMENTS ............. 40

VIII. EXTERNAL CONSIDERATIONS FOR THE PROTOTYPE .......... 49

IX. FUTURE CONSIDERATIONS FOR THE PROTOTYPE ................ 51

APPENDICES ................................................................ 54

BIBLIOGRAPHY ......................................................... 89
LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Example of Main Menu</td>
<td>13</td>
</tr>
<tr>
<td>2.</td>
<td>Example of System Definition Menu</td>
<td>15</td>
</tr>
<tr>
<td>3.</td>
<td>Input/Update Screen to Define the Development System</td>
<td>16</td>
</tr>
<tr>
<td>4.</td>
<td>Log/Report Originating Requirements Menu</td>
<td>19</td>
</tr>
<tr>
<td>5.</td>
<td>Input/Update of Originating Document</td>
<td>20</td>
</tr>
<tr>
<td>6.</td>
<td>Screen for Input/Update of Association between Relevant Originating Requirements and a Development System</td>
<td>22</td>
</tr>
<tr>
<td>7.</td>
<td>Input/Update Screen for Explicit Originating Requirements</td>
<td>24</td>
</tr>
<tr>
<td>8.</td>
<td>Input/Update of Association between Implicit Requirements Identifier and a Development System</td>
<td>25</td>
</tr>
<tr>
<td>9.</td>
<td>Input/Update Screen for Implicit Requirements (Form I)</td>
<td>26</td>
</tr>
<tr>
<td>10.</td>
<td>Input/Update Screen for Implicit Requirements (Form II)</td>
<td>27</td>
</tr>
<tr>
<td>11.</td>
<td>Screen for Tracing Information between Implicit and Source Requirements</td>
<td>29</td>
</tr>
<tr>
<td>12.</td>
<td>Input/Update of Association between an Inexplicit Requirement Identifier and a Development System</td>
<td>30</td>
</tr>
<tr>
<td>13.</td>
<td>Input/Update Screen for Inexplicit Requirements</td>
<td>31</td>
</tr>
<tr>
<td>14.</td>
<td>Input/Update Inexplicit Requirements (Form II)</td>
<td>32</td>
</tr>
<tr>
<td>15.</td>
<td>Originating Requirements</td>
<td>33</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>16</td>
<td>Log/Record Working Specifications Facility</td>
<td>36</td>
</tr>
<tr>
<td>17</td>
<td>Input/Update Screen to Record and Maintain Working Requirements and Specifications Information</td>
<td>37</td>
</tr>
<tr>
<td>18</td>
<td>Input/Update of Association between Working Specifications</td>
<td>38</td>
</tr>
<tr>
<td>19</td>
<td>Input/Update of Tracing Information between Working Specifications and Other Requirements/Specifications</td>
<td>39</td>
</tr>
<tr>
<td>20</td>
<td>Log/Report Requirements Document Menu</td>
<td>40</td>
</tr>
<tr>
<td>21</td>
<td>Input/Update to Define Requirements Specifications Documents</td>
<td>41</td>
</tr>
<tr>
<td>22</td>
<td>Input/Update of Association between Relevant Requirements Specification Document and a Development System</td>
<td>43</td>
</tr>
<tr>
<td>23</td>
<td>Input/Update to Record Requirements Specifications</td>
<td>44</td>
</tr>
<tr>
<td>24</td>
<td>Input/Update of Tracing Information between Source Requirement and Requirement Specification Document</td>
<td>45</td>
</tr>
<tr>
<td>25</td>
<td>Report Services Facility Menu</td>
<td>46</td>
</tr>
</tbody>
</table>
CHAPTER I

OVERVIEW OF REQUIREMENTS ENGINEERING

Introduction to Requirements Issues

Discrete, well-organized, and traceable requirements support the primary goal of quality systems development while also providing visibility and control over the systems definition process. Requirements engineering is a linearly-recursive process of defining the requirements of a system, and maintaining the integrity of those requirements throughout the entire software development life cycle. This process involves all of the tasks that occur before the systems design process begins.

It is customary to consider issues related to the what and why of the problem to be solved, not how the problem is to be solved. For example, if the customer's environment demands that a certain action occur in 750 milliseconds, then it is a requirement of the system to respond in 750 milliseconds. This is a requirement that should be stated and stored as such. The developer's ability to produce the 750 milliseconds response is a design related issue, and belongs at a later place in the life-cycle. The products of the requirements definition phase need to be evaluated
for (1) completeness, (2) consistency, (3) testability, and (4) traceability. The completeness item, the consistency item, and the testability item depend on the ability to trace the specified requirements.

The goal of requirements engineering is to thoroughly evaluate the needs which a system must satisfy (6). Phil Crosby (1) defines quality as conformance to requirements. If the quality concept is to be applied to the software development process then it is necessary to define software requirements that are unique and explicitly measurable.

Statement of the Problem

Jack Distasto (3), in his article "Software Management: A Survey of the Practice in 1980," summarizes nicely some of the problems that currently plague the software development process. There are four principal areas addressed in this article. They are (1) obtaining satisfactory software requirements, (2) improving software cost estimating, (3) achieving significant productivity improvements, and (4) maintaining control and visibility of software development projects. Distasto claims that "obtaining satisfactory software requirements remains the single largest obstacle to software project success" (3, p. 1003).

The need for traceability.--Quality requirements are dependent upon the systems engineer's ability to first identify the discrete requirements and then to organize
these requirements for further analysis and definition. Throughout the entire software development life-cycle, and especially during the definition phase, it is necessary to evaluate the impact of changes to the requirements process. Someone must be in the position to determine the ramifications of changes to the systems requirements. Once changes are identified and evaluated then traceability provides the thread for updates to the requirements documentation. The impact of a change to each requirement of a system is then evaluated and the proper corrective action taken. By storing identifiable discrete requirements the process is simple and visible.

The need for automation.—Why do these problems exist and what are some of the beginning steps to help correct this large expensive problem? An immediate step to help this problem is the automation of the requirements engineering process. A long-term solution is to build a totally integrated software development environment that supports all phases of software development. Systems of that nature most often evolve over long time spans and are not easily constructed. Some immediate steps to help correct this problem are to

1. Get the requirements of the user into some easily changeable media, and try to insure that the needs of the user and not the wishes of the software developer are the primary issues addressed in the software development process;
2. Establish a well-defined link of communication and mutual trust between user and developer;

3. Make it easy enough to propose changes to the requirements, access the impact of the proposed changes, and implement the changes. Early detection of errors can prevent costly project expenses in the later phases of the software life-cycle;

4. Establish a habit of enforcement of update to the requirements documentation, so that changes are always clear at any given point in the software development life-cycle. These procedures need to be simple enough to not be the bottleneck in the flow of information in the definition process; and

5. Make as smooth a transition as possible from the storage and retrieval of requirements to the automatic generation of software test plans from the stored requirements.

Synthesis of Related Requirements

There has been significant progress in the area of requirements engineering, but existing systems require that you fit your problem to their mold (4, 5). These systems require that the systems designer use the prescribed methodology of that system. These methods and procedures are not generic, and hence require adherence to a particular methodology to get the job or class of jobs done. The emphasis of methodology independence in the requirements process is not a trivial task, but one that will be the focus of much research and development in the years to come. These efforts have been helpful but are methodology dependent, difficult to use, and do not provide the foundation to help develop a generic approach for the development of software.
There are two major well recognized tools to support the requirements portion of the software life cycle. These tools are (1) SREM--Software Requirements Engineering Methodology and (2) PSL/PSA--Problem Statement Language/Problem Statement Analyzer.

There have been other attempts at the use of project-related information, but none that apply to the actual statement of requirements and the manipulation of the requirements. The major distinguishing factor in the current approach and the suggested other requirements systems is that previous systems are methodology dependent and this approach proposes a more generic solution.

Software Requirements Engineering Methodology

SREM is a methodology for the specification of large missile defense systems. It was developed for the United States Army Ballistic Missile Defense program. The foundation of SREM is the concept of a network of stimulus/response conditions. The concept of traceability is supported by the mapping of these stimulus response conditions into a collection of networks. It is the correct building of the interconnected networks that build the traceability relation. SREM provides a very powerful integrated software development environment. If the development of a system requires many stimulus/response conditions, then SREM can be of great help in the area of synthesizing a solution to the problem.
However many problems of the real world cannot be made to fit this particular model of the system representation process. The application of SREM to other types of problems (e.g., artificial intelligence applications, large business information systems, compiler design) is a very difficult task. In order to get a new software engineer trained in the use of SREM takes as much time as does the actual development of a small to medium task (5).

**Problem Statement Language/Problem Statement Analyzer**

The Problem Statement Language/Problem Statement Analyzer (PSL/PSA) is a systems development tool based on the primitive concepts of entities, relationships and attributes. This tool has been used for specification of large scale information processing systems. PSL/PSA is little beyond a tool kit to aid in the documentation and analysis of information processing systems. This system is nice if your problem fits Chen's Entity-Relationship-Attribute model (2). However, this is not always the case. A sample is that of the stimulus/response condition proposed by the applications that fit into the SREM domain. The generation of an Entity-Relationship-Attribute diagram of a series of stimulus-response conditions would be a difficult and tedious task (5).
Hypothesis

This issue of generic requirements and the ability to trace these requirements independent of any methodology is fundamental in the development of systems in which quality and reliability are to be made cost effective.

Suggested Solution

A better solution to the problem of non-generic, not-automated, methodology-dependent, and untraceable requirements is to dismiss the two above models and to determine exactly what is necessary for requirements traceability and then build a prototype of a system to support the building of systems that support the concept of generic requirements. The use of an acceptable methodology is important and it is not to be inferred in this paper that it is not important. For this reason there must be the capability to store methodology related information on demand. This ability to store methodology information does not interfere with the basic underlying traceability issue. The rational for a methodology-independent requirements traceability actually goes much deeper. It does provide the above stated features, but more important it provides the foundation for the smooth integration of requirements engineering into the software development tool kit. The development of a technology for the construction of integrated software development environments will be the focus of much research and development in the coming years.
CHAPTER BIBLIOGRAPHY


A RESPONSE TO THE STARS PROGRAM

A sample of the suggested technology research thrust is that of the STARS (Software Technology for Adaptable and Reliable Systems) program. This program is an initiative by the Department of Defense to improve software technology. An objective for this prototype was to demonstrate the ability of the Advanced Computer Systems Laboratory at Texas Instruments to participate in the development of the Integrated Support Software Environment of the STARS program. Critical to the development of such an environment is the ability to show that a STARS contractor understands and is able to support the capturing and tracing of software requirements. The STARS program should help to establish a trend for the next several years for the development of quality software on a large scale basis. Although this paper directly addresses the concept of requirements and traceability, this is in light of its use as a foundation for another level of simplification of complexity of the software development process.

Overview of the STARS Program

In February, 1982 a select group of invited software engineers met in Raleigh, North Carolina to begin the STARS
program. The STARS (Software Technology for Adaptable and Reliable Systems) program is the beginning of a five-year program to upgrade the state of practice in the area of software technology (1). Edith Martin, Deputy Under Secretary of the Department of Defense, in her keynote address, stated the following:

The threat facing the United States presents the defense community with formidable military requirements. Defense systems must be capable of deep penetration and wide range as well as have quick and flexible response, great precision, high reliability, long endurance, easy and fast mobility, and great lethality. A main ingredient in the technology-based defense strategy is the use of weapons, which have computers as integral components (3).

Martin also made it clear in her keynote address that the Department of Defense was not primarily in the software technology business, but that to maintain the edge in mission critical areas, it was essential to be the leader in the real-time embedded software area. The embedded software area poses new and more difficult problems than does the conventional applications programming project. When viewed in the light of a military strategy based on high technology, computers and software become critical defense resources, and in software as in no other computer application area are the requirements as difficult or the timing as critical. With the emphasis of the STARS program and the VHSIC (Very High Speed Integrated Circuit) program, the United States plans to stay ahead (3).
Support Systems Task Area

There are many areas to be addressed by the STARS effort. One of the most fundamental areas to be addressed is the Support Systems Task Area. This area will address the automation of tools, methods, and the integration of these methods into a coherent and consistent environment for the development of reliable software. The November, 1983 journal of the Computer Society of the IEEE is dedicated to the explanation of the essential features of STARS (1). The range of activities in the Support Systems Task Area include the following:

1. Consolidate and develop environments and methods supporting the development and maintenance of software systems;

2. Develop enhancements to the environments by integrating more advanced automated tools;

3. Conduct research and development to support continued improvement in support systems technology;

4. Demonstrate the value and effectiveness of the methods and environments developed; and

5. Facilitate the wide availability and use of the end products (2).

The Support Systems Task Area promises to provide the necessary funding for research and development in the area of software technology.
CHAPTER BIBLIOGRAPHY


CHAPTER III

REQUIREMENTS TRACEABILITY PROTOTYPE

The Requirements Traceability Prototype was developed to assist in tracing originating (source) requirements throughout the entire system development life-cycle. The first phase of the Requirements Traceability Prototype development provides a requirements traceability facility for tracing requirements from originating (source) requirements to the requirements specifications developed for each Computer Software Configuration Item (CSCI).

The Requirements Traceability Prototype is a menu driven system and future versions offer interactive assistance in its use and application. The initial (or main) menu that is presented to the user upon entry to the system is represented below (Figure 1).

REQUIREMENTS TRACEABILITY PROTOTYPE

SELECT ONE OF THE FOLLOWING FACILITIES:

A--(SUB)SYSTEM DEFINITION
B--LOG/REPORT ORIGINATING (SOURCE) REQUIREMENTS
C--LOG/REPORT WORKING REQUIREMENTS SPECIFICATION
D--LOG/REPORT REQUIREMENTS DOCUMENT(S)
END--(EXITS)

Fig. 1--Example of Main Menu
CHAPTER IV

DEFINING THE DEVELOPMENT SYSTEM

Initial considerations in the use of the prototype involve the definition of a development system and proper initialization of the prototype. The size of the respective development system and/or project should dictate how the Requirements Traceability Prototype should be initialized and development systems defined to the Requirements Traceability Prototype. For an average sized project and/or system, a database is defined and initialized that contains all the information required by the Requirements Traceability Prototype for the project and/or system under development. The Requirements Traceability Prototype uses a centralized database which contains all the necessary data to provide requirements traceability.

Experience with the use of the Requirements Traceability Prototype dictates optimal efficiency decisions to be made when using the system for a given project/system. It is assumed that for each development system an originating documentation package has been developed specifying all originating requirements. This document package, if conforming to MIL-STD-SDS (Military Standard Software
Development Guide), may be comprised of the System/Segment Specification and a Statement of Work Document.

System Definition Menu

A menu, represented by Figure 2, is displayed to the user upon entry to the (Sub) System Definition Facility.

REQUIREMENTS TRACEABILITY PROTOTYPE
(SUB) SYSTEM DEFINITION FACILITY

SELECT ONE OF THE FOLLOWING:

A--INPUT/UPDATE DEVELOPMENT SYSTEM INFORMATION
B--REPORT ALL (SUB) SYSTEMS DEFINED TO THE PROTOTYPE
END--(EXITS)

Fig. 2--Example of System Definition Menu

Input and Update of Development System Information

Prior to using the prototype and performing any prototype functions, the development system must be defined to the system. This information is stored in the Requirements Traceability Prototype database and is required to perform most all system functions. A five-character identifier (acronym) must be chosen for the development system.

The following Input/Update screen, represented by Figure 3, allows the definition of each system to the prototype.
INPUT/UPDATE DEVELOPMENT SYSTEM INFORMATION

SYSTEM IDENTIFIER: _____ (5 characters)

SYSTEM TITLE: ________________________________

CONTRACT NUMBER: __________________________

CONTRACTING AGENCY: ________________________

CONTRACT DATE: __________ DD-MMM-YY

SYSTEM DESCRIPTION:

_________________________________________________________________

_________________________________________________________________

Fig. 3--Input/Update Screen to define the Development System.

The following list describes each of the fields where data are requested for Figure 3.

1. Development System Identifier.--This is a logical Identifier (acronym) that the system development team assigns to the system under development. This logical identifier must be five characters or less and must be unique for each system defined to the prototype.

2. Development System Title.--This is the full title of the development system.

3. Contract Number.--This is the contract number associated with the system development effort.

4. Contracting Agency.--(Self explanatory.)

5. Contract Date.--Date of contract initiation. DD-MMM-YY (2 digits for day, month abbreviated to 3 characters, 2 digits for year.)

All Development Systems Defined to the Prototype

A report is available that lists all of the systems that have been defined to the prototype database. The report includes, for each development system, the following information: (1) System Title, (2) System Identifier, (3) Contract Number, (4) Contract Date, (5) Contracting Agency, and (6) System Description. See Appendix A for an example of this report.
CHAPTER V

ORIGINATING REQUIREMENTS

As a premise to a system development effort, either formally or informally, originating (source) requirements are established to initiate system conceptualization and further requirements specifications. A formal description of system objectives and functional scope are usually documented in a Request For Proposal (RFP) document (within the Texas Instruments environment). The customer and/or the project involved in the system development effort composes the formal document(s) which articulates all requirements established prior to initiation of the system development effort. For the purpose of providing, to the customer and/or project team, traceability of requirements throughout the development activity, it is essential to "pinpoint" each originating requirement and then trace how and where that requirement influences further requirement specification, design, implementation and test decisions.

We have defined (for the purpose of easing the traceability problem) three categories of Originating Requirements.

1. Explicit Requirements.--Documented (source) originating requirements. These requirements are established/documentated in an accountable document such as the Statement of Work.
2. **Implicit Requirements.**--Requirements that are not articulated directly in an originating (source) document, but are felt to be implicit within the specifications of those originating (source) requirements. Implicit requirements can and should be traceable to explicit requirements (described above).

3. **Inexplict Requirements.**--Undocumented or informally established originating (source) requirements. Inexplicit requirements (as defined here) cannot be traceable to explicit or implicit requirements (described above).

**Log/Report Originating Requirements Menu**

A menu, represented by Figure 4, is displayed to the user upon entry of the Log/Report Originating Requirements Facility.

**REQUIREMENTS TRACEABILITY PROTOTYPE**
**LOG/REPORT ORIGINATING (SOURCE) REQUIREMENTS MENU**
SELECT ONE OF THE FOLLOWING:
A--RECORD ORIGINATING (SOURCE) REQUIREMENTS DOCUMENT
B--ASSOCIATE ORIGINATING (SOURCE) DOCUMENT TO A DEVELOPMENT (SUB) SYSTEM
C--RECORD EXPLICIT REQUIREMENTS
D--RECORD ASSOCIATION BETWEEN IMPLICIT REQUIREMENTS AND DEVELOPMENT SYSTEM
E--RECORD IMPLICIT REQUIREMENTS (FORM I)
F--RECORD IMPLICIT REQUIREMENTS (FORM II)
G--RECORD TRACING BETWEEN IMPLICIT AND EXPLICIT REQUIREMENTS
H--RECORD ASSOCIATION BETWEEN INEXPLICIT REQUIREMENTS AND DEVELOPMENT (SUB) SYSTEM
I--RECORD INEXPLICIT REQUIREMENTS (FORM I)
J--RECORD INEXPLICIT REQUIREMENTS (FORM II)
K--REPORT SERVICES FACILITY

Fig. 4--Log/Report Originating Requirements Menu
Recording Originating Requirements Documents

It is necessary to record each document that contains traceable originating (source) requirements to the system. Explicit requirements is traced by reference to the particular document and location (paragraph and/or sentence within paragraph) within that document.

The prototype provides the input/update screen shown in Figure 5 to record each document containing originating (source) requirements. It is important to note that a five-character logical identifier for each originating requirement document must be defined to the system. This five-character identifier should resemble (or establish) an acronym for the document. The identifier should easily identify the document it represents, since it is used in many reports.

INPUT/UPDATE ORIGINATING REQUIREMENTS DOCUMENT

ORIGINATING DOCUMENT IDENTIFIER: ______

DOCUMENT NUMBER: ______ VOLUME: ______ REVISION: ______

TITLE: __________________________________________

DD-MMM-YY

DEVELOPER(S): _________________________________

DESCRIPTION:

Fig. 5--Input/Update of Originating Document

The following list describes each of the fields where data are requested in Figure 5.
1. **Originating Document Identifier.**—This is a logical name (acronym) that the system development team (in the use of the prototype) assigns to each originating requirements document. This logical name can be five (5) characters in length and is used in all of the reports generated by the Requirements Traceability Prototype where originating requirements documents are referenced. A logical name should be chosen so that it can easily be associated with the document it identifies, for example, if it is a Statement of Work document, the Identification chosen could be "SOW." The use of a logical name (acronym) was required to enhance readability of reports generated by the system.

2. **Document Number.**—This is a unique number assigned to the document.

3. **Volume.**—This is the volume number of the document.

4. **Revision.**—This is a revision letter/number identifying a revision of the document.

5. **Title.**—This is the full title associated with the document.

6. **Date Published.**—This is the most recent publication date associated with the document. (DD-MMM-YY.)

7. **Developer(s).**—This is the developer(s) or author(s) of the document.

8. **Description.**—This is a brief description of the assignment.

The information provided above should be adequate to identify and locate the exact document referenced.

**Tying Documents to the System**

It is necessary to associate each relevant originating (source) requirements document to the development system defined to the Requirements Traceability Prototype. More than one originating (source) requirements document may be applicable to a development system/project.
The Requirements Traceability Prototype produces the input/update screen shown in Figure 6 and ties one document to the development system. After that association is entered in the database, more documents can be tied to the system by repeating the procedure. Note that both the system and the relevant originating (source) requirements document must already be defined to the database (using the facilities described earlier in this document).

ASSOCIATE AN ORIGINATING (SOURCE) REQUIREMENTS DOCUMENT TO A DEVELOPMENT SYSTEM

ENTER:

SYSTEM IDENTIFIER: _____

DOCUMENT IDENTIFIER: __________

Fig. 6--Screen for Input/Update of association between relevant Originating Requirements and a Development System.

The following list describes each of the fields where data are requested in Figure 6.

1. System Identifier.--This is a logical identifier (acronym) that the system development team has assigned to the system under development. A database integrity constraint is applied to this field (after the input of the System Identifier) to assure that the system identifier entered is defined to the prototype (using the screen to Input/Update System Information).

2. Document Identifier.--This is a logical identifier (acronym) that the system development team assigns to the document. An integrity constraint is applied to this field to insure that the Document Identifier entered has already been defined to the database.
Recording Explicit Requirements

After a requirements document is defined to the prototype (as described in explicit requirements) it can be recorded/entered to the system. It is necessary, at the offset, to identify and extract singular requirements from the documented sources. When entering explicit originating requirements in the database, use the screen represented in Figure 7. Notice that the five-character identifier chosen for the originating requirements document is required when entering an explicit requirement. The prototype verifies that the originating document identifier entered in this screen is valid by checking to see if the document has already been defined to the database. The reference entered should correspond to the paragraph identifier (i.e., 2.3.4) or sentence number of requirement within the paragraph (i.e., 1.2.3*3 is the third sentence within paragraph 1.2.3).

The following list describes each of the fields where data are requested in Figure 7.

1. Originating Document Identifier.—This must be the corresponding logical name (acronym) that was chosen for the originating document when defined to the system. When a Document Identifier is entered, the system verifies that such a Document Identifier exists, or was defined to refer to an originating requirements document.

2. Reference.—This is a unique identifier for each singular requirement identified in the originating requirements document. The reference identifier entered should correspond to the paragraph identifier (i.e., 2.3.4) or a requirement within the paragraph (i.e., 2.3.4*3), which indicates that it
is the third requirement to be identified within the paragraph (2.3.4) where the requirement resides in the document.

3. Text (of Requirement).--This is the textual description of the identified originating requirement. This description should be concise, yet unambiguous.

4. Deviated/Waivered.--This records whether the requirement has been officially deviated and/or waived.

INPUT/UPDATE EXPLICIT REQUIREMENTS

DOCUMENT IDENTIFIER: _____
REFERENCE: ______________________

TEXT (OF REQUIREMENT):

________________________________________

HAS THIS REQUIREMENT BEEN DEVIATED/WAVERED (YES/NO)? _____

Fig. 7--Input/Update Screen for Explicit Originating Requirements.

Recording Implicit Requirements

During the analysis of explicit (documented) originating requirements (i.e., those contained in the Statement of Work), it is often determined that there are other requirements that, although not stated, are within the statement of originating requirements. These are called Implicit requirements. Implicit requirements must be traceable to explicit (documented) requirements.
Defining Implicit Requirements Identifier

It is necessary to define an identifier for the implicit requirements that are defined for the system. Since many systems could be defined to the database, there must be a way to associate which implicit requirements are applicable to which system. Production versions of the system will handle this automatically. This version requires the screen represented in Figure 8 to be used to define the implicit requirement identifier that is unique for every system defined to prototype.

ASSOCIATE AN IMPLICIT REQUIREMENT IDENTIFIER TO A DEVELOPMENT SYSTEM

SYSTEM IDENTIFIER: _____

IMPLICIT REQUIREMENT IDENTIFIER: _____

Fig. 8--Input/Update of association between Implicit Requirements Identifier and a Development System.

The following list describes each of the fields where data are requested in Figure 8.

1. **System Identifier**.--This is the logical identifier (acronym) that the system development team has assigned to the system under development.

2. **Implicit Requirement Identifier**.--This is the unique logical identifier chosen for the implicit requirements that is defined for the system under development.
Defining Implicit Requirements

All implicit requirements should be recorded to the prototype, along with a trace to the explicit requirements from which they were established. Figures 7 and 8 represent the Input/Update record and maintain all implicit requirements established during the analysis of originating requirements. The logical identifier for each implicit requirement must be defined to the system. This identifier is used in reports generated by the system.

INPUT/UPDATE IMPLICIT REQUIREMENT (FORM I)

IMPLICIT REQUIREMENT IDENTIFIER: 

REFERENCE: __________________________

DESCRIPTION: _____________________________________________

HAS THIS REQUIREMENT BEEN DEACTIVATED/IGNORED (YES/NO)? 

Fig. 9--Input/Update Screen for Implicit Requirements (Form I).

Both the screen shown above and the following screen must be used to enter the available information concerning implicit requirements. The following list describes each of the fields where data are requested in Figure 9.

1. Implicit Requirement Identifier.--This is a logical identifier chosen for the implicit requirements that is defined for the system under development.

2. Reference.--This is a unique identifier for the implicit requirement.
3. **Description.**--This is the textual description of the implicit requirement.

4. **Deactivated/Ignored.**--Answer Yes or No if requirements have been deactivated or ignored.

The following screen is the second screen used to enter the available information concerning implicit requirements.

**INPUT/UPDATE IMPLICIT REQUIREMENT (FORM II)**

**IMPLICIT REQUIREMENT IDENTIFIER:** ____

**REFERENCE:** __________________________

**ORIGINATOR:** __________________________

**DATE:** ________________________________

**RATIONALE:** __________________________

HAS THE CUSTOMER BEEN NOTIFIED OF THIS IN WRITING (YES/NO)?

IF SO, ENTER DOCUMENT IDENTIFIER: ____ REFERENCE: ____

IF NOT, WHY NOT?

Fig. 10--Input/Update Screens for Implicit Requirements (Form II).

The following list describes each of the fields where data are requested in Figure 10.

1. **Implicit Requirement Identifier.**--This is a logical identifier chosen for the implicit requirements that is defined for the system under development.

2. **Reference.**--This is a unique identifier for the implicit requirement.

3. **Originator (optional).**--This is the name of individual/organization that has identified the implicit requirement.
4. Date.--This is the date the implicit requirement was established (or entered). (DD-MMM-YY.)

5. Rationale (optional).--If offered, this is an explanation of the justification for the implicit requirement.

6. Customer Been Notified?.--This is a Yes or No question indicating if the customer has been notified of this requirement. A deviation or a waiver may be appropriate to submit to customer.

7. If so, Enter Document Identifier.--Either a memo, letter or official deviation or waiver was submitted to customer.

8. Reference.--Reference or document identifier that has been (or is) defined to the system, through the entry facility for Originating Requirements Documents facility.

9. If Not, Why?.--This explains why the customer was not notified of this requirement in writing and may reference telephone conversations, etc.

Furnishing Tracing for Implicit Requirements

Since implicit requirements are defined as those derived from documented sources (Originating Requirements), it is necessary to provide the tracing information to the database as shown in the screen in Figure 11.

The following list describes each of the fields where data are requested in Figure 11.

1. Implicit Requirement Identifier.--This is the unique identifier that was chosen for the implicit requirements defined for the development system.

2. Implicit Requirement Reference.--This is a unique identifier for the implicit requirement.

3. Source Requirement Document Identifier.--This is the unique logical name (acronym) that was chosen for the originating document when defined to the system.
4. **Source Requirement Document Reference.**—This is the unique identifier from the source requirement document.

**TRACE INFORMATION BETWEEN IMPLICIT REQUIREMENTS AND ORIGINATING SOURCE REQUIREMENTS**

**IMPLICIT REQUIREMENT IDENTIFIER:** _____________________

**IMPLICIT REQUIREMENT REFERENCE:** _____________________

| traces to: |

| V |

**SOURCE REQUIREMENT DOCUMENT IDENTIFIER:** ________________

**SOURCE REQUIREMENT DOCUMENT REFERENCE:** ________________

Fig. 11—Screen for tracing information between Implicit and Source Requirements.

**Recording Inexplicit Requirements**

During the analysis of Explicit (Documented) Originating Requirements (i.e., Statement of Work), the development team can institute new requirements which are not directly traceable to Explicit (Documented) Requirements. We have labeled this class of Originating Requirements as Inexplicit. All Inexplicit Requirements should be recorded to the prototype.

**Defining Inexplicit Requirements Identifier**

It is necessary to define an identifier for the inexplicit requirements that are intended to be defined for the system. Since many systems could be defined for the
system. Since many systems could be defined to the database, there must be a way to associate which inexplicit requirements are applicable to which system. Production versions of the system will handle this automatically. This (prototype) version requires the screen represented in Figure 12 to be used to define the inexplicit requirement identifier that is unique for every system defined to the prototype.

ASSOCIATE AN INEXPLICIT REQUIREMENT IDENTIFIER TO A DEVELOPMENT SYSTEM

SYSTEM IDENTIFIER: _____
INEXPLICIT REQUIREMENT IDENTIFIER: _____

Fig. 12--Input/Update of association between an Inexplicit Requirement Identifier and a Development System.

The following list describes each of the fields where data are requested in Figure 12.

1. System Identifier.--This is the unique logical identifier (acronym) that the system development team has assigned to the system under development.

2. Inexplict Requirement Identifier.--This is the unique logical identifier (acronym) chosen for the inexplicit requirements that is defined for the system under development.

Defining Inexplict Requirements

All inexplicit requirements should be defined to the system. Figures 13 and 14 represent the Input/Update Screens that are furnished to record and maintain all inexplicit requirements established during the analysis of originating
requirements. The logical identifier for all inexplicit requirements that is established for the development (sub) system must already be defined to the system.

**INPUT/UPDATE INEXPLICIT REQUIREMENTS (FORM I)**

**INEXPLICIT REQUIREMENTS IDENTIFIER:**

**REFERENCE:**

**DESCRIPTION:**

---

HAS THIS REQUIREMENT BEEN DEACTIVATED/IGNORED? ____ (YES/NO)

Fig. 13--Input/Update Screen for Inexplicit Requirements

The following list describes each of the fields where data are requested in Figure 13.

1. **Inexplicit Requirements Identifier.**--This is a logical identifier chosen for the system under development.

2. **Reference.**--This is a unique identifier for the Inexplicit Requirement.

3. **Description.**--This is the textual description of the inexplicit requirement.

4. **Deactivated/Ignored.**--Answer Yes or No if the requirements have been either deactivated or ignored.

The following list describes each of the fields where data are requested in Figure 14.

1. **Inexplicit Requirements Identifier.**--This is a logical identifier chosen for the system under development.

2. **Reference.**--This is a unique identifier for the Inexplicit Requirement.
3. **Originator (optional).**--This is the name of the individual/organization that has identified the inexplicit requirement.

4. **Date.**--This is the date the inexplicit requirement was established (or entered). (DD-MMM-YY.)

5. **Rationale.**--This is furnished to allow specification of any information regarding the justification for the requirement.

6. **Has Customer Been Notified In Writing?**--This is a self explanatory Yes or No Question.

7. **If So, Enter Document Identifier.**--This is the document identifier that has been entered to the system which references the document that was sent to the customer.

8. **Reference.**--This is the requirements reference within the document furnished to the customer.

9. **If No, Explanation.**--This explains why the customer was not notified in writing or references telephone conversations, etc.

**INPUT/UPDATE INEXPLICIT REQUIREMENTS (FORM II)**

| INEXPLICIT REQUIREMENTS IDENTIFIER: | __________________________ |
| REFERENCE: | __________________________ |
| ORIGINATOR: | __________________________ |
| DATE: | __________________________ |
| RATIONALE: | __________________________ |

**HAS THE CUSTOMER BEEN NOTIFIED OF THIS IN WRITING? ____**

**IF SO, ENTER: DOCUMENT IDENTIFIER: __________________**
**REFERENCE: __________________**

**IF NOT, WHY NOT?**

Fig. 14--Input/Update Inexplicit Requirements (Form II)
Furnished Reports Regarding Originating Requirements

A menu, represented by Figure 15, is displayed to the user upon requesting the Originating (source) Requirements Report Services Facility.

ORIGINATING (SOURCE) REQUIREMENTS REPORT SERVICES FACILITY

SELECT ONE OF THE FOLLOWING:

A--REPORT OF ALL DOCUMENTS DEFINED TO THE PROTOTYPE
B--REPORT OF ALL DOCUMENTS DEFINED FOR SPECIFIC SYSTEM
C--REPORT OF ALL EXPLICIT REQUIREMENTS FOR A SPECIFIC DOCUMENT
D--REPORT OF ALL IMPLICIT REQUIREMENTS FOR SPECIFIC (SUB) SYSTEMS
E--REPORT OF ALL INEXPLICIT REQUIREMENTS FOR SPECIFIC (SUB) SYSTEMS
F--REPORT SHOWING TRACEABILITY OF IMPLICIT REQUIREMENTS
END--EXITS

Fig. 15--Originating Requirements

Report of All Documents Defined to the Prototype

This report shows a sorted listing of all documents that have been defined to the database. See Appendix B for an example of this report.

Report of All Documents Defined for a Specific System

This report shows a sorted listing of all documents that have been defined to the database and associated to a specific (sub) system. See Appendix C for an example of this report.
Report of All Explicit Requirements for a Specific Document

This report shows a sorted list of all requirements that have been defined to the database for a specific document. See Appendix F for an example of this report.

Report of All Implicit Requirements for a Specific System

This report lists all implicit requirements established for a specific development system. For each implicit requirement the Description, Reference, Rationale, Originator and Date information is listed. See Appendix D for a sample of this report.

Report of All Inexplicit Requirements for Specific Systems

This report lists all Inexplicit Requirements for a specific development system. For each Inexplicit Requirement, the Reference, Description, Rationale, Originator, and Date information is listed. See Appendix E for a sample of this report.

Report Showing Traceability of Implicit Requirements

This report shows the traceability of all implicit requirements for a specific development system. See Appendix G for a sample of this report.
CHAPTER VI

WORKING REQUIREMENTS SPECIFICATIONS

Various systems analysis, requirements analysis and specifications techniques can be employed to take the originating requirements and develop a formal specification of system requirements, hardware requirements and software requirements. These formal specifications usually form the basis for the composition of a requirements document(s). We use the term "working specifications" to include all English narrative, diagrams (e.g., data flow, context, Warnier/Orr, etc.) and formal requirements language specifications which are used during requirements analysis. These specifications may at some point be baselined and put in Texas Instruments' archives. It is critical to trace originating requirements through the development of the working requirements specifications. The prototype provides a facility to record and describe what working requirements specifications have been developed, along with the traceability information to the originating requirements.

Log/Record Working Specifications Menu

A menu, represented by Figure 16, is displayed to the user upon entry of the Log/Record Working Specifications Facility.
REQUIREMENTS TRACEABILITY PROTOTYPE
LOG/RECORD WORKING SPECIFICATIONS MENU

SELECT ONE OF THE FOLLOWING:

A--RECORD WORKING SPECIFICATION INFORMATION
B--ASSOCIATE A WORKING SPECIFICATION WITH A DEVELOPMENT (SUB) SYSTEM
C--RECORD TRACING FOR WORKING SPECIFICATIONS
END--EXITS

Fig. 16--Log/Record Working Specifications Facility

Recording Working Requirements Specifications

It is necessary to record each identifiable working requirements specification to the system. The prototype provides an input/update screen, shown in Figure 17, for the purpose of recording and maintaining this information.

The following list describes each of the fields where data are requested in Figure 17.

1. Working Requirements Specification Identification.--This is a unique logical identifier (acronym) that the system development team assigns to the working requirement specification.

2. Specification Title.--This is the title associated with the working requirement specification.

3. Drawing Number.--This is the reference number of the specification in the drawing control library.

4. Revision.--This is a revision letter/number identifying a revision of the specification in the drawing control library.

5. Sheet.--This is the letter/number of the sheet with the specification document in the drawing control library.

6. Date.--This is the date the working requirement specification was established (or entered).
7. **Developer(s).**—This is the name(s) of the developer(s) of the working requirement specification.

8. **Description (Of Specification).**—This is the textual description of the working requirement specification.

INPUT/UPDATE WORKING REQUIREMENTS SPECIFICATIONS

WORKING REQUIREMENTS SPECIFICATION ID: __________________________

SPECIFICATION TITLE: ____________________________________________

DRAWING NUMBER: _______ REVISION: _______ SHEET: _______

DATE: ____________ (DD-MMM-YY)

DEVELOPER(S): ____________________________________________

DESCRIPTION (OF SPECIFICATION):

Fig. 17--Input/Update Screen to record and maintain Working Requirements and Specification Information.

Associate a Working Specification with a Development System

It is necessary to associate a working specification to a development system already defined to the database. More than one working specification may be defined for a particular development system/project.

The prototype provides the input/update screen shown in Figure 18 that ties one working specification to the development system. After that association is entered in the database, more working specifications can be tied to the system by repeating the procedure. Note that both the
system and the working specifications must already be defined to the prototype database (using the facilities described earlier in this document).

ASSOCIATE A WORKING SPECIFICATION TO A DEVELOPMENT SYSTEM
ENTER:
SYSTEM IDENTIFIER: _____
WORKING SPECIFICATION IDENTIFIER: _______

Fig. 18--Input/Update of association between Working Specifications.

The following list describes each of the fields where data are requested in Figure 18.

1. System Identifier.--This is a logical identifier (acronym) that the system development team has assigned to the system under development.

2. Working Specification Identifier.--This is a logical identifier (acronym) that the system development team assigns to the specification.

Record Tracing for Working Specifications

It is necessary to provide traceability between working specifications and/or (requirements) documents. The screen represented by Figure 19, is provided for that purpose.

The following list describes each of the fields where data are requested in Figure 19.

1. Working Specifications Identifier.--This is a logical name (acronym) that the system development team assigns to each working specification.
2. **Requirement/Specification Identifier.**—This is a logical name (acronym) assigned by the system development team.

3. **Requirement/Specification Reference.**—This is a unique number assigned to the document.

TRACE INFORMATION BETWEEN WORKING SPECIFICATIONS AND OTHER SPECIFICATIONS/REQUIREMENTS

WORKING SPECIFICATIONS IDENTIFIER: ______

| traces to: |
| V |

REQUIREMENT/SPECIFICATION IDENTIFIER: ______

REQUIREMENT/SPECIFICATION REFERENCE: ______

Fig. 19--Input/Update of Tracing Information between Working Specifications and other Requirement/Specifications.
CHAPTER VII

THE REQUIREMENTS SPECIFICATION DOCUMENT(S)

A requirements specification document is generally developed during the requirements definition and analysis phase of the life-cycle.

It is necessary to define the document(s) to the database along with the entire specification in order to receive maximum benefit from the system. To provide traceability back to the originating requirements, references must be furnished for each specification, tying it back to its source requirements (see Figure 20).

REQUIREMENTS TRACEABILITY PROTOTYPE
LOG/REPORT REQUIREMENTS DOCUMENT MENU

SELECT ONE OF THE FOLLOWING:

A--DEFINING REQUIREMENTS SPECIFICATION DOCUMENT(S)
B--ASSOCIATE A REQUIREMENTS SPECIFICATION DOCUMENT TO A DEVELOPMENT (SUB) SYSTEM
C--RECORD REQUIREMENTS SPECIFICATION
D--RECORD TRACEABILITY BETWEEN (SOURCE) REQUIREMENT AND REQUIREMENT SPECIFICATION
E--REQUIREMENTS DOCUMENT REPORT SERVICES FACILITY
END--EXITS

Fig. 20--Log/Report Requirements Document Menu
Log/Report Requirements Document Menu

A menu, represented by Figure 20, is displayed to the user upon entry to the Log/Report Requirements Document Facility.

Defining the Specification Document

It is necessary to define the Specification Document to the system. This is accomplished by selecting the Defining Requirements Specification Document field on the Log/Report Requirements Document Menu (see Figure 20 above). This selection produces the screen shown in Figure 21.

**INPUT/UPDATE REQUIREMENTS SPECIFICATION DOCUMENT**

<table>
<thead>
<tr>
<th>DOCUMENT IDENTIFIER:</th>
<th>DOCUMENT NUMBER:</th>
<th>VOLUME:</th>
<th>REVISION:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TITLE:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<p>| DATE PUBLISHED: |</p>
<table>
<thead>
<tr>
<th>(DD-MMM-YY)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

| DEVELOPER(S): |
|               |
|               |

<table>
<thead>
<tr>
<th>DESCRIPTION:</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Fig. 21--Input/Update to define Requirements Specifications Documents.

The following list describes each of the fields where data are requested in Figure 21.
1. **Document Identifier**.--This is a logical name (acronym) that the system development team assigns to each originating requirements document for use by the system. This logical name can be five (5) characters in length and is used in all of the reports generated by the Requirements Traceability Prototype where originating requirements are referenced. A logical name should be chosen so that it can easily be associated with the document it identifies, for example, if it is a Statement of Work document, the Identification chosen should be "SOW." The use of a logical name (acronym) was required to enhance readability of reports generated by the system.

2. **Document Number**.--This is a unique number assigned to the document.

3. **Volume**.--This is the volume number of the document.

4. **Revision**.--This is a revision letter/number identifying a revision of the document.

5. **Title**.--This is the full title associated with the document.

6. **Date Published**.--This is the most recent publication date associated with the document. (DD-MMM-YY.)

7. **Developer(s)**.--This is the developer(s) or author(s) of the document.

8. **Description**.--This is a brief description of the assignment.

**Tying a Requirements Specification to the System**

It is necessary to associate each requirements specification document to a development (sub) system already defined to the prototype. More than one requirements specification document may be applicable to a development system/project.
The system provides the input/update screen shown in Figure 22 that ties one document to the development system. After that association is entered in the database, more documents can be tied to the system by repeating the procedure. Note that both the system and the relevant requirements specification document must already be defined to the Requirements Traceability Prototype database (using the facilities described earlier in this document).

ASSOCIATE REQUIREMENTS SPECIFICATION DOCUMENT TO A DEVELOPMENT (SUB) SYSTEM

ENTER:

SYSTEM IDENTIFIER: 

DOCUMENT IDENTIFIER: 

Fig. 22--Input/Update of association between Relevant Requirements Specification Document and a Development System.

The following list describes each of the fields where data are requested in Figure 22.

1. System Identifier.--This is a logical identifier (acronym) that the system development team has assigned to the system under development. A database integrity constraint is applied to this field (after the input of the System Identifier) to assure that the system identifier entered is defined to the Requirements Traceability Prototype (using the screen to Input/Update System Information).

2. Document Identifier.--This is a logical identifier (acronym) that the system development team assigns to the document. An integrity constraint is applied to this field to insure that the Document Identifier entered has already been defined to the database.
Recording Requirements Specifications

After the requirements specification document is defined to the Requirements Traceability Prototype, each requirement specification can be recorded/entered to the system. It is necessary to identify and extract each Singular requirement in the document. These are entered into the system using the following form (see Figure 23). This form is accessed by selecting the Record Requirements Specification field on the Log/Report Requirements Document Menu.

INPUT/UPDATE REQUIREMENT SPECIFICATION

REQUIREMENT SPECIFICATION IDENTIFIER: ______
REFERENCE: ______________
DESCRIPTION (TEXT): ______________________________

Fig. 23--Input/Update to record Requirements Specification.

The following list describes each of the fields where data are requested in Figure 23.

1. Requirement Specification Identifier.--This is a logical name (acronym) that the system development team assigns to each originating requirements document.

2. Reference.--This is unique identifier for each singular requirement identified in the originating requirements document.

3. Description.--This is a brief description of the requirement.
Providing Traceability Back to Source Document

Since many or all source requirements should be addressed in the requirements specification document, it is necessary to provide the tracing information to the database.

TRACE INFORMATION BETWEEN SOURCE REQUIREMENT AND REQUIREMENT SPECIFICATION

REQUIREMENT SPECIFICATION DOCUMENT IDENTIFIER: _____

REQUIREMENT SPECIFICATION REFERENCE: _____

| traces to: |

V

SOURCE REQUIREMENT DOCUMENT IDENTIFIER: _____

SOURCE REQUIREMENT REFERENCE: _____

Fig. 24--Input/Update of tracing information between Source Requirement and Requirement Specifications Documents.

The following list describes each of the fields where data are requested in Figure 24.

1. Requirement Specification Document Identifier.--This is a logical name (acronym) assigned by the system development team.

2. Requirement Specification Reference.--This is a unique number assigned to the document.

3. Source Requirement Document Identifier.--This is a logical name (acronym) that the system development team assigns to each originating requirements document.

4. Source Requirement Reference.--This is a unique number assigned to the document.
Requirements Document Report
Services Facility

A menu, represented by Figure 25, is displayed to the user upon entry to the Requirements Document Report Services Facility.

REQUIREMENTS TRACEABILITY PROTOTYPE

REQUIREMENTS DOCUMENT REPORT
SERVICES FACILITY

SELECT ONE OF THE FOLLOWING:

A--TRACEABILITY REPORT GIVEN A DOCUMENT IDENTIFICATION
B--REQUIREMENTS REPORT FOR A SPECIFIC DOCUMENT
C--UNTRACED REQUIREMENTS FROM A SPECIFIC DOCUMENT
D--UNTRACED REQUIREMENTS TO A SPECIFIC DOCUMENT
E--TRACEABILITY REPORT BETWEEN TWO SPECIFIC DOCUMENTS
END--EXITS

Fig. 25--Report Services Facility Menu

There are several reports regarding requirement specifications and the traceability to originating requirements.

Traceability Report Given a Document Identifier

This report shows for each requirement within a document, where that requirement traces. Upon selection of this report (from the menu), the system prompts the user for a Document Identification. After the Document Identification is furnished, Requirements Traceability Prototype generates the report. See Appendix H for a sample of this report.
### Requirements Report for a Specific Document

This report shows a sorted listing of all requirements for a specific document currently contained in the database. Upon selection of this report (from the menu), the system prompts the user for a Document Identification. After the Document Identification is furnished, the system generates the report. See Appendix F for a sample of this report.

### Untraced Requirements from a Specific Document

This report shows a sorted list of all requirements from a specific document which do not trace to any other sources (requirements). Upon selection of this report (from the menu), the system prompts the user for a Document Identification. After the Document Identification is furnished, the system generates the report. See Appendix I for a sample of this report.

### Untraced Requirements to a Specific Document

This report shows a sorted list of all requirements from a specific document which are not traced to (from any other sources). In other words, this report furnishes a list of all requirements that are not addressed by any other requirements and/or specifications. Upon selection of this report (from the menu), the system prompts the user for a Document Identification. After the Document Identification
is furnished, the system generates the report. See Appendix J for a sample of this paper.

**Traceability Report Between Two Specific Documents**

This report shows requirements traceability between two documents. It indicates for each requirement in the first document where it is traceable to requirements in the second document. See Appendix K for a sample of the report.
CHAPTER VIII

EXTERNAL CONSIDERATIONS FOR
THE PROTOTYPE

The major advantages of the Requirements Traceability Prototype are the simplicity of its design and its ease of use while still maintaining traceable project visibility and control. The selection of the supporting tools and procedures for the construction of this prototype made it possible for much of the work to be done by a software technician, eliminating the need for a software engineer to be involved.

The prototype was carried out as a project to demonstrate the ability and feasibility of requirements traceability. There were several considerations that entered into the way in which this project was carried out. Because of customer demands this project had to be implemented in the span of two months. This meant that careful consideration was given to the tools that were focused on the problem. There was not time to spend on many iterations of paper and pencil design strategies. The tools used in the specification cycle had to be able to be used in the implementation cycle. For this reason we chose INGRES as the focal point for the specification, design, and implementation of the
prototype. There were many factors that led to that decision, but the main issues were (1) ease of use (training and usage), (2) flexibility of the Relational Model, (3) forms concept of the INGRES application development system, and (4) power of the query language and Report Writer.
FUTURE CONSIDERATIONS FOR THE REQUIREMENTS

TRACEABILITY PROTOTYPE

Future Considerations

The point should be once again emphasized that this was a demonstration. This demonstration has now opened doors for more work to be done in the area of the development of an Integrated Support Software Environment as described in Chapter II (the STARS effort in the Department of Defense). In order to build on a system where one of the central focuses is quality, the ability to measure conformance to requirements is a central issue. The ability to store and retrieve requirements and to trace the logical nature of these requirements will be a necessity. The Requirements Traceability Prototype provides this basic foundation.

There will be gains in the productivity of the software developer, by the evolution of an Integrated Support Software Environment. This is a necessity in the real-time embedded computer systems community. The real gains will come when we can take advantage of research and development being done in the area of Knowledge Based Expert Systems (1). This use of Artificial Intelligence has great promise for the next generation of software and hardware development.
Conclusions

The Requirements Traceability Prototype will provide a flexible base on which to build such a system. We have a working model of traceability, one in which we are satisfied at the current time. Using this demo as a set of specifications, we can now move forward in the area of design and implementation of a production system that will meet the needs of the Integrated Support Software Environment.
CHAPTER BIBLIOGRAPHY

APPENDICES
APPENDIX A

ALL SYSTEMS DEFINED

DEVELOPMENT (SUB) SYSTEMS DEFINED TO PROTOTYPE

SYSTEM TITLE: VHSIC Hardware Description Language
SYSTEM IDENTIFIER: VHDL
CONTRACT NUMBER: 666666-99 CONTRACT DATE: 11-NOV-83
CONTRACTING AGENCY Rome Air Development Center
SYSTEM DESCRIPTION:
Develop a Hardware Description Language that can be used to describe VHSIC hardware components

SYSTEM TITLE: Ada Language System / Navy
SYSTEM IDENTIFIER: ALS/N
CONTRACT NUMBER: 123-45689 CONTRACT DATE: 11-NOV-83
CONTRACTING AGENCY Navy Ocean Systems Center
SYSTEM DESCRIPTION:
Develop an environment using the Army ALS for the purpose of providing an Ada programming development environment

SYSTEM TITLE: Ada Interactive Monitor
SYSTEM IDENTIFIER: AIM
CONTRACT NUMBER: 987-34343 CONTRACT DATE: 11-JUN-83
CONTRACTING AGENCY Navy Ocean Systems Center
SYSTEM DESCRIPTION:
Develop and study the interfaces between the Army ALS and the USAF AIM.
APPENDIX B

REPORT OF ALL DOCUMENTS DEFINED TO THE PROTOTYPE

<table>
<thead>
<tr>
<th>DOCUMENT IDENTIFIER</th>
<th>DATE</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQSP</td>
<td>6-12-83</td>
<td>REQUIREMENTS SPECIFICATION VHDL</td>
</tr>
<tr>
<td>REP</td>
<td>4-12-82</td>
<td>REQUEST FOR PROPOSAL FOR VHDL</td>
</tr>
<tr>
<td>SOW</td>
<td>3-13-82</td>
<td>STATEMENT OF WORK FOR VHDL</td>
</tr>
</tbody>
</table>

NOV. 15, 1983
APPENDIX C

DOCUMENTS DEFINED FOR A SPECIFIC DEVELOPMENT SYSTEM

NOV. 15, 1983

REPORT OF ALL DOCUMENTS DEFINED FOR THE VHSIC HARDWARE DESCRIPTION LANGUAGE PROJECT

Document Identifier: SOW Date: 13-NOV-83
Document number: 342423423 Volume: _1 Revision: 1
Title: VHSIC Hardware Description Language SOW
Description:
Statement of Work for the VHDL project.

Document Identifier: RFP Date: 13-NOV-83
Document number: 324646333 Volume: _1 Revision: 1
Title: VHSIC Hardware Description Language RFP
Description:
Request for Proposal for the VHDL Project.

...
APPENDIX D

REPORT OF ALL IMPLICIT REQUIREMENTS
VHSIC HARDWARE DESIGN LANGUAGE

IMPLICIT REQUIREMENTS
VHDL SHALL HAVE PREDEFINED VIEWS.

A DESIGN BODY SHALL PERMIT THE DECLARATION OF ADDITIONAL USER DEFINED VIEWS.

THE USER SHALL BE ABLE TO CHANGE AN ASPECT OF A VARIANT.

THE USER SHALL BE ABLE TO ADD AN ASPECT TO A VARIANT.

THE USER SHALL BE ABLE TO DELETE AN ASPECT FROM A VARIANT.

A REFERENTIALLY DEFINED ASPECT MUST NOT CONTAIN ANY INFORMATION, OTHER THAN THE REFERENCE.
APPENDIX E

REPORT OF ALL INEXPLICIT REQUIREMENTS
VHSIC HARDWARE DESIGN LANGUAGE

INEXPLICIT REQUIREMENTS
Document ID: TEAM  Requirement Reference: 2.01.01*02
Description:
A DESIGN ENTITY SHALL PERMIT THE DESCRIPTION OF ONLY ONE INTERFACE.

Document ID: TEAM  Requirement Reference: 2.01.01*03
Description:
A DESIGN ENTITY SHALL PERMIT THE DESCRIPTION OF ONLY ONE DESIGN BODY.

Document ID: TEAM  Requirement Reference: 2.01.01*04
Description:
A DESIGN BODY SHALL PERMIT THE DESCRIPTION OF ONE OR MORE VARIANT DESIGN DESCRIPTIONS.

Document ID: TEAM  Requirement Reference: 2.01.01*05
Description:
A DESIGN BODY SHALL PERMIT THE DESCRIPTION OF MULTIPLE VIEWS, WHICH ARE EACH UNIQUELY NAMED, OF A DESIGN.

Document ID: TEAM  Requirement Reference: 2.01.01*06
Description:
A VARIANT SHALL PERMIT ONLY ONE ASPECT FROM ANY SINGLE VIEW.

Document ID: TEAM  Requirement Reference: 2.01.01*07
Description:
A VARIANT MAY CONTAIN ASPECTS FROM ALL VIEWS.
APPENDIX F

REQUIREMENTS REPORT FOR A SPECIFIC DOCUMENT
VHSIC HARDWARE DESIGN LANGUAGE

REQUIREMENTS LISTING

ORIGINATING (SOURCE) DOCUMENT:

DESCRIPTION LANGUAGE
Document ID: DODLR
Date Published: 18-nov-1983
Number: 1
Volume: 1
Revision: 1
Originator(s): INSTITUTE FOR DEFENSE ANALYSIS

Document Description:

THIS DOCUMENT CLEARLY SPECIFIES THE REQUIREMENTS FOR THE VHSIC HARDWARE DESIGN LANGUAGE (VHDL). THIS DOCUMENT, TOGETHER WITH THE CONTRACTUAL STATEMENT OF WORK, FORM A BINDING CONTROL DOCUMENTATION SYSTEM FOR THE VHDL DEVELOPMENT EFFORT.
Each data object shall be declared explicitly.

No restriction shall be imposed on user defined types unless it is imposed on all types.

The constraints that characterize subtypes shall include range, precision, scale, and index ranges.

The value of a subtype constraint may be specified when the data object is declared.

VHDL shall be extendable to allow additional primitive types.

The VHDL shall have strong type checking between operators, operands, and the corresponding results.
APPENDIX G

REPORT SHOWING TRACEABILITY OF IMPLICIT REQUIREMENTS
### VHSIC HARDWARE DESIGN LANGUAGE

#### TRACEABILITY OF IMPLICIT REQUIREMENTS

<table>
<thead>
<tr>
<th>Implicit Requirements</th>
<th>Trace to other Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier: Reference:</td>
<td>Identifier: Reference:</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>05</em>A</td>
<td>TEAM 2.01.01*05</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>05</em>B</td>
<td>TEAM 2.01.01*05</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>08</em>A</td>
<td>TEAM 2.01.01*08</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>08</em>B</td>
<td>TEAM 2.01.01*08</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>08</em>C</td>
<td>TEAM 2.01.01*08</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>09</em>A</td>
<td>TEAM 2.01.01*09</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>09</em>B</td>
<td>TEAM 2.01.01*09</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>09</em>C</td>
<td>TEAM 2.01.01*09</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>09</em>D</td>
<td>TEAM 2.01.01*09</td>
</tr>
<tr>
<td>DERIV 2.01.03<em>01</em>03*A</td>
<td>IBMIR 2.01.03<em>01</em>03</td>
</tr>
<tr>
<td>DERIV 2.01.03<em>01</em>03*B</td>
<td>IBMIR 2.01.03<em>01</em>03</td>
</tr>
<tr>
<td>DERIV 2.01.03<em>01</em>03*C</td>
<td>IBMIR 2.01.03<em>01</em>03</td>
</tr>
<tr>
<td>DERIV 2.01.03<em>01</em>03*D</td>
<td>IBMIR 2.01.03<em>01</em>03</td>
</tr>
<tr>
<td>DERIV 2.01.03<em>01</em>03*E</td>
<td>IBMIR 2.01.03<em>01</em>03</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>A</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>B</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>C</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>D</td>
<td>TEAM 2.01.05*02</td>
</tr>
</tbody>
</table>
APPENDIX H

TRACEABILITY REPORT GIVEN A DOCUMENT IDENTIFIER
### Traceability of Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBMLR 2.01.01*01</td>
<td>DODLR 2.0</td>
</tr>
<tr>
<td>IBMLR 2.01.02*02</td>
<td>DODLR 2.0</td>
</tr>
<tr>
<td>IBMLR 2.01.02*04</td>
<td>DODLR 2.0</td>
</tr>
<tr>
<td>IBMLR 2.01.02*04</td>
<td>DODLR 5.0</td>
</tr>
<tr>
<td>IBMLR 2.01.02*05</td>
<td>DODLR 2.01</td>
</tr>
<tr>
<td>IBMLR 2.01.02*06</td>
<td>DODLR 2.01</td>
</tr>
<tr>
<td>IBMLR 2.01.03*01</td>
<td>DODLR 3.03.03</td>
</tr>
<tr>
<td>IBMLR 2.01.03<em>01</em>A</td>
<td>DODLR 3.03.03</td>
</tr>
<tr>
<td>IBMLR 2.01.03<em>01</em>B</td>
<td>DODLR 3.03.03</td>
</tr>
<tr>
<td>IBMLR 2.01.03<em>01</em>C</td>
<td>DODLR 3.03.03</td>
</tr>
<tr>
<td>IBMLR 2.01.03<em>01</em>D</td>
<td>DODLR 3.03.07</td>
</tr>
<tr>
<td>IBMLR 2.01.03<em>01</em>E</td>
<td>DODLR 3.03.03</td>
</tr>
<tr>
<td>IBMLR 2.01.03.01*03</td>
<td>DODLR 3.02</td>
</tr>
<tr>
<td>IBMLR 2.02.01*01</td>
<td>DODLR 3.02</td>
</tr>
<tr>
<td>IBMLR 2.02.01*02</td>
<td>DODLR 3.03.02</td>
</tr>
<tr>
<td>IBMLR 2.02.01*03</td>
<td>DODLR 3.02.02</td>
</tr>
<tr>
<td>IBMLR 2.02.01*04</td>
<td>DODLR 3.02</td>
</tr>
<tr>
<td>IBMLR 2.02.01*05</td>
<td>DODLR 3.02</td>
</tr>
</tbody>
</table>
APPENDIX I

REPORT SHOWING UNTRACED REQUIREMENTS FROM A GIVEN DOCUMENT
Requirements from IBMLR that do not trace back to the DODLR document

Identifier: Reference:

<table>
<thead>
<tr>
<th>IBMLR</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBMLR</td>
<td>1.01</td>
</tr>
<tr>
<td>IBMLR</td>
<td>1.02</td>
</tr>
<tr>
<td>IBMLR</td>
<td>1.03</td>
</tr>
<tr>
<td>IBMLR</td>
<td>1.04</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.0</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.01</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.01.01</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.01.02</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.01.03</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.01.03.01</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.01.04</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.01.05</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.02</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.02.01</td>
</tr>
<tr>
<td>IBMLR</td>
<td>2.02.01.03.03</td>
</tr>
</tbody>
</table>
APPENDIX J

REPORT SHOWING UNTRACED REQUIREMENTS TO A GIVEN DOCUMENT
UNTRACED
REQUIREMENTS
TO
LANGUAGE REQUIREMENTS FOR VHSIC HARDWARE DESCRIPTION LANGUAGE

Requirements in DODLR document that are not traced to from IBMLR document
Identifier: Reference:

<table>
<thead>
<tr>
<th>DODLR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td></td>
</tr>
<tr>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>1.04.02</td>
<td></td>
</tr>
<tr>
<td>1.04.04</td>
<td></td>
</tr>
<tr>
<td>1.04.05</td>
<td></td>
</tr>
<tr>
<td>1.04.07</td>
<td></td>
</tr>
<tr>
<td>1.05</td>
<td></td>
</tr>
<tr>
<td>1.06</td>
<td></td>
</tr>
<tr>
<td>1.07</td>
<td></td>
</tr>
<tr>
<td>2.02</td>
<td></td>
</tr>
<tr>
<td>2.03</td>
<td></td>
</tr>
<tr>
<td>2.04</td>
<td></td>
</tr>
<tr>
<td>2.05</td>
<td></td>
</tr>
<tr>
<td>2.06</td>
<td></td>
</tr>
<tr>
<td>2.07</td>
<td></td>
</tr>
<tr>
<td>3.0</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX K

REPORT SHOWING TRACEABILITY BETWEEN TWO GIVEN DOCUMENTS
TRACEABILITY OF REQUIREMENTS BETWEEN IMPLICIT REQUIREMENTS AND INEXPlicit REQUIREMENTS

<table>
<thead>
<tr>
<th>Requirements first document</th>
<th>Requirements second document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier: Reference:</td>
<td>Identifier: Reference:</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>05</em>A</td>
<td>TEAM 2.01.01*05</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>05</em>B</td>
<td>TEAM 2.01.01*05</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>08</em>A</td>
<td>TEAM 2.01.01*08</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>08</em>B</td>
<td>TEAM 2.01.01*08</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>08</em>C</td>
<td>TEAM 2.01.01*08</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>09</em>A</td>
<td>TEAM 2.01.01*09</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>09</em>B</td>
<td>TEAM 2.01.01*09</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>09</em>C</td>
<td>TEAM 2.01.01*09</td>
</tr>
<tr>
<td>DERIV 2.01.01<em>09</em>D</td>
<td>TEAM 2.01.01*09</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>A</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>B</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>C</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>D</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>E</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 2.01.05<em>02</em>F</td>
<td>TEAM 2.01.05*02</td>
</tr>
<tr>
<td>DERIV 3.01.01<em>02</em>A</td>
<td>TEAM 3.01.01*02</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY

Books

Crosby, Phil, Quality is Free, New York, Menton Books, 1980.


Articles


Government Documents