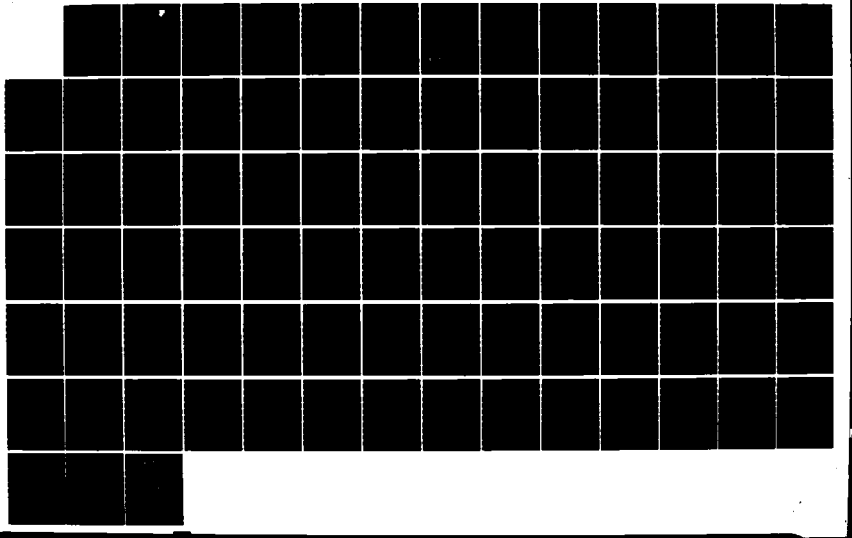
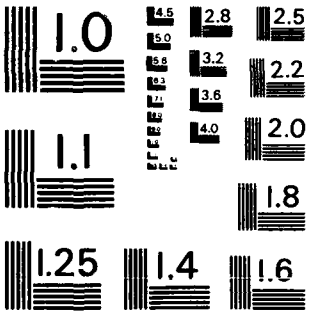
AD-A162 655 SPECIFICATION TECHNOLOGY GUIDELINES(U) BOEING AEROSPACE 1/1
CO KENT WA D R ADDLEMAN ET AL. AUG 85 RADC-TR-85-128
F30602-84-C-0073

UNCLASSIFIED F/G 9/2 NL





(12)

RADC-TR-85-128
Final Technical Report
August 1985

SPECIFICATION TECHNOLOGY GUIDELINES

Boeing Aerospace Company

David R. Addleman, Margaret J. Davis and P. Edward Presson

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-85-128 has been reviewed and is approved for publication.

APPROVED: William E. Rzepka

WILLIAM E. RZEPKA
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, Jr.
Technical Director
Command and Control Division

FOR THE COMMANDER: Richard W. Pouliot

RICHARD W. POULIOT
Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A162655

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS N/A
2a SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5 MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-85-128
6a NAME OF PERFORMING ORGANIZATION Boeing Aerospace Company 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c ADDRESS (City, State, and ZIP Code) Seattle WA 98124	7b ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b OFFICE SYMBOL (If applicable) COEE	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-84-C-0073
8c ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F; PROJECT NO: 5581; TASK NO: 22; WORK UNIT ACCESSION NO: 13

11 TITLE (Include Security Classification)
SPECIFICATION TECHNOLOGY GUIDELINES

12 PERSONAL AUTHOR(S)
David R. Addleman, Margaret J. Davis, Edward P. Presson

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM 2Mar84 TO 1Mar85	August 1985	

16 SUPPLEMENTARY NOTATION
N/A

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
09	02		Software Specification Methodology Guidelines Software Specification Tools Software Design Methodology Software Requirements Methodology

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

The purpose of this research was to provide the Air Force system and software development manager with an uncomplicated set of guidelines for matching existing methods and tools to the needs of specific development projects. The Specification Technology Guidelines effort was divided into five major tasks.

The first task studied software requirements in six Air Force mission areas. In the performance of this task information was gathered regarding programming characteristics, software development requirements, and the support environments of typical application software. Within each mission area, differences in application and development of software most affecting methodology selection were given more attention. All Air Force software efforts were deemed to fall into 18 generic software categories.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
William E. Rzepka	(315) 330-4063	RADC (COEE)

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

PREFACE

This document is the final technical report, CDRL Item A002, that describes the results of the five tasks involved in developing the specification technology guidelines and incorporating them into a Specification Technology Guidebook, CDRL Item A003. This report was prepared in accordance with the statement of work, contract F30602-84-C-0073. It summarizes the work done for the Rome Air Development Center by Boeing Aerospace Company in Kent, Washington.

In task 1 of this contract, the system developments in which the Air Force uses computers and software were studied and organized into generic categories. As part of this task, a telephone survey was conducted of Air Force personnel associated with software developments to review current Air Force practice and experience, if any, with specification and design methodologies.

In task 2, existing system requirements, software requirements, and software design methodologies were studied and evaluated with respect to the category of system and life cycle phase to which they were applicable.

In task 3, a set of guidelines were developed for the software manager to use in selecting a coherent specification methodology and support tools to meet specification needs of system requirements, software requirements, and software design activities.

In task 4, these table-driven selection guidelines were documented in a guidebook. The modular design that the tables provide simplify the use of the guidebook and allow it to be expanded to include new methodologies and techniques. The guidebook also describes the various methodologies and tools studied in task 2.

In task 5, the guidelines were applied to three example problems for C^3I software development projects. A primary consideration imposed on each example was compatibility with the Ada* programming language. The other considerations used for system requirements and design of the C^3I problems were derived from actual requirements set forth in C^3I RFP's, and working knowledge of the requirements for C^3I software and system projects gained by Boeing Aerospace engineers during the last decade.

*Ada is a trademark of the U.S. Department of Defense (Ada Joint Program Office).

TABLE OF CONTENTS

TABLE OF CONTENTS - continued

LIST OF FIGURES

ABBREVIATIONS

CADSAT	military version of PSL/PSA
C^3I	Command, Control, Communications & Intelligence
DCDS	Distributed Computing Design System
DoD	Department of Defense
DSSD	Data Structured System Design
HDM	Hierarchical Design Methodology
IORL	Input-Output Requirements Language
JSD	Jackson System Design
MIL-STD	military standard
OSL	overall significance level
PAISLey	Process-Oriented, Applicative, Interpretable Specification Language
pdl	program design language
PSL/PSA	Problem Statement Language/Problem Statement Analyzer
RADC	Rome Air Development Center
SADT	Structured Analysis and Design Technique
SARA	System Architect's Apprentice
SCR	Software Cost Reduction project
SREM	Software Requirements Engineering Methodology
VDM	Vienna Development Method

1. PROJECT OVERVIEW

1.1. Background

During the 1970's, many techniques and tools were developed to support system and software development processes. Most development concentrated on programming and testing activities, but a *mini-proliferation* of specification and analysis tools occurred for supporting the *front end* life cycle activities: system requirements analysis, software requirements analysis, and software design. Complex specification methodologies appeared (e.g., that were based on data flow, control flow, and finite state machine modeling techniques, among others) and incorporated specialized analysis tools (e.g., formal languages, graphical descriptions, static analyzers, etc.). Much has been written about their capabilities, problem domains, and relative degrees of sophistication and success. However, no single methodology or technique is universally applicable to the specification of all problem environments.

Rather than being enlightened, some users are confused by this proliferation of methodologies and techniques and are no closer to understanding which methodology best fits project requirements. This situation has been further complicated with the introduction of the Ada programming language. Users must understand which of the methods and techniques result in designs taking maximum advantage of the software engineering principles which Ada directly supports and implements.

In such an atmosphere, the technical manager is confronted with many claims, counter-claims, comparisons, and evaluations of existing specification methods, most of which are uncorroborated. No effective focus has existed for aiding software development managers during selection of the specification methodology or tool best suited to their problem. The Specification Technology Guidelines effort was conceived to provide the needed focus and to organize methodologies and tools information.

1.2. Project Summary

The purpose of this research was to provide the Air Force system and software development manager with an uncomplicated set of guidelines for matching existing methods and tools to the needs of specific development projects. The Specification Technology Guidelines effort was divided into five major tasks, described briefly in the following paragraphs (for detailed descriptions refer to sections 2.0, 3.0, 4.0, 5.0, and 6.0 of this report).

The first task studied software requirements in six Air Force mission areas. In the performance of this task information was gathered regarding programming

characteristics, software development requirements, and the support environments of typical application software. Within each mission area, differences in application and development of software most affecting methodology selection were given more attention. All Air Force software efforts were deemed to fall into 18 generic software categories. Section 2.0 of this report describes these categories in detail. Summaries of the Air Force missions are contained in appendices A through F of the guidebook, with each appendix listing the software functions for that mission, by software categories.

The second task studied existing system requirements and software design methodologies. Mature methodologies were identified (in this case "mature" refers to methodologies that can be used by other than their developing organization) and analyzed to determine which software categories they satisfied and the life cycle phase(s) to which they applied. Section 3.0 of this report describes this task in detail.

The third task used information from the first two tasks in constructing guidelines that aid a project manager in selecting methodologies and supporting tools for a particular software category and life cycle phase. The guidelines support and technically complement system requirements, software requirements, and software design specification activities. Task 3 is described in section 4.0 of this report.

The fourth task documented the results of the first three tasks in a Specification Technology Guidebook (hereafter called the Guidebook). The Guidebook was written for use by technical managers of system and software development projects and presents step-by-step procedures and explanations for using the guidelines developed in task 3. The Guidebook also describes how a particular methodology and its products will satisfy Air Force specification standards. Task 4 is detailed in section 5.0 of this report.

The fifth task developed examples detailing use of the Specification Technology Guidebook. The three examples were drawn from typical C^3I software development projects. The first example approaches methodology selection from the viewpoint of requirements capabilities and depicts development of C^3I system software for interactive displays. The second example approaches methodology selection from the viewpoint of design capabilities and depicts an airborne software system for use on an unnamed special purpose computer. The third example approaches methodology selection from the viewpoint of a highly-skilled manager who knows the capabilities he wants in a methodology. Features required in the selected methodologies of all three examples included: data base management (needed for storing software tools and life-cycle products by phase), user friendliness, and compatibility of life-cycle products. In addition, the selections were constrained to be compatible with the Ada

programming language. Task 5 is described in detail in section 6.0 of this report.

1.3. Scope of Effort

The Guidebook effort was constrained to select among existing requirements and design methodologies and tools. The Guidebook was written for Air Force managers who perform daily planning and execution of system and software acquisition and development. As such, it was prepared for users with a general technical background, who are not necessarily familiar with the intricacies of requirements and design technology.

1.4. Brief Description of Guidebook

The purpose of the Guidebook is to aid the technical project manager in selecting specification and design methodologies for a given project and life cycle phase. Three different table-driven selection paths are provided, along with example usage of each. In addition, the Guidebook provides paragraph models that will aid in preparing future Statements of Work that stipulate the use of particular methodologies. The structure of the Guidebook is illustrated in figure 1-1.

The organization of the Guidebook permits easy access since:

(a) A novice user can start with the introduction,

(b) A more experienced user who desires to select a requirements methodology for a project can turn to section 2 of the Guidebook and follow the instructions describing how to accomplish that selection (this procedure is *Path 1* in the Guidebook),

(c) A more experienced user who desires to select a design methodology for a project can turn to section 2 of the Guidebook and follow the instructions describing how to accomplish that selection (this procedure is *Path 2* in the Guidebook),

(d) A very experienced user who knows the features he wants in a methodology, regardless of the specification phase, can turn to section 2 and follow the instructions describing how to select a methodology using a set of desired capabilities (this procedure is *Path 3* in the Guidebook), or

(e) Any user desiring information on a particular methodology or tool can turn to the descriptions in section 4 of the Guidebook.

The following paragraphs describe the contents of the Guidebook on a section-by-section basis.

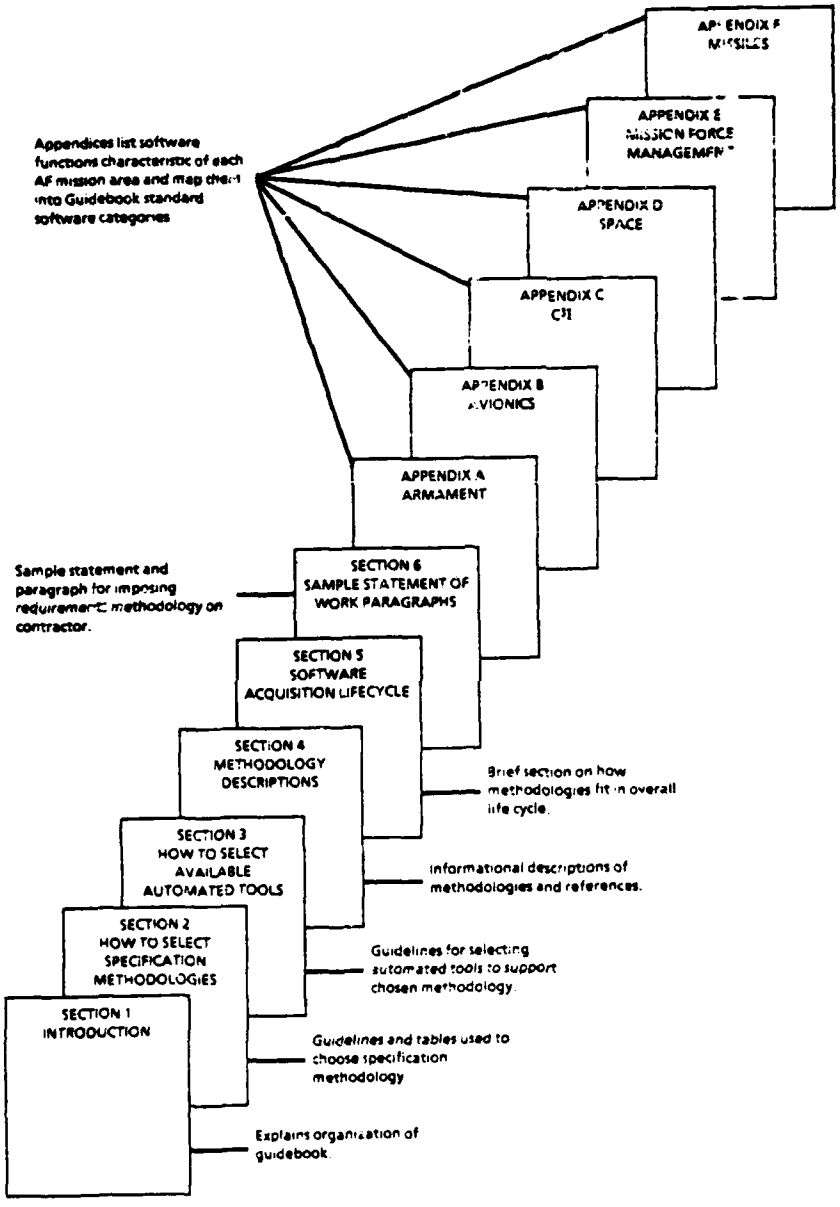


Figure 1-1: Guidebook Organization

The first section introduces the Guidebook, discusses its purpose, and describes its organization. There is a brief discussion of the need for appropriate specification technology and how, in general, the Guidebook should be used during selection.

The second section presents guidelines for using the table-driven selection paths (i.e., Paths 1, 2, and 3). Our table-driven approach has two advantages:

(1) It is easy to follow and use without extensive training or reading, and

(2) the tables are constructed for easy expansion to incorporate new methodologies and techniques as they mature.

The third section provides guidelines for selecting automated tools that will support candidate methodologies.

The fourth section describes methodologies and automated tool sets.

The fifth section briefly describes pertinent software acquisition life cycles. It discusses Air Force Phased Acquisition objectives and their relation to specification technologies used in the Guidebook.

The sixth section provides model paragraphs for use in preparing future Statements of Work (SOWs) which will direct the use of specification technologies in the development of Air Force Systems.

Appendices A through F summarize major Air Force missions. Major functions within each mission are described and information gathered regarding programming characteristics, software development requirements, and the support environments of typical application software. Within each mission area, differences in application and development of software most affecting methodology selection were given more attention. Mission software efforts were decomposed into 18 generic software categories (see figure 1-2). Tables in the appendices list the software functions for a mission, by software categories. These tables aid the user in fitting his proposed software development into one of the Guidebook categories.

2. SUMMARY OF TASK 1

2.1. Categorize Air Force System Developments

The first task studied Air Force missions and organized system developments into broad, generic categories relating computers and software development to the missions. These system categories relate software functions to software categories for later usage in methodology selection. The requirements for specification technology were defined for each software category in terms of the

SOFTWARE CATEGORIES TABLE

Category	Characteristics	Description
(1) Arithmetic Based	Data oriented	Programs that do primarily arithmetic (e.g., payroll and wind tunnel data analysis) operations. A real-time environment is not necessary. Small, throwaway programs for preliminary analysis also fit in this category.
(2) Event control	Control-oriented processing	Does real-time processing of data resulting from external events. An example might be a computer program that processes telemetry data.
(3) Process control	Control-oriented processing	Receives data from an external source and issues commands to that source to control its actions based on the received data.
(4) Procedure control	Complex processing	Controls other software; for example, an operating system that controls execution of time-shared and batch computer programs.
(5) Navigation	Complex processing	Does computations and modeling to compute information required to guide an airplane from point of origin to destination.
(6) Flight Dynamics	Control-dominated complex processing	Uses the functions computed by navigation software and augmented by control theory to control the entire flight of an aircraft.

Figure 1-2 Software Categories Table (part 1 of 3)

continued		
Category	Characteristics	Description
(7) Orbital Dynamics	Control-dominated complex processing	Resembles navigation and flight dynamics software, but has the additional complexity required by orbital navigation, such as a more complex reference system and the inclusion of gravitational effects of other heavenly bodies.
(8) Message processing	Data-dominated complex processing	Handles input and output *messages, processing the* text or information contained therein.
(9) Diagnostic S/W	Data-oriented processing	Used to detect and isolate hardware errors in the computer in which it resides or in other hardware that can communicate with that computer.
(10) Sensor and signal processing	Control-dominated complex processing	Similar to that of message *processing, except* that it requires greater processing, analyzing, and transforming the input into a usable data processing format.
(11) Simulation	Complex, depending on entity being simulated	Used to simulate an environment, mission situation, other hardware, and inputs from these to enable a more realistic evaluation of a computer program or a piece of hardware.
(12) Database management	Data-oriented processing	Manages the storage and access of (typically large) groups of data. Such software can also often prepare reports in user-defined formats, based on the contents of the database.

Figure 1-2 Software Categories Table (part 2 of 3)

continued		
Category	Characteristics	Description
(13) Data Acquisition	Control-dominated complex processing	Receives information in real-time and stores it in some form suitable for later processing; for example, software that receives data from a space probe and files it for later analysis.
(14) Data presentation	Data-oriented	Formats and transforms data, as necessary, for convenient and understandable displays for humans. Typically, such displays would be for some screen presentation.
(15) Decision and planning aids	Data-dominated complex processing	Uses artificial intelligence techniques to provide an expert system to evaluate data and provide additional information and consideration for decision and policymakers.
(16) Pattern and image processing	Data-dominated complex processing	Used for computer image generation and processing. Such software may analyze terrain data and generate images based on stored data.
(17) Computer system Software	Data-oriented	Provides services to operational computer programs (i.e., problem-oriented).
(18) Software development tools	Data-oriented	Provides services to aid in the development of software (e.g., compilers, assemblers, static and dynamic analyzers).

Figure 1-2 Software Categories Table (part 3 of 3)

life cycle development activities of system requirements analysis, software requirements analysis, and software design. These specification technology requirements are built into the tables presented in Guidebook section 2.0. The requirements were based on considerations of available technology and the formats prescribed by current Air Force specification standards (MIL-STD-490, Specification Practices, 30 Oct 68) for system and software requirements and software design documentation. At the kick-off meeting held at RADC in April 1984, it was agreed to support MIL-STD-SDS (to be released as MIL-STD-2167).

We suggested starting with the standard software categories defined in the Software Test Handbook produced under RADC contract number F30602-82-C-0050, and relating these categories to the major Air Force missions. These categorical relationships were successfully used in developing the Software Test Handbook.

To assure that the results of the previous handbook were applicable for our research, we undertook an informal, updating survey of current Air Force mission usage of requirements and design methodologies for software development. Survey results are detailed in section 2.2. The survey showed that no basic structural changes would be required in adopting the approach taken by the Software Test Handbook, although we separated the Missile/Space mission description into independent appendices, resulting in six appendices instead of five. Additionally, minor changes were made in mission descriptions to account for our focus on software development requirements instead of software testing requirements.

2.2. Survey of Air Force Requirement Specification Problems

This survey of Air Force personnel (both civilian and military) determined general and specific problems in preparation and analysis of requirement specifications, and examined methodology and tool experience.

Since Boeing proposed using existing descriptions of Air Force missions, and the software functions and software categories used in the Software Test Handbook (RADC contract F30602-82-C-0050), the survey determined which changes to this informational base were needed.

2.2.1. Survey Participants

The initial interview list comprised Air Force personnel contacted during the Software Test Handbook survey. Each contact was asked to recommend other individuals with pertinent experience in software development and use of

specification methodologies and tools. A total of 32 contacts were made, and 25 extended interviews held.

Most participants were military and civilian Air Force personnel in all six Air Force mission areas, and included staff and project personnel at various levels of responsibility. Additionally, we talked with personnel from DoD, Aerospace Corporation, and MITRE Corporation.

2.2.2. Survey Results

The relatively informal survey was structured so that each participant was asked the same general questions. Survey questions fell into five groups, and are discussed in the following sections. Of course, the discussions often ranged widely afield, depending on the experience and interests of each participant; these chats often provided additional insights.

2.2.2.1. Question 1: Major Problem Areas

One survey participant said the major problem with requirement specifications is that "we still don't know how to write an effective one." Most of those surveyed did not respond so strongly, but felt there were major problems in the specification of systems and software. The descriptions of major problem areas tended to fall into three categories:

(1) The ambiguity of specifications - Requirement specifications are written by the contractor to be understood by Air Force acquisition organizations; as a result, some users had trouble understanding the language of the specification and felt they couldn't intelligently comment on it.

(2) Difficulty in generating a B-level specification from an A-level specification - In a study of the Space Division mission, an excellent B5 spec was the single most critical element in a successful system development.

(3) The time consuming tracing of requirements from concept definition through system test - This task is deemed enormously important and is currently being accomplished by hand.

2.2.2.2. Question 2: Unique Problems of Specific Air Force Missions

A surprising survey result was that almost every participant said that they had no unique requirement specification problems characteristic of their area.

Several participants identified unique characteristics of their product areas, such as the unit cost of armaments, the high reliability required of spacecraft, and the many human interfaces in C^3 systems. Few of these problems were truly

problems of specifying requirements; the problems were, instead, specific cases of the general problems encountered by all.

2.2.2.3. Question 3: Experience with Methodologies and Tools

Most of those surveyed did not use any methodology or tool as part of their standard practice. Few had used more than one methodology or tool. None considered that their use of any methodology or tool had been a success, nor did anyone know of a success story from others.

PSL/PSA (and CADSAT, its military equivalent) seemed the best known tool. Second most frequently mentioned was Docwriter (now called TEMSE). However, only a fraction of the participants had direct experience with either.

2.2.2.4. Question 4: Requirements Tracking and Most Critical Phases

We asked each survey participant about the need for requirements tracking, and to identify the life cycle phases when this need was most critical. Almost everyone felt that there was a strong need to be able to track requirements. The two most frequently mentioned phases were (1) the transition from user need to system specification, and (2) the transition from system specification to a B-level spec. A significant fraction declined to identify any phase as being most critical.

One iconoclast said that requirements tracing was no real problem; and that it was simply a bookkeeping job. He further stated that all the methodologies and tools were attempting to solve the wrong problem. His estimation was that these tools addressed only five percent of the total task. He felt the major problem was the creative, inventive analysis required when going from user needs to system spec, from system spec to segment spec, and from segment spec to B-level spec. His opinions were not seconded elsewhere.

2.2.2.5. Question 5: Rapid Prototyping

Most of those surveyed agreed that a rapid prototyping capability sounded attractive, but said they did not know how to accomplish it and that they doubted any usable capability would be available soon. Several said that it was entirely the responsibility of the contractor.

2.2.3. Effect of Survey on Specification Technology Guidebook

The survey provided no information that required changing the standard software categories used in the Software Test Handbook. It did, however, provide insight into the problems faced by the Air Force in the specification and use of requirements; and that insight helped us produce a more usable and

relevant guidebook.

3. SUMMARY OF TASK 2

3.1. Evaluate Current Methodologies

We identified, categorized, and characterized specification technology methodologies and automated tools that were suitable for practical application to Air Force mission software development projects. Although the number of methodologies described in the computer science and software engineering literature was considerable, we selected 12 methodologies for evaluation. The considerations we used in their selection are described in section 3.2 below.

Each of the 12 methodologies was categorized by its primary philosophy of requirements and design specification. The requirements analysis categories (i.e., the way a requirements methodology models the system to be produced) are:

1. Flow-oriented - models the system in terms of control or data flow.
2. Finite-state machine - models state of system and transitions between states.
3. Object-oriented - models the entities (objects, concepts, processes) in the system, including relationships among them and events and mechanisms that change them.

The design categories are:

1. Decomposition - guided either by function, control, or data hierarchical relationships.
2. Encapsulation - producing either data or process abstractions.
3. Data Structured - guided by inherent relationships among input and output data items.
4. Programming Calculus - guided by assertions about the effect a series of statements must produce.

We developed a standard format for characterizing each methodology. The format is a composite of characteristics found in three different sources: (1) the *Methodman I* report, (2) a study done by Hughes Aircraft Company entitled *Reusable Software Implementation Technology Reviews*, and (3) Boeing internal studies of software engineering methodologies and tools. We used a similar form to characterize automated tool sets. Appendix A of this report contains an annotated copy of both formats.

Boeing software engineers with relevant expertise in specification technologies reviewed and concurred with our evaluations of methodologies and tools.

3.2. Criteria for Inclusion

The major criterion for inclusion of a methodology was "maturity such that the methodology could be used by organizations other than their originators." This criterion encourages inclusion of all the common methodologies that have been repeatedly field tested and their results well documented; it excluded in-house methodologies developed by a company for its own use, in accordance with its own standards. Thus, we evaluated a group of methodologies that are as representative of *state of the practice* of specification technology as possible.

The emphasis on maturity immediately eliminated from consideration those methodologies which are:

a. theoretical descriptions never implemented,

b. academic exercises developed to prove a particular technique is feasible, and

c. developments in progress as knowledge engineering (expert system) research projects.[1]

The methodologies we characterized are listed below.

DSSD	Data Structured Systems Design
HDM	Hierarchical Development Method
SADT	Structured Analysis and Design Technique
SA/SD	Structured Analysis and Structured Design (Realtime Yourdon)
SCR	Software Cost Reduction - Navy
SREM	Software Requirements Engineering Methodology
VDM	Vienna Development Method
DCDS	Distributed Computing Design System
JSD	Jackson System Design
PAISLey	Process-oriented, Applicative, Interpretable Specification Language
SARA	System ARchitect's Apprentice
USE	User Software Engineering Methodology.

DSSD, SADT, SA/SD, and SREM are well-known, well-exercised, basic methodologies. They were developed from research performed in the early 1970's in response to the *crisis in software*. Many DoD contractors use "customized" methodologies having strange names, but most of these methodologies are, in fact, closely related to the four methodologies above. Before a contract officer

1. The field of knowledge engineering is itself immature.

rejects a methodology proposed by a company, the question might first be asked: "On which methodology is this one based?" It may prove acceptable as is, or the company may accept a recommendation to use its "parent" methodology in its place.

HDM and VDM are representative of specification technologies based on formal specification languages.[2] They have been successfully used outside their developing organizations on significant projects.

DCDS is the methodology being developed as an extension of SREM to distributed systems design.

SCR and JSD are representative of technologies based on the characteristics of abstraction and encapsulation. SCR has seen use by the Navy and Bell Labs. JSD is commercially available in England.

PAISLey is representative of a trend to *operational* (prototyping) requirements specification methodologies.[3] PAISLey is not truly mature in the sense that it has been used by any organization other than its originator. However, the methodology was developed expressly for embedded systems. For development of less than life-critical software, the benefits should outweigh the risk of using it.

SARA and USE are representative of technologies developed in university settings that are full-scale methodologies, rather than feasibility demonstrations of single ideas.

Some methodologies were excluded from evaluation because the information we could gather on them was too meager. This is particularly a problem with methodologies developed outside the United States, where language and time barriers complicate our acquiring information on them.

The past year has seen the release and publication of material about methodologies billing themselves as *Ada compatible* or *object-oriented* (implying compatibility with Ada). In some cases, these announcements relate to modifications of existing methodologies and can be considered as additional information about them.[4] The basic description of these methodologies is not appreciably altered by such modifications, except that now it can include text about Ada compatibility. In other cases, the announcements describe methods that can be incorporated into existing methodologies to increase the Ada compatibility of produced designs.

---

2. Affirm, CLEAR, Ordinary, SDM, Z, SLAN-4, and LARCH are other methodologies in this category.

3. GIST is another methodology in this category.

4. ANNA is an exception to either case, in that it is an entirely new methodology designed for Ada compatibility. Unfortunately, material on ANNA was not readily avail-

3.3. Tool Evaluations

There are literally thousands of commercially available software packages (software tools) that claim to assist requirements analysis and design. It was impractical to review them all during this effort. We concentrated, instead, on reviewing and classifying tool sets that support particular methodologies through the specification phases of software development. Our reason for doing this was simple: tools tailored specifically to the methodology are much more effective than generic tools. And finally, most methodologies already have a set of software tools developed for them (e.g., REVS is tailored specifically for SREM).

When methodologies (DSSD, HDM, SREM, DCDS, PAISLey, SARA, USE) had tool sets developed specifically for them, we described both the methodology and tool set features in the methodology description section of the Guidebook. We did not describe these tool sets a second time in the software tools section.

We evaluated one tool set (TAGS[5]) for use with SADT and three tool sets (ARGUS, EXCELERATOR, PROMOD) for use with SA/SD. TAGS is certainly the best tool set based on SADT. There are at least four more tool sets available for SA/SD that we did not review. We confined our evaluations to tool sets for which we found software engineers having hands-on experience with the product (e.g., characteristics like usability require actual experience to be properly evaluated).

We treated PSL/PSA as a generic automated tool set useful both in project support and documentation. PSL/PSA can be used with several methodologies (i.e., particularly helpful to those performing data flow analysis).

4. SUMMARY OF TASK 3

4.1. Define Selection Guidelines

We developed guidelines to aid a project manager in selecting a final candidate methodology and supporting tools for a particular software category and life cycle phase. This selection process matches the methodology's abstract modeling capability to the conceptual needs of the software category and life cycle phase. In doing this we have fulfilled the *explicit* requirements of the Statement of Work (SOW) for the effort. But, the guidelines we developed also fulfill the *implicit* requirements of the SOW in terms of practicality of a selected

able until February 1985, too late for inclusion in the Specification Technology Guidebook.

5. Also known as IORL.

methodology to a project. Our guidelines match the relative power of the methodology and support tools to the relative significance of the development project. This extends the original intent of the SOW, and effectively tailors final selection to the project.

The guidelines were set up to hide the identity of candidate methodologies until selection is completed. This blind selection process mitigates the effects of selector bias and acquaints the selector with methodologies he might otherwise never consider.

The selection guidelines provide three different paths. The first two paths require the user to rate the significance of the project and to determine its software category. Once project significance and software category are determined, the user can choose Path 1 and base selection on the requirements specification phase; or Path 2 and base selection on the design specification phase. Path 3, provided for the experienced manager, bypasses significance rating and software categorization, and allows the manager to base selection on individual methodology capabilities needed for a project.

4.2. Significance Level Determination

A common sense notion in software development is that a methodology for a project should be appropriate to the *size* of the project. Discussions in the literature about the differences in *programming-in-the-small* versus *programming-in-the-large* are elaborations of this notion.

Our significance level concept is a refinement of the notion of project *size*. We developed this concept because projected raw count of source code statements, in itself, is not a sufficient discriminator for choosing one methodology over another. For example, formal, verifiable, complex methodologies normally considered more appropriate to large projects might be recommended for small projects involving life-critical decisions.

The significance level concept examines the software development project from three viewpoints: project considerations, software considerations, and quality considerations. Under project considerations, cost, criticality, and schedule, measure the importance of funding agency attributes to the project. Under software considerations, complexity, development formality, and software utility, measure the conceptual effort required by the project, and thus measure the significance of the project to its developers. Under quality considerations, reliability, correctness, maintainability, and verifiability measure the importance end-users attribute to the project.

Quality considerations exist other than the four listed above. We incorporated only those considerations that were independent of software category. Two

reasons support this decision.

(1) Simplification - Since the guidelines describe a manual system, additional quality considerations would have meant the addition of worksheets (as many as 18), one for each software category.

(2) Clarity - The relationship of the unused quality attributes to individual software categories is not straightforward.

4.2.1. Considerations

The relationship of each consideration to significance level is explained below.

Project Considerations

(1) Cost - The significance of a project increases in direct proportion to relaxation of constraints on cost. A low budget with tight constraints indicates a lower significance than a relatively unconstrained budget.

(2) Criticality - The significance of a project rises in direct proportion to the criticality of the assignment. A project in which flight crew safety is involved is more significant than one whose failure would be of nuisance impact to its users.

(3) Schedule - The significance of a project increases in direct proportion to relaxation of constraints on scheduling. A project with tight controls on scheduling is less significant than one whose schedule can slip in order to get the software 'right'.

Software Considerations

(4) Complexity - The significance of a project rises in direct proportion to the complexity of the solution. A difficult problem whose solution is hard to validate is more significant than a problem whose solution is straight-forward and easy to check.

(5) Development Formality - The significance of a project rises in direct proportion to the desired level of contractor controls. The stronger the control (and the more formal the review of interim results), the more significant the project.

(6) Software Utility - The significance of a project rises in direct proportion to the utility of the application. Software developed as a one-shot feasibility demonstration is less significant than software developed to provide real-time support to a C^3I task.

(7) Reliability - The significance of a project rises in direct proportion to level of reliability needed. Software that should respond correctly to nominal conditions is less significant than software which must have its faults[6] removed as soon as they occur.

(8) Correctness - The significance of a project rises in direct proportion to level of correctness needed. From a specification point of view, *level of correctness* is a measure of how completely the software or its design satisfies project requirements and constraints. Software that is considered acceptable when its operation provides the functionality needed even though it does not meet other constraints (such as a user-friendly interface) is less significant than software whose design must be formally validated against the requirements specification.

(9) Maintainability - The significance of a project rises in direct proportion to the level of maintainability needed. Software that is not expected to be maintained is of less significance than software expressly developed so that the extent of changes are optimally localized.

(10) Verifiability - The significance of a project rises in direct proportion to the level of verifiability needed. Software whose documentation is to be casually maintained in its source code is of less significance than software whose documentation is complete (includes requirements through source code documents) and always up-to-date.

4.2.2. Characteristics of Each Significance Level

A project of significance level 0 has the following characteristics:

6. A fault is an undesirable response to anomalous conditions.

- Low Budget, emphasis on minimum cost
- No criticality assignment
- Tight Schedule
- Straight-forward solution; easy to checkout
- Few defined requirements; informal development;
  used locally
- One-shot; Prototype; Test Software; Demonstration Software
- Respond correctly to nominal conditions
- Functionality met; constraints ignored
  No maintenance expected
- Documentation in source code

Test generators, conversion table programs, and trade study simulations are examples of projects that would have significance level 0.

A project of significance level 1 has the following characteristics:

- Normal cost constraints
- Nuisance Impact
- Some schedule constraints
- Moderate Complexity
- Normal to Strong Contractor Controls
- Ground Based Software; Data Reduction;
  Mission Prep Software
- Faults corrected periodically;
  temporary workarounds provided
- Functionality and constraints met
- Predict impact of changes
- Source code documentation updated

Editors, compilers, mission and environmental simulators are examples of projects that would have significance level 1.

A project of significance level 2 has the following characteristics:

- Some Cost Flexibility
- Mission Impact
- Normal Schedule Constraints
- Greater Complexity
- Strong Contractural Controls; Formal Reviews
- Realtime Avionics, C^3 and C^3I software
- Faults removed ASAP
- Implementation validated against design specification
- Impact of changes somewhat localized
- Full complement of documentation; design documentation updated too

AWACS, ALCM, PMALS, C^3I, and Avionics mission planning are examples of projects that would have significance level 2.

A project of significance level 3 has the following characteristics:

- Cost not predominant factor; relatively unconstrained
- Nuclear, flight crew safety
- Additional fault detect requirements will not impact schedule
- Difficult Problem; Complex Solution; Hard to Validate
- Generally Contracted Rigid Controls Over Development
- Highly Critical Applications; Possible Catastrophic Results
- No faults
- Design validated against requirements specification
- Extent of changes optimally localized
- Requirements through source code documentation always up-to-date

Nuclear control and life-critical software are examples of projects that would have significance level 3.

4.3. Methodology Power Rating

We rated thirty individual capabilities for each methodology. Six capabilities were useful for requirements specification:

state modeling	object modeling
data flow modeling	timing performance specification
control flow modeling	accuracy performance specification

Fourteen were useful for design specification:

functional decomposition	module interface definition
data decomposition	formal verification
control decomposition	configuration management
data abstraction	completeness analysis
process abstraction	consistency analysis
data base definition	Ada compatibility
concurrency/synchronization specification	notation for code behavior

Ten were independent of life cycle phase:

prototyping	validation
test plan generation	usability
automated tools available	maturity
traceability	training/experience level
transition between phases	MIL-STD documentation

Each capability was rated on a scale of 0 to 3, where 0 is no support and 3 most support. The actual definitions of these numbers for a specific capability is found in Appendix B. In general, the factors that determine the relative power of one methodology over another are:

(a) formality of notation,

(b) complexity of specification produced,

(c) rigor of mathematical foundation, and

(d) degree of automated support.

4.4. Fitting Methodology to Phase, Category, and Significance

Paths 1 and 2 of the guidelines lead to methodology selection based on life-cycle phase, software category, and significance level. Path 1 differs from Path 2 by life-cycle phase; path 1 is concerned with requirements specification and path 2 with design specification. Methodologies are fitted to software categories by tables that list desired capabilities by software category. The tables permit matching methodology concepts desired by a user to concepts needed for a particular category. Methodologies are related to project significance by tables comparing them against a fictional *ideal* methodology (one that provides only the capabilities desired, at exactly the level of support wanted). We assumed a uniform level of support for all capabilities desired; that is, each capability should provide the same level of support as the overall significance rating of the project.

Path 3 of the guidelines permits the experienced manager to choose the set of capabilities he desires in a methodology. It also permits him to select a non-uniform level of support for the chosen capabilities.

5. SUMMARY OF TASK 4 - BUILD THE GUIDEBOOK

The Specification Technology Guidebook, which documents the guidelines developed in the Task 2, was designed using the Software Test Handbook (RADC-TR-84-53, Vol II) as a model. Test Handbook organization was reviewed for its applicability to the Guidebook, and for its usefulness to technical managers of system and software development projects. We found the basic structure of the Test Handbook to be satisfactory for the Specification Technology Guidebook. The organization is shown in Figure 1-1.

The design of the selection paths and the tables that support them was a very complex and time-consuming process. Our goal was to design a table-driven selection methodology that would include all the important considerations and that could be easily used. The design originally proposed omitted life cycle phases and relative significance of the project as considerations for selection criteria. We decided any selection criteria would prove inadequate if it merely indicated methodology capabilities without addressing how well those capabilities were supported.

The second design, produced during the contract, included all important considerations, but required the user to make many (up to 50) small arithmetic calculations. We decided, in the interests of user friendliness and simplification, to design tables with the scoring precalculated. Accomplishing this required over 1700 computations, but our third design successfully removed most of the drudgery from guidelines usage.

An additional benefit of the table-driven selection approach is that the tables can be easily modified to reflect new capabilities for existing specification and design technologies, as they are developed.

The future penalty for the precalculated simplicity of the selection process is that more work will be required to modify or revise the methodology score tables. For each methodology added or updated, someone must compute new scores for software category, life cycle phase, and significance level Instructions for adding new methodologies and computing scores for the tables are in Section 7.0 of this report.

The Air Force Mission appendices from the Software Test Handbook were revised and updated for inclusion in the Specification Test Guidelines. The most significant change was replacement of the appendix combining the Missile/Space missions; current Air Force organization is better reflected by the Missile mission (Appendix D) and the Space mission (Appendix F).

6. SUMMARY OF TASK 5 - METHODOLOGIES FOR C^{3}I SYSTEMS

In this task, the guidelines developed in Tasks 1, 2. and 3 were used to select software requirements and software design methodologies to form the *front-end* of an environment especially designed to support the development of a C^{3}I system. Selected tools, especially those that address software design activities, are required to be compatible with the Ada Programming Language.

The survey of C^{3}I systems summarized in Appendix C of the guidebook, and our experience with a variety of ground-based and airborne C^{3}I systems, convinced us that we couldn't fully demonstrate usage for this mission with a single example. The complexity and variety of the many C^{3}I systems, requiring different combinations of software functions, makes it difficult for a single set of methodologies to suffice. Therefore, we chose two C^{3}I systems to serve as examples. Additionally, we decided to demonstrate use of all three selection paths.

A ground-based interactive display system is used to demonstrate Path 1, which allows choosing a methodology based on the needs of the requirements phase. An air-borne system targeted for a special purpose computer is used to demonstrate Path 2, which allows choosing a methodology based on the needs of the design phase. Our third example demonstrates how an experienced project manager can select a methodology based on specific capabilities, without referencing the actual project for which he wants that methodology.

7. INSTRUCTIONS FOR MAINTENANCE OF GUIDEBOOK

Since the guidelines are extensible, it is possible to update the Guidebook to: (a) add methodologies, (b) reflect capability enhancements in existing methodologies, (c) change (by addition or deletion) the list of capabilities against which a methodology is scored, and (d) reflect any changes in the relationships of software categories to desired capabilities.

Section 7.1 describes how to compute the scores for a methodology. The scores are a function of the software category, the path (life cycle phase), and the significance level of the project. (Scoring a methodology generates 18*2*4=144 values.) Section 7.2 explains what changes are necessary if the list of capabilities rated (presently 30) is expanded.

7.1. Scoring a Methodology

The process described below shows how to update the existing scores for a previously evaluated methodology or to add scores for a methodology never before evaluated. An update would become necessary when a revised version of the methodology or its tool set is released. A methodology can be added as information becomes available for rating its capabilities, as long as it is usable by organizations other than its developers.

Scoring a methodology can proceed only after its individual capabilities have been rated. Appendix B details the rating system on which the present score tables are based. The appendix also contains a table (figure B-1) listing capability ratings for the methodologies evaluated in the Guidebook. The table was used to compute the Path 1 and 2 scores given in the Guidebook; and is identical to Guidebook figure 2-22. This figure should be updated if the ratings for a methodology are revised or if a new methodology is rated.

Figure 7-1 is a two page worksheet that organizes the calculations for a single software category. Figure 7-2 is a completed worksheet. In this example, the software category used is 5 and the methodology is SREM. Down the left side of the first page of the worksheet is a list of the 30 capabilities used to rate methodologies. The next column *RATING* has space for listing the individual ratings of those capabilities. We used the capabilities ratings shown in figure B-1 of appendix B to calculate the methodology scores used in the guidebook.

The third column *DESIRED* has space for marking the capabilities which are desired for the software category. The capabilities are divided into three groups: requirements, design, and universal. Notice that the *DESIRED* column is already marked for the universal capabilities group. The scoring process assumes that all universal capabilities are desirable. Figure 7-3 is a matrix of requirements capabilities versus software category with *x*'s marking the desired

Methodology Scoring Worksheet - page 1

Software Category _____ Methodology _____

CAPABILITY	RATING	DESIRED	VALUE	
REQUIREMENTS				
state modeling				COUNT(R)=
data flow modeling				
control flow modeling				
object modeling				SUM(R)=
timing performance spec				
accuracy performance spec				
DESIGN				
functional decomposition				
data decomposition				
control decomposition				
data abstraction				
process abstraction				
data base definition				COUNT(D)=
concurrency/synchronicity				
module interface definition				
formal verification				
configuration management				
completeness analysis				SUM(D)=
consistency analysis				
Ada compatibility				
code behavior notation				
UNIVERSAL				
prototyping		X		
test plan generation		X		
automated tool available		X		COUNT(U)= 10
traceability		X		
transistion between phases		X		
validation		X		
usability		X		
maturity		X		SUM(U)=
training/experience level		X		
MIL-STD documentation		X		

Figure 7-1: Methodology Scoring Worksheet (part 1 of 2)

Methodology Scoring Worksheet - page 2

Software Category _____ Methodology _____

SUM(1) = SUM(R) + SUM(D) + SUM(U) =

SUM(2) = SUM(D) + SUM(U) =

COUNT(1) = COUNT(R) + COUNT(D) + COUNT(U) =

COUNT(2) = COUNT(D) + COUNT(U) =

PATH 1

OSL=0: MS(1,0) = SUM(1) =

OSL=1: MS(1,1) = SUM(1) - COUNT(1) =

OSL=2: MS(1,2) = SUM(1) - 2*COUNT(1) =

OSL=3: MS(1,3) = SUM(1) - 3*COUNT(1) =

PATH 2

OSL=0: MS(2,0) = SUM(2) =

OSL=1: MS(2,1) = SUM(2) - COUNT(2) =

OSL=2: MS(2,2) = SUM(2) - 2*COUNT(2) =

OSL=3: MS(2,3) = SUM(2) - 3*COUNT(2) =

Figure 7-1: Methodology Scoring Worksheet (part 2 of 2)

Methodology Scoring Worksheet - page 1

Software Category 5 Methodology SREM

CAPABILITY	RATING	DESIRED	VALUE	
REQUIREMENTS				
state modeling	3		0	COUNT(R)=
data flow modeling	1	x	1	5
control flow modeling	1	x	1	
object modeling	1	x	1	SUM(R)=
timing performance spec	2	x	2	7
accuracy performance spec	2	x	2	
DESIGN				
functional decomposition	1	x	1	
data decomposition	1		0	
control decomposition	1		0	
data abstraction	0		0	
process abstraction	0		0	
data base definition	3		0	COUNT(D)=
concurrency/synchronicity	1	x	1	
module interface definition	2	x	2	9
formal verification	2	x	2	
configuration management	0	x	0	
completeness analysis	3	x	3	SUM(D)=
consistency analysis	3	x	3	
Ada compatibility	2	x	2	17
code behavior notation	3	x	3	
UNIVERSAL				
prototyping	2	X	2	
test plan generation	0	X	0	
automated tool available	3	X	3	COUNT(U)= 10
traceability	3	X	3	
transistion between phases	3	X	3	
validation	3	X	3	
usability	1	X	1	
maturity	3	X	3	SUM(U)=
training/experience level	3	X	3	23
MIL-STD documentation	2	X	2	

Figure 7-2: Example Use of Methodology Scoring Worksheet
(part 1 of 2)

METHODOLOGY SCORING WORKSHEET - page 2

Software Category 5 Methodology SREM

SUM(1) = SUM(R) + SUM(D) + SUM(U) = 7 + 17 + 23 = 47

SUM(2) = SUM(D) + SUM(U) = 17 + 23 = 40

COUNT(1) = COUNT(R) + COUNT(D) + COUNT(U) = 5 + 9 + 10 = 24

COUNT(2) = COUNT(D) + COUNT(U) = 9 + 10 = 19

PATH 1

OSL=0: MS(1,0) = SUM(1) = 47

OSL=1: MS(1,1) = SUM(1) - COUNT(1) = 47 - 24 = 23

OSL=2: MS(1,2) = SUM(1) - 2*COUNT(1) = 47 - 2*24 = -1

OSL=3: MS(1,3) = SUM(1) - 3*COUNT(1) = 47 - 3*24 = -25

PATH 2

OSL=0: MS(2,0) = SUM(2) = 40

OSL=1: MS(2,1) = SUM(2) - COUNT(2) = 40 - 19 = 21

OSL=2: MS(2,2) = SUM(2) - 2*COUNT(2) = 40 - 2*19 = 2

OSL=3: MS(2,3) = SUM(2) - 3*COUNT(2) = 40 - 3*19 = -17

Figure 7-2: Example Use of Methodology Scoring Worksheet
(part 2 of 2)

Desirable Requirements Phase Capabilities

		Modeling Techniques				Performance Specification	
		State	Flow		Object	Timing	Accuracy
			Data	Control			
SOFTWARE CATEGORY	1		x		x		x
	2			x		x	
	3			x		x	
	4		x	x		x	
	5		x	x	x	x	x
	6	x	x	x	x	x	
	7	x	x	x	x	x	x
	8		x	x	x	x	
	9		x		x		x
	10	x	x	x	x	x	x
	11	x	x	x	x	x	
	12		x		x		x
	13	x	x	x	x	x	x
	14		x		x		x
	15	x	x	x	x		x
	16		x	x	x		x
	17		x	x	x	x	
	18		x		x		

Figure 7-3: Requirements Capabilities versus Software Category

entries. Figure 7-4 is a matrix of design capabilities versus software category with *x*'s marking the desired entries. Across the software category row, the capability entries marked as desired in figures 7-3 and 7-4 are exactly those capabilities to be marked in the *DESIRED* column on the worksheet. The set of desired capabilities describes the *ideal* methodology for the software category.

The fourth column *VALUE* has space for recording the ratings of the capabilities marked as desired. Undesirable capabilities are given a value of zero.

The last column on the first page has space for recording intermediate values. For each group of capabilities it is necessary to count the number of desired capabilities (note that the count for the universal group is pre-printed) and sum the numbers in the *VALUE* column. The definitions for the each group are listed below.

COUNT(R)	number of requirements capabilities marked as desired
SUM(R)	sum of values in requirements capabilities group
COUNT(D)	number of design capabilities marked as desired
SUM(D)	sum of values in design capabilities group
COUNT(U)	number of universal capabilities marked as desired
SUM(U)	sum of values in universal capabilities group

The second page of the worksheet has the formulas for computing the scores. The four formulas labeled COUNT(1), COUNT(2), SUM(1), and SUM(2) are intermediate values. (The number inside the parentheses indicates the path number, corresponding to Paths 1 or 2 in the Guidebook. Since the success of Path 3 depends on the expertise of a user, any methodology scoring for Path 3 will derive from the user's own formula.)

Desirable Design Phase Capabilities

		Architectural Design Concerns								Quality Assurance Techniques				Detailed Design Concerns	
		Structuring Techniques					Data base defin ition	Con cur ency	Mod ule Inter face Dfn	Formal Verif cation	Config uration Mgmt	Static Analysis		Ada Compat ible	Code Behav ior
		Decomposition			Abstraction							Comp lete	Consis tency		
		func tion al	data	cont trol	data	pro cess									
S	1		x		x		x		x		x	x	x	x	x
O	2	x		x		x		x	x		x	x	x	x	x
F	3	x		x		x		x	x		x	x	x	x	x
T	4	x				x		x	x	x	x	x	x	x	x
W	5	x						x	x	x	x	x	x	x	x
A	6	x	x	x	x	x		x	x	x	x	x	x	x	x
R	7	x		x	x	x		x	x	x	x	x	x	x	x
E	8	x	x		x	x		x	x		x	x	x	x	x
	9		x		x		x		x		x	x	x	x	x
C	10	x		x	x	x		x	x		x	x	x	x	x
A	11	x	x	x	x	x		x	x		x	x	x	x	x
T	12		x		x		x		x		x	x	x	x	x
E	13	x		x	x	x		x	x		x	x	x	x	x
G	14		x		x		x		x		x	x	x	x	x
O	15	x				x	x		x		x				
R	16	x	x		x		x	x	x		x	x	x	x	x
Y	17	x	x		x	x		x	x		x	x	x	x	x
	18	x	x		x				x		x	x	x	x	x

Figure 7-4: Design Capabilities versus Software Category

The general formula for computing the Methodology Score (MS) for a particular path and significance level is:

$$MS(path, sl) = SUM(path) - sl * COUNT(path).$$

The first term of the formula [SUM(path)] is the support power the methodology provides. The second term [sl * COUNT(path)] is the support power an ideal methodology would provide at a given significance level with a uniform [7] rating (the specific sl) per capability. Thus, the second term serves as a correction factor.[8] The MS value represents the support power of the methodology relative to the support power of the *ideal* methodology. A positive result indicates that the selected methodology provides more support to the specification process than nominally desired. A negative result indicates less support than nominally desired.

The remaining 8 formulas on the page are resolutions of the general formula for Path 1 (the *requirements* phase path), Path 2 (the *design* phase path), and significance levels 0 through 3. The Methodology Score (MS) computation for both paths includes all the universal capabilities since they will be desirable in both requirements and design specification phases. The MS computation for Path 1 includes requirements and design capabilities since it chooses a methodology that will be used beginning in the requirements phase and continuing through the design phase. The MS computation for Path 2 excludes the requirements capabilities since the selected methodology will be used beginning

7. The formula for computing the Methodology Score (MS) if the capabilities desired in the *ideal* methodology are treated non-uniformly is:

$$MS = \sum_{\text{all desired capabilities}} (r_{methodology} - r_{ideal})$$

where $r_{methodology}$ is the methodology's rating for a particular capability and r_{ideal} is the rating given that capability in the *ideal* methodology. Note that the MS sum only includes the capabilities desired in the *ideal*. Thus, methodologies are not penalized for providing capabilities other than those nominally desired, which is reasonable since undesired capabilities can be ignored.

8. Notice that the correction factor for significance level 0 is 0. This corresponds to the realistic assumption that development of a project with the following characteristics does not need specification technology: (a) Low, tight budget, (b) no criticality assignment, (c) tight schedule, (d) straight-forward solution and easy to check out, (e) few formal requirements, (f) expected use in a local environment as test or demonstration software, (g) not required to recover from anomalous conditions, (h) acceptance predicated solely on correct functionality, (i) not expected to be maintained, and (j) documentation confined to source code.

in the design phase.

The results for path 1 are used to update Guidebook figures 2-9 through 2-12 (for significance level 0 through 3 respectively). The results for path 2 are used to update Guidebook figures 2-17 through 2-20 (for significance level 0 through 3 respectively). The columns of the figures represent software categories; the rows of the figures represent methodologies. Updating a methodology's scores is a matter of locating the proper row and column and modifying the value. Figure 7-5 locates the proper entry for updating the OSL value of 2 for path 1, software category 5 and methodology F (SREM). Adding a methodology is a matter of adding a new row, and entering values in the proper columns for the each software category.

7.2. Expansion of the Capabilities List

If a capability is added to the capability list, it will be necessary to recompute the scores for every methodology with respect to each software category in which that capability is desired. If the capability belongs to the universal group, it will be necessary to recompute scores for every methodology and every software category. If the capability belongs to the requirements group, it will only be necessary to recompute Path 1 scores. If the capability belongs to the design or universal group, it will be necessary to recompute both Path 1 and 2 scores.

The steps for adding a capability follow:

(1) Define the capability.

(2) Define the significance level ratings for the capability. (The ratings used in the guidebook are defined in Appendix B.)

(3) Decide the group (requirements, design, or universal) to which the capability belongs. That is, is the utility of the capability independent of life cycle phase? If so, then it belongs to the universal group. If not, then in which phase is it useful?

(4) If the group is requirements or design, update the appropriate desirability matrix (figure 7-2 or 7-3) to show the desirability of the capability relative to the 18 software categories.

(5) Rate the methodologies for that capability, adding the ratings to Guidebook figure 2-22.

(6) For every methodology recompute the significance level scores for the paths and software categories impacted by the addition of the capability.

Path 1

Path 1
Methodology Scores (OSL = 2)

OSL 2

Methodology	Software Category																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	-20	-21	-21	-22	-23	-24	-29	-22	-20	-29	-24	-20	-29	-20	-18	-21	-22	-17
B	-6	-10	-10	-9	-9	-13	-10	-10	-6	-10	-13	-6	-10	-6	-9	-9	-10	-5
C	-18	-17	-17	-20	-21	-22	-26	-21	-18	-26	-22	-18	-26	-18	-18	-21	-21	-17
D	-12	-13	-13	-14	-15	-15	-19	-15	-12	-19	-15	-12	-19	-12	-10	-13	-15	-11
E	-13	-16	-16	-18	-18	-18	-20	-15	-13	-20	-18	-13	-20	-13	-16	-16	-15	-12
F	0	-2	-2	-2	-1	-6	-5	-6	0	-5	-6	0	-5	0	-3	-3	-6	-2
G	-17	-25	-25	-26	-23	-26	-27	-23	-17	-27	-26	-17	-27	-17	-21	-19	-23	-17
H	3	-4	-4	-5	-2	-5	-6	-2	3	-6	-5	3	-6	3	-7	2	-2	3
I	-16	-23	-23	-24	-26	-23	-27	-20	-16	-27	-23	-16	-27	-16	-18	-21	-20	-16
J	-11	-8	-8	-10	-9	-14	-13	-11	-11	-13	-14	-11	-13	-11	-14	-14	-11	-12
K	-4	1	1	-1	-1	-1	-5	1	-4	-5	-1	-4	-5	-4	-6	-3	1	0
L	-4	-7	-7	-4	-9	-8	-9	-8	-4	-9	-8	-4	-9	-4	-8	-7	-8	-4

category

SREM

Figure 7-5: Example Update to Guidebook Path 1 (OSL=2) Table

8. RECOMMENDATIONS

We recommend that the methodology used in the Guidebook be computerized, so that a user-friendly interface (perhaps menu-driven) would lead the user step-by-step through the selection procedure. The software could be programmed to perform all computations and pattern matching now required of the user. The Guidebook selection method has been designed in a modular fashion so that tables can be easily changed to reflect advances in the technology, and easily expanded to include new methodologies and techniques. This same structure could be implemented on a computer, permitting the same growth and flexibility as the Guidebook. Further, the ability of a menu-driven scheme to vary its actual presentations based on the software category involved would make it feasible to include more quality attributes in the significance level computations.

Our second recommendation is to re-evaluate the 18 standard software categories after several years of use. By that time, any significant problems should be evident, either in the categories themselves or in clarity of definitions. Usage in the field often uncovers difficulties unforeseen by the designers; and software engineers often have different vantage points for viewing the world of software. During our investigation, we found no compelling reason to modify the categories, which were derived as part of another contract to build a Software Test Handbook (RADC-TR-84-53, Vol. II) for the Air Force.

Our third recommendation is that consideration be given to a study that rates the effectiveness of usage of the Guidebook selections. Appropriate questions might be:

(a) Was the methodology a success? (Remember our survey did not uncover any successes.)

(b) What capabilities would have been helpful, but were not provided? (Should the list of capabilities be expanded?)

(c) What capabilities were not used? (Should the recommended capabilities for a particular software category be adjusted?)

(d) Did any capability have a negative impact on meeting project goals such as cost or schedule? (Does the significance level determination need adjustment?)

BIBLIOGRAPHY

Ali Mili, *Workshop Notes of International Workshop on Models and Languages for Software Specification and Design,* IEEE Computer Society Press, March 30, 1984.

Abbot, R.J. and D.K. Moorhead, "Software Requirements and Specifications: A Survey of Needs and Languages," *Journal of Systems and Software*, pp. 297-316, December 1981.

Alford, M.W., "Software Requirements Engineering Methodology (SREM) At the Age of Four," *COMPSAC '80 Proceedings*, pp. 366-374, 1980.

Alford, Mack, *Requirements For Distributed Data Processing Design,* IEEE publication CH1445-6/69/00000-001, 1979.

Alford, Mack, "SREM At the Age of Eight: The Distributed Computing Design System," Draft, December 1984.

Alford, Mack W., "A Requirements Engineering Methodology for Real-Time Processing Requirements," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 60-69, January 1977.

Azuma, M., T. Tabata,Y. Oki, andS. Kamiya, "SPD: A Humanized Documentation Technology," *Proceedings of COMPSAC '83*, Chicago, Ill., November 7-11, 1983.

Balzer, Robert and Neil Goldman, "Principles of Good Software Specification and Their Implications for Specification Languages," *Proceedings of Specifications of Reliable Software*, pp. 58-67, IEEE Computer Society Press, April 3-5, 1979.

Beck, Leland L. and Thomas E. Perkins, "A Survey of Software Engineering Practice: Tools, Methods, and Results," *Transactions on Software Engineering*, vol. SE-9, no. 5, pp. 541-561, IEEE Computer Society Press, September 1983.

Beichter, F.W., O. Herzog, and H. Petzsch, "SLAN-4 -- A Software Specification and Design Language," *Transactions on Software Engineering*, vol. SE-10, no. 2, pp. 155-162, IEEE Computer Society Press, March 1984.

Belcastro, Richard J., "Specification Template Speeds Software Design," *EDN Design Management*, October 27, 1982.

Bell, Thomas E. and David C. Bixler, *A Flow Oriented Requirements Statement Language*, TRW Software Series, April 1976.

Bergland, G. D., "A Guided Tour of Program Design Methodologies," *Computer*, pp. 13-37, IEEE Computer Society Press, October 1981.

Bjorner, Dines, *The VDM Principles of Software Specification & Program Design*, Lecture Notes in Computer Science: Formalization of Programming Concepts, pp. 45-74.

Boebert, W.E., W.R. Franta, and H. Berg, "NPN: A Finite-State Specification Technique for Distributed Software," *Proceedings of Specifications of Reliable Software*, pp. 139-149, IEEE Computer Society Press, April 3-5, 1979.

Boehm, B.W., T.E. Gray, and T. Seewaldt, "Prototyping Versus Specifying: A Multiproject Experiment," *Transactions on Software Engineering*, vol. SE-10, no. 3, pp. 290-302, IEEE Computer Society Press, May 1984.

Booch, Grady, in *Software Engineering with Ada*, The Benjamin/Cummings Publishing Company, Inc., 1983.

Boydstun, Louis E., Daniel Teichroew, Steven Spewak, Yuzo Yamamoto, and Guy Starner, "Computer Aided Modeling of Information Systems," *Proceedings of the COMPSAC '80*, IEEE publication CH1607-1/80/0000-0037, Chicago,Ill., October 27-31.

Brackett, Michael H., *Developing Data Structured Information Systems*, Ken Orr & Associates, Inc, Topeka, Kansas, 1983.

Britton, Kathryn Heninger, R. Alan Parker, and David L. Parnas, "A Procedure for Designing Abstract Interfaces for Device Interface Modules," *Proceedings of 5th International Conference on Software Engineering*, pp. 195-204, March 1981.

Burstini, Meir, Yoram Forscher, Yossi Maimon, and Itzhak Rotbard, "SuperPDL - A Software Design Tool," *Proceedings of SoftFair '83*, pp. 307-313, IEEE publication CH1919-0/83/0000/0307, July 25-28, 1983.

Campbell, Roy H. and Peter A. Kirslis, "The SAGA Project: A System for Software Development," *Proceedings ACM Sigsoft/Sigplan Software Engineering Symposium*, pp. 73-80, ACM, Pittsburgh, PA, April 23-25, 1984.

Celko, Joe, John S. Davis, and John Mitchell, "A Demonstration of Three Requirements Language Systems," *SIGPLAN Notices*, vol. 18, no. 1, January 1983.

Chen, B.-S. and R.T. Yeh, "Formal Specification and Verification of Distributed Systems," *Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 710-722, IEEE Computer Society Press, November 1983.

Cheng, Lorna L., Mary Lou Soffa, and Yee-Hong Yang, "Simulation of an I/O Driven Requirements Language," *Proceedings of COMPSAC '82*, Chicago, Ill., November 8-12, 1982.

Chmura, Louis J. and David M. Weiss, "The A-7E Software Requirements Document: Three Years of Change Data," *Proceedings from AGARD Conference CP-330*, September 1982.

Clemmensen, Geert B. and Ole N. Oest, *Formal Specification and Development of an Ada Compiler - A VDM Case Study*, IEEE publication 0270-5257/84/00000/430, 1984.

Collahan, J.E. Jr. and N. Roussopoulos, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets," *Transactions on Software Engineering*, vol. SE-9, no. 5, pp. 603-616, IEEE Computer Society Press, September 1983.

Coulter, Mel A., "Evolution of the Structured Methodologies: Implications for Instruction," *Proceedings of the American Institute for Decision Sciences 11th Annual Meeting*, pp. 291-293, March 1982.

Cristian, F., "Correct and Robust Programs," *Transactions on Software Engineering*, vol. SE-10, no. 2, pp. 163-174, IEEE Computer Society Press, March 1984.

Davis, Alan M., "Formal Techniques and Automatic Processing to Ensure Correctness in Requirements Specifications," *Proceedings of Specifications of Reliable Software*, pp. 15-35, IEEE Computer Society Press, April 3-5, 1979.

Davis, Alan M., "Automating the Requirements Phase: Benefits to Later Phases of the Software Life-Cycle," *Proceedings of COMPSAC '80*, pp. 42-48, IEEE publication CH1607-1/80/0000-0042, Chicago,Ill., October 27-31, 1980.

Davis, Gary I., *Man-Machine Interface Design Using Structured Analysis and Simulation Techniques*, IEEE publication 0360-8913/82/0000-0233.

Davis, Margaret J., *Structural Specification: High-Level Program Design,* University of Delaware , May 1984. Master's Thesis

Deutsch, Michael S., "A Software Engineering Development System Using Structured Methods," *Proceedings of the 15th Asilomar Conference*, pp. 156-159, IEEE Computer Society Press, October 9-11, 1981.

Deutsch, Michael S., "An Industrial Software Engineering Methodology Supported by an Automated Environment," *Proceedings of the National Computer Conference*, pp. 301-307, 1982.

Dibble, R., "Software Design and Development Using Mascot," *Proceedings from AGARD Conference CP-330 (Software for Avionics)*, September 6-10, 1982.

Doane, Robert B., *The Evolving Nature of the C3 Systems Acquisition Process,* Source of paper currently unknown.

Dubois, E., J.P. Finance, and A. Lamsweerde, "Towards A Deductive Approach to Information System Specification and Design," in *Requirements Engineering Environments*, ed. Y. Ohno, pp. 23-32, North-Holland Publishing Company, 1982.

Durell, William R., *Cohesion: A Means to Data Base Flexibility,* Computerworld, October 10, 1983.

Emerson, Thomas J., "The Logical Structure of Software Design," *Proceedings of COMPSAC '81*, pp. 363-368, IEEE publication CH1698-0/81/0000/0363, Chicago, Ill., November 16-20, 1981.

Epple, W.K., M.D. Hagemann, M.K. Klump, and U. Rembold, "The Use of Graphic Aids for Requirements Specification of Process Control Systems," *Proceedings of COMPSAC '83*, Chicago, Ill., November 7-11, 1983.

Everhart, C. R., "A Unified Approach to Software System Engineering," *Proceedings of COMPSAC '80*, pp. 49-55, IEEE Computer Society Press, Chicago,Ill., October 27-30, 1980.

Freeman, Peter and Anthony I. Wasserman, *Software Development Methodologies and Ada* , DoD, November 1982. (Known as Methodman I.)

Gerhart, Susan L. and David S. Wile, "Preliminary Report on the Delta Experiment: Specification and Verification of A Multiple-User File Updating Module," *Proceedings of Specifications of Reliable Software*, pp. 198-211, IEEE Computer Society Press, April 3-5, 1979.

Giddings, Richard V., "Accommodating Uncertainty in Software Design," *Communications of the ACM*, vol. 27, no. 5, pp. 428-434, May 1984.

Gieszl, Louis R., *The Logical Design of a Major AFLC Information System* , IEEE publication CH1810-1/82/0000/0153.

Goguen, Joseph A., "An Introduction to OBJ: A Language for Writing and Testing Formal Algebriac Program Specifications," *Proceedings of Specifications of Reliable Software*, pp. 170-189, IEEE Computer Society Press, April 3-5, 1979.

Grabow, Paul C., William B. Noble, and Cheng-Chi Huang, *Reusable Software Implementation Technology Reviews*, Hughes Aircraft Company, October 1984.

Heitmeyer, C.L. and J.D. McLean, "Abstract Requirements Specification: A New Approach and Its Application," *Transactions on Software Engineering*, vol. SE-9, no. 5, pp. 580-589, IEEE Computer Society Press, September 1983.

Heninger, Kathryn L., "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 1, pp. 2-13, January 1980.

Hester, S.D., D.L. Parnas, and D.F. Utter, "Using Documentation as a Software Design Medium," *The Bell System Technical Journal*, pp. 1941-1977, October 1981.

Hirschmann, Lutz and Niels Christensen, "The Computer Aided Specification System Easy ," *Proceedings from AGARD Conference CP-330 (Software for Avionics)*, September 6-10, 1982.

Hoare, C.A.R., *Communicating Sequential Processes*, IEEE publication ACM 0001-0782/78/0800.

Howden, William E., "Validation of Scientific Programs," *Computing Surveys*, vol. 14, no. 2, June 1982.

Howden, William E., "Contemporary Software Development Environments," *Communications of the ACM*, vol. 25, no. 5, pp. 318-329, May 1982.

Irvine, C. A. and John W. Brackett, "Automated Software Engineering Through Structured Data Management," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, January 1977.

Iwahashi, David S., "Order and Discipline: Benefits of Structured Techniques," *Datamation*, October 1979.

Jackson, Michael, *System Development*, Prentice/Hall International, 1983.

Jalote, Pankaj, "Specification and Testing of Abstract Data Types," *Proceedings of COMPSAC '83*, Chicago, Ill., November 7-11, 1983.

Jard, Claude and Gregor V. Bochmann, "An Approach to Testing Specifications," *The Journal of Systems and Software*, 1983.

Johnson, Donna, "Selecting a Methodology for Requirements Analysis," *Proceedings of IEEE Phoenix Conference*, pp. 357-361, IEEE publication CH1864-8/83/0000/357, 1983.

Jolley, Truman, *Software Engineering Toolbox Selection for the Cobbler's Children*, BCS BAC-SD, June 30, 1983.

Jones, C.B., *The Vienna Development Method: The Meta-Language*, Lecture Notes in Computer Science, Springer-Verlag, 1978.

Jones, Capers, "A Survey of Programming Design and Specification Techniques," *Proceedings of Specifications of Reliable Software*, pp. 91-103, IEEE Computer Society Press, April 3-5, 1979.

Jones, Cliff B., *Software Development: A Rigorous Approach*, Prentice/Hall International, 1980.

Jordan, D. and B. Hauxwell, *Proceedings from AGARD Conference CP-330 (Software for Avionics)*, September 6-10, 1982.

Kampen, G.R., "SWIFT: A Requirements Specification System for Software," in *Requirements Engineering Environments*, ed. Y. Ohno, pp. 77-84, North-Holland Publishing Company, 1982.

Kanda, Yasunori and Masakatsu Sugimoto, "Software Diagram Description: SDD and It's Application," *Proceedings of COMPSAC '80*, Chicago,Ill., October 27-31, 1980.

Kemmerer, R.A., "Testing Formal Specifications to Detect Design Errors," *Transactions on Software Engineering*, vol. SE-11, no. 1, pp. 32-43, IEEE Computer Society Press, January 1985.

Ken Orr & Associates, Inc., *Data Structured Systems Development Methodology*, Ken Orr & Associates, Inc, Topeka, Kansas, 1977.

Koch, G. and U. Rembold, "The Necessity of Requirements Engineering in Process Automation," *Proceedings of COMPSAC '80*, Chicago,Ill., October 27-31, 1980.

Krause, K. W. and L. A. Diamant, "A Management Methodology for Testing Software Requirements," *Proceedings of IEEE COMPSAC '78.*

Kuni, Tosiyasu L. and Kazunori Yamaguchi, "Formalism for Design Evolution," *Proceedings of COMPSAC '80*, Chicago,Ill., October 27-31, 1980.

Kuo, H. C. and J. Ramanathan, "Concept Based Tool for Standardized Program Development," *Proceedings of COMPSAC '81*, November 16-20, 1981.

Laventhal, Mark S., "Synchronization Specifications for Data Abstractions," *Proceedings of Specifications of Reliable Software*, pp. 119-125, IEEE Computer Society Press, April 3-5, 1979.

Lehman, M.M., V. Stenning, and W.M. Turski, "Another Look at Software Design Methodology," *(SEN) ACM Sigsoft Software Engineering Notes*, vol. 9, no. 2, pp. 38-53, ACM, April 1984.

Lipka, Stephen E., "Some Issues in Requirements Definition," *Proceedings of COMPSAC '80*, pp. 56-58, IEEE Computer Society Press, Chicago, Ill., October 27-31, 1980.

Ludewig, J., "ESPRESO - A System for Process Control Software Specification," *Transactions on Software Engineering*, vol. SE-9, no. 4, pp. 427-436, IEEE Computer Society Press, July 1983.

Lund, John C. Jr., Michael R. Ordun, and Ronald J. Wojcik, "Implementation of the Calling Card Service Capability - Application of a Software Methodology," *Proceedings of the International Conference on Communication*, Denver, June 1981.

M.H. Cheheyl, *et al*, "Verifying Security," *ACM Computing Surveys*, vol. 13, no. 3, pp. 279-340, September 1981.

Maekawa, Mamoru, "Extensibility and Adaptability of Distributed Computing Systems," *Proceedings of COMPSAC '80*, Chicago, Ill., October 27-31, 1980.

Marca, D. and D. Thornhill, "Modeling Software Configurability Requirements," in *Requirements Engineering Environments*, ed. Y. Ohno, pp. 51-58, North-Holland Publishing Company, 1982.

Marca, D. and C. McGowan, "Static and Dynamic Data Modeling for Information System Design," *Proceedings of IEEE Software Engineering 6th International Conference*, September 13-16, 1982.

Matsumoto, Y., T. Tanaka, and S. Kawakita, "Specification Transformations and a Requirements Specification of Real-Time Control," in *Requirements Engineering Environments*, ed. Y. Ohno, pp. 143-150, North-Hol'and Publishing Company, 1982.

McCoyd, Gerard C. and John R. Mitchell, "System Sketching: The Generation of Rapid Prototypes for Transaction Based Systems," *(SEN) ACM Sigsoft Software Engineering Notes*, vol. 7, no. 5, December 5, 1982.

Meyer, Betrand, "On Formalism in Specifications," *Software*, vol. 2, no. 1, pp. 6-26, IEEE Computer Society Press, January 1985.

Mili, A., "System Requirements Specification: A Simplified Approach," in *Requirements Engineering Environments*, ed. Y. Ohno, pp. 123-132, North-Holland Publishing Company, 1982.

Miller, T.J. and B.J. Taylor, "A Requirements Methodololgy for Complex Real-Time Systems," in *Requirements Engineering Environments*, ed. Y. Ohno, pp. 133-142, North-Holland Publishing Company, 1982.

Mishima, Yoshitake, Susumu Murai, and Masahi Okada, *Experience with Tool-Kit Approach in SMEF Prototyping,* IEEE publication CH1919-0/83/0000/0223.

Morton, Richard and Karl Freburger, "Toward Methodology for Function Specifications," *Proceedings of COMPSAC '80*, pp. 201-206, IEEE Computer Society Press, Chicago, Ill., October 27-31, 1980.

Nakao, Okazuo, Koichi Haruna, Noishia Komoda, and Hiroyuki Kaji, "A Structural Approach to System Requirements Analysis of Information Systems," *Proceeding of COMPSAC '80*, pp. 207-213, IEEE Computer Society Press, Chicago, Ill., October 27-31, 1980.

Nyari, Erika and Harry Sneed, "SOFSPEC: A Pragmatic Approach to Automated Specification Verification," *The Journal of Systems and Software*, 1983.

Orr, Ken, *Structured Requirements Definition*, Ken Orr & Associates, Inc, Topeka, Kansas, 1981.

Parnas, David L., "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM*, pp. 1053-1058, December 1972.

Payton, T., S. Keller, J. Perkins, S. Rowan, and S. Mardinly, "SSAGS: A Syntax and Semantics Analysis and Generation System," *Proceedings of COMPSAC '82*, Chicago, Ill., November 8-12, 1982.

Penedo, Maria Heloisa, Daniel M. Berry, and Gerald Estrin, "An Algorithm to Support Code-Skeleton Generation for Concurrent Systems," *Proceedings 5th International Conference on Software Engineering*, pp. 125-135, March 1981.

Pettus, Robert O. and Michael J. Trask, "A Pragmatic Approach to Top-Down Program Design," *Proceedings of IEEE 1982 Southeast Conference*, April 4-7, 1982.

Phillips, N.C.K., "Safe Data Type Specification," *Transactions on Software Engineering*, vol. SE-10, no. 3, pp. 285-289, IEEE Computer Society Press, May 1984.

Pirnia, Sham and Marsha J. Hayek, *Requirements Definition Approach for an Automated Requirements Traceability Tool*, IEEE publication 0547-3578/81/0000-0389.

Poston, Robert M., "Preventing Software Requirements Specification Errors with IEEE 830," *Software*, vol. 2, no. 1, pp. 83-86, IEEE Computer Society Press, January 1985.

Price, C. P. and D. Y. Forsyth, "Practical Considerations in the Introduction of Requirements Analysis Techniques ," *Proceedings from AGARD Conference CP-330 (Software for Avionics)* , September 6-10, 1982..

Prywes, N., B. Szymanski , and Y. Shi, "Dataflow Specification of Concurrent Programs," *Proceedings of COMPSAC '83*, Chicago, Ill., November7-11, 1983.

Rajarman, M.K., "A Characterization of Software Design Tools ," *(SEN) ACM Sigsoft Software Engineering Notes*, October 1982.

Ramamritham, K. and R.M. Keller, "Specification of Synchronizing Processes," *Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 722-733, IEEE Computer Society Press, November 1983.

Rauch-Hinden, Wendy, "The Software Industry Automates Itself," *SOFTWARE*, October 1983.

Razouk, Rami R. and Gerald Estrin, "Modeling and Verification of Communication Protocols in SARA: the X.21 Interface," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1038-1052, December 1980.

Reifer, Donald J. and Stephen Trattner, "Software Specification Techniques: A Tutorial ," *Proceedings of COMPCON '76*, pp. 39-44.

Riddle, W.E., "A Study of Software Technology Maturation," *(SEN) ACM Sigsoft Software Engineering Notes*, vol. 9, no. 2, pp. 21-37, ACM, April 1984.

Roman, G.C. and R.K. Israel, "A Formal Treatment of Distributed Systems Design," in *Requirements Engineering Environments*, ed. Y. Ohno, pp. 3-12, North-Holland Publishing Company, 1982.

Roman, G.C., "A Rigorous Approach to Building Formal System Requirements ," *Proceedings of COMPSAC '82*, Chicago, Ill., November 8-12, 1982.

Ross, Douglas T., John B. Goodenough, and C.A. Irvine, "Software Engineering: Process, Principles, and Goals," *Computer*, pp. 17-27, IEEE Computer Society Press, May 1975.

Ross, Douglas T. and Kenneth E. Schoman, Jr., "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 6-15, January 1977.

Ross, Douglas T., "Reflections on Requirements," *Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 2-5, IEEE Computer Society Press, January 1977.

Ross, Douglas T., "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 16-34, January 1977.

Rudkin, Ralph I. and Kenneth D. Shere, "Structured Decomposition Diagram: A New Technique for System Analysis ," *DATAMATION*, October 1979.

Rzepka, William E., "Software Design Methodologies-Some Management Perspectives ," *Rome Air Development Center*, no. RADC-TR-82-50, Griffiss Air Force Base, NY.

Rzepka, William E., *Using SREM to Specify Command and Control Software Requirements*, RADC-TR-82-319, Rome Air Development Center, Griffiss Air Force Base, NY, 1982.

Rzepka, William E., "RADC SREM Evaluation Program - A Status Report," *(SEN) ACM Sigsoft Software Engineering Notes*, vol. 8, no. 1, pp. 20-22, January 1983.

Scheffer, Paul A., "The Software Designer Workbench DWB ," *IEEE publication CH1919-0/83/0000/0291.*

Schoffelman, Daniel J., "Some Practical Guidelines for Software Design," *Proceedings of IEEE Conference*, Phoenix, AZ, 1983.

Shaw, M., G.T. Almes, J.M. Newcomer, B.K. Reid, and W.A. Wulf, "A Comparison of Programming Languages for Software Engineering ," *Software - Practice and Experience*, vol. 11, pp. 1-52, 1981.

Shaw, R.C., P.N. Hudson, and N.W. Davis, "Introduction of a Formal Technique into a Software Development Environment," *(SEN) ACM Sigsoft Software Engineering Notes*, vol. 9, no. 2, pp. 54-79, April 1984.

Simpson, H. R., "Mascot Development to Improve Software Structure and Integrity ," *Proceedings of IEEE Conference CP-330 (Software for Avionics)* , September 6-10, 1982.

Smoliar, S. W., "Operational Requirements Accommodation in Distributed System Design," *Proceedings of COMPSAC '80*, pp. 214-219, IEEE Computer Society Press, Chicago, Ill., October 27-31, 1980.

Smoliar, Stephen W., "Using Applicative Techniques to Design Distributed Systems," *Proceedings of Specifications of Reliable Software*, pp. 150-161, IEEE Computer Society Press, April 3-5, 1979.

Software Engineering Technical Committee of the IEEE Computer Society Press, *IEEE Guide to Software Requirements Specifications*, IEEE Computer Society Press, New York, NY 10017, 1984.

Standish, Thomas A. and Richard N. Taylor, "Arcturus: A Prototype Advanced Ada Programming Environment," *Proceedings ACM Sigsoft/Sigplan Software Engineering Symposium*, pp. 57-64, ACM, Pittsburgh, PA, April 23-25, 1984.

Stephens, Sharon A. and Leonard L. Tripp, "Requirements Expression and Verification Aid ," *Software Engineering*, 1978.

Stone, A., D. Hartschuh, and B. Castor, *SREM Evaluation*, Rome Air Development Center, February 1984.

Studer, R., "Using VDM for the Development of Interactive Application Systems," in *Requirements Engineering Environments*, ed. Y. Ohno, pp. 13-22, North-Holland Publishing Company, 1982.

Swann, T.G., "Requirements Decomposition and Other Myths," *Proceedings from AGARD Conference CP-330 (Software for Avionics)*, September 6-10, 1982.

Taylor, Bruce, "Say What You Mean with a Language for Software Specifications," *Data Communications*, pp. 131-143, March 1982.

Teichroew, Daniel, Ernest A. Hershey III, and Michael J. Bastarache, *An Introduction to PSL/PSA* , ISDOS Project - University of Michigan, March 1974.

Tse, T. H. and L. Pong, "A Review of System Development Systems," *The Australian Computer Journal*, vol. 14, no. 3, August 1982.

Walter, Claudio, "Control Software Specification and Design: An Overview," *Computer*, February 1984.

Wartik, Steven and Arthur Pyster, "The 'Diversion' Concept in Interactive Computer System Specifications ," *Proceedings of COMPSAC '83*, Chicago, Ill., November 7-11, 1983.

Wasserman, Anthony I. and Susan K. Stinson, "A Specification Method for Interactive Programs," *Proceedings of Specifications of Reliable Software*, pp. 68-79, IEEE Computer Society Press, April 3-5, 1979.

Wasserman, Anthony I., "The User Software Engineering Methodology: an Overview," in *Information System Design Methodologies -- A Comparative Review*, ed. A.A. Verrign-Stuart, North Holland Publishing Company, 1982.

Wasserman, Anthony I. and Steven Gutz, "The Future of Programming," *Communications of the ACM*, pp. 196-206, March 1982.

Witt, Bernard I., "Communicating Modules: A Software Design Model for Concurrent Distributed Systems," *Computer*, vol. 18, no. 1, pp. 67-77, IEEE Computer Society Press, January 1985.

Yamano, Yoichi and Yoshiharu Matsumoto, "Unified Functional Design Technique Based on Data Flow Concept ," *Proceedings of COMPSAC '81*, November 16-20, 1981.

Yau, S.S. and M.U. Caglayan, "Distributed Software System Design Representation Using Modified Petri Nets," *Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 733-745, IEEE Computer Society Press, November 1983.

Yeh, Raymond T. and Pamela Zave, "Specifying Software Requirements," *Proceedings of the IEEE*, vol. 68, no. 9, September 1980.

Yeh, Raymond T., "Requirements Analysis - A Management Perspective," *Proceedings of COMPSAC '82*, pp. 410-416, IEEE Computer Society Press, Chicago, Ill., November 8-12, 1982.

Zajonc, Peter C. and Kevin J. McGowan, *Proto-cycling: A New Method for Application Development Using Fourth Generation Languages*, IEEE publication CH1919-0/83/0000/0127.

Zave, Pamela, "An Operational Approach to Requirements Specification for Embedded Systems," *IEEE Transactions on Software Engineering*, vol. SE-8, no. 3, May 1982.

Zave, Pamela, "Operational Specification Languages," *Proceedings ACM '83*, October 1983.

Zave, Pamela, "An Overview of the PAISLey Project-1984," *(SEN) ACM Sigsoft Software Engineering Notes*, pp. 12-19, July 1984.

Zelkowitz, Marvin V. and James Lyle, "Implementation of Program Specifications ," *Proceedings of COMPSAC '80*, Chicago, Ill., October 27-31,1980.

APPENDIX A

Standard Description Formats

A.1 Methodology Description Format

1. General Aspects

A. Identification

Gives the name and acronym of the methodology and identifies the developing/supporting organization.

B. Overview

Contains a short description of the salient features of the methodology.

C. Identifies the specification life cycle phases supported:

Requirements Analysis, Architectural Design (intermodule communication, data structures), or Detailed Design (module functionality).

Complementary methodologies will be listed for phases not supported.

D. Software Categories

Lists standard software categories which are compatible with this methodology.

#	Category	#	Category
1	Arithmetic-based	2	Event Control
3	Process Control	4	Procedure Control
5	Navigation	6	Flight Dynamics
7	Orbital Dynamics	8	Message Processing
9	Diagnostic S/W	10	Sensor/signal Processing
11	Simulation	12	Database Management
13	Data acquisition	14	Decision/planning aids
15	Data presentation	16	Pattern/image processing
17	Computer System Software	18	S/W development tools

E. Suitable for systems of size:

— Small (<2,000 lines of code)

— Medium (2,000 - 10,000 lines of code)

— Large (>10,000 lines of code)

2. Technical Aspects

A. Primary approach

For a requirements methodology, the approaches are:

— flow-oriented,

— object-oriented, and

— state-oriented.

For a design methodology, the approaches are:

— data-structured,

— decomposition,

— encapsulation, and

— programming calculus.

B. Supports

Traceability
Functional hierarchy/decomposition
Data hierarchy/data abstraction
Interface definition
Database definition
Data flow
Sequential control flow
Concurrency/parallelism
Formal program verification
Iterative development

C. Workproducts

Are they relevant to MIL-STD documentation?

a. Textual

Descriptions of reports, documents produced.

b. Graphical

Descriptions of diagrams produced.

D. Performance Specification

Does the methodology have the capability to specify or test timing and/or accuracy constraints that apply to individual system functions?

E. Operating Qualities Specification

Does the methodology have the capability to specify the following constraints?

— Man/machine interaction

— Fault-tolerance

— Portability

— Reusability

— Security

F. Ada compatibility

Ada Feature	Supported
Packages	
Tasks	
Generics	
Exception Handling	
Types	
Representations	

X indicates support of feature.
C indicates conflict with feature.

G. Quality Assurance

How does the methodology check or enforce:

— Consistency ?

— Completeness ?

— Validation ?

H. Independent of

Are the resulting specifications independent of:

— Implementation Language ?

— Hardware Architecture ?

— Operating System Architecture ?

3. **Support Aspects**

A. Automated Tools

Describes which automated tools are available.

B. Language

Identifies the language used in the following specification phases and its degree of formality.

— Requirements Specification

— Architectural Design

— Detailed Design

4. **Management Aspects**

Does the methodology support project, technical, or configuration management? How?

5. **Usage Aspects**

A. Equipment/Facilities Needed to use

Identify specific hardware and software (operating systems, graphics packages) required to use the methodology or associated automated tools.

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

Is the methodology mature? Has it been used outside the developing organization? How much?

6. Transferability

A. Availability

Is the methodology in the public domain, commercially available, etc.?

B. Training Available

— Public documentation

— Proprietary documentation

— Consultants

— Seminars - scheduling and cost, if known

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1			
1 - 3			
3 - 6			
> 6			

The table entries reflect the amount of training and experience time required to use the methodology effectively. A USER is an individual who develops or assists in developing requirements and/or design specifications. An ORGANIZATION is a group of users developing specifications as a team.

D. Primary Source of Documentation

List references.

A.2 Tool Set Description Format

1. General Aspects

A. Identification

Gives the name and acronym of the tool or tool set and identifies the developing/supporting organization.

B. Methodologies

Lists what methodology the tool set supports.

C. Life cycle phases supported:

Identifies which of the specification phases the tool set supports:

— requirements analysis

— architectural design (intermodule communication, data structures)

— detailed design (module functionality)

D. Software Categories

Lists standard software categories which are compatible with this methodology.

#	Category	#	Category
1	Arithmetic-based	2	Event Control
3	Process Control	4	Procedure Control
5	Navigation	6	Flight Dynamics
7	Orbital Dynamics	8	Message Processing
9	Diagnostic S/W	10	Sensor/signal Processing
11	Simulation	12	Database Management
13	Data acquisition	14	Decision/planning aids
15	Data presentation	16	Pattern/image processing
17	Computer System Software	18	S/W development tools

E. Suitable for systems of size:

— Small (<2,000 lines of code) ?

— Medium (2,000 - 10,000 lines of code) ?

— Large (>10,000 lines of code) ?

2. Technical Aspects

A. Supports

Traceability
Functional hierarchy/decomposition
Data hierarchy/data abstraction
Interface definition
Database definition
Data flow
Sequential control flow
Concurrency/parallelism
Formal program verification
Iterative development

B. Workproducts

Are they relevant to MIL-STD documentation?

a. Textual

Description of reports, documents produced.

b. Graphical

Description of diagrams produced.

C. Performance Specification

Does the tool set have the capability to specify or test timing and/or accuracy constraints that apply to individual system functions?

D. Operating Qualities Specification

Does the tool set have the capability to specify the following constraints?

— Man/machine interaction

— Fault-tolerance

— Portability

— Reusability

— Security

E. Ada compatibility

Ada Feature	Supported
Packages	
Tasks	
Generics	
Exception Handling	
Types	
Representations	

X indicates support of the feature.
C indicates conflict with the feature.

F. Quality Assurance

How does the tool set support

a. Consistency checking ?

b. Completeness checking ?

c. Validation ? by a manual or computer-processed procedure ?

d. Rapid prototyping ?

Does it prototype the man-machine interface? the software modularization scheme? the functionality of the system? Is the execution mode of the prototype a simulation or a symbolic execution? Is the prototype suitable for pre-release?

e. Performance validation ? of correctness or efficiency?

3. **Support Aspects**

A. Degree of Integration

Vertical - within one phase of the software life cycle? Or horizontal - across more than one phase of the software life cycle?

B. Language

Identifies language(s) used for specification phases and its degree of formality.

— Requirements Specification

— Architectural Design

— Detailed Design

4. Management Aspects

Does the tool set support project, technical, or configuration management? How?

5. Usage Aspects

A. Equipment/Facilities Needed to use

Identify specific hardware and software (operating systems, graphics packages) required to use the tool set or associated automated tools.

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

Has the tool set been used outside the developing organization? How much?

6. Transferability

A. *Availability*

Is the tool set in the public domain, commercially available, etc.?

B. *Training Available*

— Public documentation

— Proprietary documentation

— Consultants

— Seminars - scheduling and cost, if known

C. *Training and Experience Required*

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1			
1 - 3			
3 - 6			
> 6			

The table entries reflect the amount of training and experience time required to use the tool set effectively. A USER is an individual who develops or assists in developing requirements and/or design specifications. An ORGANIZATION is a group of users developing specifications as a team.

APPENDIX B

CAPABILITY RATINGS

Interpretations of Capability Ratings

Scale runs from **1** to **3**, with **3** indicating best support and **1** indicating least.

STATE MODELING - Representation in diagrammatic form of the system *state* and transformations to the *state* resulting from inputs or other stimuli.

3 Use of transition diagrams as primary technique for description of requirements.

2 Use of transition diagrams as secondary technique.

1 State indirectly encapsulated by an abstraction (process or data).

DATA FLOW MODELING - Representation of the flow of information objects between various processing elements and/or storage elements.

3 Use of data flow diagrams as primary technique for description of requirements.

2 Use of data flow diagrams as secondary technique.

1 Inputs/outputs indicated on diagrams.

CONTROL FLOW MODELING - Representation of the sequence in which processing will take place.

3 Use of control flow diagrams as primary technique for description of requirements.

2 Use of control flow diagrams as secondary technique.

1 Flow of control implied in diagram (such as transition diagrams).

OBJECT MODELING - Representation of data or processes as independent objects with their own state information and capacity to change.

3 Use of process or data abstractions as primary technique for description of requirements.

2 Use of process or data abstractions as secondary technique.

1 Use of Entity-Relationship diagrams as secondary technique for modeling software system.

TIMING SPECIFICATION - Statement of the timing constraint for a particular processing step.

3 Have capability to formally specify and measure timing constraints.

2 Have capability to formally specify but not measure timing constraints.

1 Can associate timing constraint with a processing step as a comment added to a diagram or textual specification.

ACCURACY SPECIFICATION - Statement of the accuracy constraint for a particular processing step.

3 Have capability to formally specify and measure accuracy constraints.

2 Have capability to formally specify but not measure accuracy constraints.

1 Can associate accuracy constraint with a processing step as a comment added to a diagram or textual specification.

FUNCTIONAL DECOMPOSITION - A function at one level is actually composed of several interconnected functions that exist as a level of greater detail.

3 Primary mechanism for design structuring is functional decomposition.

2 Secondary mechanism for design structuring is functional decomposition.

1 Structure charts of design drawn.

DATA DECOMPOSITION - A set of data at one level is actually composed of several interconnected pieces of data that exist as a level of greater detail.

3 Primary mechanism for design structuring is data decomposition.

2 Secondary mechanism for design structuring is data decomposition.

1 Data dictionary is maintained.

CONTROL DECOMPOSITION - The flow of control at one level is actually composed of several interconnected control flows that exist as a level of greater detail.

3 Primary mechanism for design structuring is control decomposition.

2 Secondary mechanism for design structuring is control decomposition.

1 Calling tree is maintained.

DATA ABSTRACTION - The concept of hiding information about the implementation of a data object and providing a set of implementation-independent

functions for use of the data object.

3 Primary mechanism for design structuring is data abstraction.

2 Secondary mechanism for design structuring is data abstraction.

1 Principle of information hiding is one consideration during design.

PROCESS ABSTRACTION - The concept of hiding information about the implementation of a process object and providing a set of implementation-independent functions for use of the process object.

3 Primary mechanism for design structuring is process abstraction.

2 Secondary mechanism for design structuring is process abstraction.

1 Principle of information hiding is one consideration during design.

DATABASE DEFINITION - The process of defining the data objects and their relationships as a data model or semantic hierarchy.

3 A database design *per se* is one part of the design process.

2 Data objects are defined as abstract data types.

1 The valid ranges and formats for data objects are defined.

CONCURRENCY/SYNCHRONICITY - Processing can be defined as separate sequential streams concurrently in execution.

3 Concurrent sequential cooperating processes with explicit synchronization specified.

2 Concurrent sequential cooperating processes without explicit synchronization specified.

1 Specification of concurrency within a code unit.

MODULE INTERFACE SPECIFICATION - The concept of having distinct and well-defined boundaries between processes or sets of data. The inputs, outputs, calling mechanism, and pre-conditions for use can be specified.

3 Specify inputs, outputs, mechanism, and pre-conditions with a formal notation. Consistency checks computer-processed.

2 Specify inputs, outputs, and mechanism with a formal notation. Consistency checks computer-processed.

1 Specify inputs, outputs, and mechanism with an informal notation. Consistency checks manually-processed.

FORMAL VERIFICATION - The process of proving that the formalization of the processing abstraction exhibits the desired behavior.

3 A theorem prover is provided to assist the verification process.

2 The specification is formal, and can be verified manually.

1 Justification is provided as narrative text.

CONFIGURATION MANAGEMENT - The way in which the methodology and/or its tools provides for organization, tracking, and maintenance of the emerging workproducts, including control of releases and multiple versions.

3 Automated support for organization, tracking, and maintenance of workproducts.

2 Partial automated support for organization, tracking, and maintenance of workproducts.

1 Version control alone is provided.

COMPLETENESS ANALYSIS

3 Computer-assistance for completeness analysis provided, syntax and simulated or symbolic execution (flow analysis) are checked.

2 Computer-assistance for syntax checks.

1 Completeness checked by manual procedures (author/reader; walkthroughs).

CONSISTENCY ANALYSIS

3 Computer-assistance for consistency analysis provided, syntax and simulated or symbolic execution (flow analysis) are checked.

2 Computer-assistance for syntax checks.

1 Consistency checked by manual procedures (author/reader; walkthroughs).

ADA COMPATIBILITY - The methodology can produce a design that can be implemented in ADA, making use of its principal features.

3 All of the principal features (packages, tasks, generics, exception handling, types, and representations) can be utilized.

2 Some of the principal features (packages, tasks, generics, exception handling, types, and representations) can be utilized. There are not any conflicts between Ada features and the methodology.

1 Some of the principal features (packages, tasks, generics, exception handling, types, and representations) can be utilized. There are conflicts between the methodoloy and and at least one Ada feature and some other incompatibilities may exist.

CODE BEHAVIOR NOTATION - The notation used for describing the internal logic or behavior of a unit of code.

3 The syntax of the notation is formal and can be checked by a computer process. (Pseudocode pdl or formal graphic pdl).

2 The behavior is specified with a structured or rigourous natural language (English) which may include control constructs such as *do ... while...*

1 The behavior is specified with totally free natural language (English).

PROTOTYPING - Simulating the behavior of a system by symbolic or simulated execution.

3 The simulation provides sufficient functionality and efficiency to be used as a pre-release.

2 The simulation mimics all the functionality of the system.

1 The simulation mimics part of the functionality of the system, such as report formats or interactive dialogue.

TEST PLAN GENERATION - Formulation of a plan and a series of tests to validate the functionality and efficiency of the implemented system.

3 Automated support for test plan generation is provided.

2 Manual procedures and guidelines for test plan generation are provided.

1 The methodology recommends that test plans be formulated.

AUTOMATED TOOLS AVAILABLE

3 The automated tools support the entire methodology process.

2 (Not Defined.)

1 The automated tools support part of the process.

TRACEABILITY - The ability to trace requirements through to design elements realizing them.

3 Automated support/enforcement for traceability is provided.

2 Manual procedures for traceability are provided.

1 Manual traces are possible.

TRANSITION BETWEEN PHASES

3 A computer-processable project database is available.

2 A data dictionary is maintained.

1 The methodology spans more than one phase of the software lifecycle.

VALIDATION - The process of verifying that the abstract representation of the system (requirements specification, design specification) does provide all the system functionality in the form desired by the customer.

3 Some form of simulated execution (symbolic or simulation) of the system is possible.

2 Structured walkthroughs are performed, possibly with the customer/user.

1 An author/reader cycle for specifications is followed.

USABILITY - Ease of use.

3 Easy to use.

2 Moderately easy to use.

1 Moderately difficult to use.

MATURITY - Extent of use of the methodology.

3 In use outside the developing organization more than five years.

2 In use outside the developing organization less than five years.

1 Methodology is still evolving.

TRAINING/EXPERIENCE LEVEL - Measure of how much experience and training is needed before a person or project team uses a methodology effectively.

3 Less than one month.

2 One to three months.

1 More than three months.

MIL-STD DOCUMENTATION - Conformance to military documentation standards (MIL-STD-SDS).

3 Directly generate documentation in MIL-STD SDS form.

2 Workproducts produced follow spirit although not strict form of MIL-STD SDS documentation.

1 Can generate documentation in MIL-STD SDS form from workproducts produced by methodology.

TABLE OF METHODOLOGY RATINGS

Capability	Methodology											
	A	B	C	D	E	F	G	H	I	J	K	L
REQUIREMENTS												
state modeling	0	1	0	2	1	3	1	1	1	1	0	2
data flow modeling	3	0	2	3	0	1	0	0	1	0	2	2
control flow modeling	2	0	3	2	0	1	0	0	0	0	2	2
object modeling	1	3	0	1	3	1	3	3	3	3	3	0
timing performance spec	0	0	1	1	1	2	0	0	0	3	3	0
accuracy performance spec	0	2	1	0	1	2	2	2	0	3	0	0
DESIGN												
functional decomposition	2	0	3	0	2	1	1	2	2	0	0	2
data decomposition	3	0	3	2	1	1	2	2	2	0	2	1
control decomposition	2	0	3	2	0	1	0	0	0	0	2	1
data abstraction	0	3	0	0	3	0	3	3	3	0	1	3
process abstraction	0	3	0	0	2	0	0	0	3	3	1	2
data base definition	1	2	0	3	1	3	2	2	2	0	0	3
concurrency/synchronicity	1	0	0	1	2	1	2	3	1	3	3	0
module interface definition	1	3	0	1	1	2	2	2	1	3	3	3
formal verification	0	3	0	0	0	2	1	1	0	0	0	3
configuration management	0	0	0	0	0	0	0	0	0	0	0	2
completeness analysis	1	3	1	1	1	3	1	3	1	3	3	3
consistency analysis	1	3	1	1	1	3	1	3	1	3	3	3
Ada compatibility	1	2	2	1	3	2	1	3	1	2	2	2
code behavior notation	0	3	0	3	2	3	3	3	2	3	0	3
UNIVERSAL												
prototyping	0	0	0	1	0	2	0	2	0	3	0	2
test plan generation	0	0	0	0	0	0	0	3	0	0	2	0
automated tool available	1	3	3	3	0	3	0	3	0	3	3	3
traceability	0	3	1	1	1	3	1	3	1	0	3	0
transistion between phases	1	3	1	2	1	3	1	3	1	0	3	2
validation	2	3	1	1	2	3	1	3	2	3	3	2
usability	2	1	1	2	3	1	1	1	2	1	1	2
maturity	3	3	3	3	2	3	2	1	2	2	2	2
training/experience level	2	1	2	2	3	3	1	1	3	1	2	1
MIL-STD documentation	1	0	1	1	2	2	0	1	0	0	2	1

Figure B-1: Methodology Capabilities Ratings

Attachment 2

Changes to Draft of Specification Technology Final Report
- per phone conference on May 3,1985
- and phone conference on May 6,1985

No.	Location	Change/Comment
1	Sec 1.1, para 2	Replace last sentence and footnote with 'This situation has been further complicated with the introduction of the Ada programming language. Users must understand which of the methods and techniques result in designs that take maximum advantage of the software engineering principles which Ada directly supports and implements'.
2	sec 1.2, para 2	Insert 'of the guidebook' after 'A through F'.
3	sec 1.2, para 4	Remove typo 'a' from before 'methodologies' in 1st sentence.
4	sec 1.2, para 7	Clarify meaning of data base management. Possibly change 'Capabilities' to 'Features' or 'Facilities

No.	Location	Change/Comment
5	figure 1-1	Make figure consistent with text in sec 1.4, 2nd para items a-e.
6	sec 1.4, para 2	Remove typo of 'to' in item e.
7	sec 1.4, para 8	Add 'which direct the use of specification technologies in the development of Air Force systems' to end of sentence.
8	sec 1.4, para 9	Add '(figure 1-3)' after '18 generic software categories.
9	sec 2.1, para 1	Insert 'system' after 'These' in 2nd sentence.
10	sec 2.1,para 3	Add 'taken' after 'approach' in 3rd sentence.
11	sec 2.2.2.1	Amplify meaning of item 1.
12	sec 2.2.2.1	Replace 'by Space' with 'of Space' in item 2.
13	sec 3.2	Replace example of problem with foreign methodologies with clause 'where language and time barriers complicate....'

No.	Location	Change/Comment
14	sec 4.2	Make changes in text so inverse proportions are explained as direct proportions.
15	sec 4.2, item 8	Expand and clarify meaning of correctness.
16	sec 4.4, para 2	Change 'chose' to 'choose' in first sentence.
17	sec 5, para 2	Change 'original proposal design' to 'design originally proposed'.
18	sec 5, para 6	Insert 'Air Force Mission' in front of appendices in first sentence.
19	sec 6, para 2	Add 'of the guidebook' after 'Appendix C'.
20	sec 6, para 2	Second sentence needs a verb.
21	sec 7.1, para 3	Add some text about where ratings come from.

No.	Location	Change/Comment
22	sec 7.1, para 4	Make clear that figures 7-3 & 7-4 are used to fill in desired column of worksheet and make some reference to setting up of ideal methodology.
23	worksheet	Change 'Score' on second sheet to 'MS'.
24	sec 7.1, para 7	Explain that third path is scored by user.
25	sec 7.1, eqn	Explain that MS stands for Methodology Score.
26	sec 7.1, para 8	Add 'nominally' before 'desired' in last 2 sentences.
27	sec 7.1,para 8	Put in rationale for formulas and paths.
28	sec 7.2, para 2	Put in pointer to Appendix B on item 2.
29	Appendix A,	Update same as guidebook version is updated.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END

FILMED

1-86

DTIC