Prospects for Automated Documentation Analysis in Support of Software Quality Assurance

James D. Arthur, Richard E. Nance and K. Todd Stevens

TR 88-33

Technical Report SRC-88-002

Prospects for Automated Documentation Analysis in Support of Software Quality Assurance (An Interim Report)

James D. Arthur, Richard E. Nance and K. Todd Stevens

Systems Research Center
at
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

January 1988

Work supported by the U.S. Navy through the Systems Research Center under Basic Ordering Agreement N60921-83-G-A165-B0038

Prospects for Automated Documentation Analysis in Support of Software Quality Assurance

(An Interim Report)

James D. Arthur, Richard E. Nance and K.Todd Stevens

Systems Research Center

at

Virginia Tech

1.0 Introduction

Software maintenance is a complex and costly activity. Such activities significantly outweigh developmental costs [BOEB79] and are estimated to consume more than half of the total life cycle cost of a system [BOEB76, LIEB78]. Factors contributing to this substantial burden include:

- the demand for and shortage of maintenance personnel who possess the necessary skills demanded by maintenance activities,
- the lack of complementary methods and techniques for performing maintenance activities, and
- the scarcity of tools for supporting activities intrinsic to the maintenance of complex software systems.

In an effort to control the complexity and costs associated with maintenance activities, a group of software engineers at the Naval Surface Warfare Center (NSWC) in Dahlgren, Virginia has developed the Automated Design Description System (ADDS) that supports maintenance activities through the use of reverse engineering techniques. This report examines ADDS relative to current technologies, and discusses the strengths and weaknesses of the system as a tool for supporting software maintenance.

The Automated Design Description System employs reverse engineering concepts to produce specialized documents tailored for the maintenance activity. Many studies tout the need for and benefits of documentation during software maintenance [WASA76]. Other authors have enunciated the principle of *concurrent documentation* [TAUR77] that reflects *current* system design and development status. Although recognized as a necessity, the maintenance of a consistently high level of documentation throughout the software life cycle is rarely achieved. Consequently, software engineers have sought methods and tools to compensate for human inadequacies in documentation. Slowly, reverse engineering concepts have emerged, as have tools such as ADDS.

Reverse engineering, as it applies to supporting the maintenance activities through documentation, can be defined as the synthesis of documents based on the analysis of an existing database. In the particular instance of ADDS (and a complementary counterpart, Micro-ADDS) that database consists of the AEGIS system source code and the Program Performance Specifications (PPS's). In essence, ADDS generates specialized documents based on the structure and relational characteristics of the AEGIS system source code.

The advantages of the reverse engineering approach exploited by ADDS are that

- documents are *automatically* synthesized, and
- document contents reflect characteristics and qualities of analyzed system software.

The disadvantages of this approach, however, are that the synthesized documents

- can only portray "what is" and not the relevant decision processes leading to the current software condition, and
- might not include maintenance changes introduced after the ADDS database has been built.

With respect to ADDS and its capability for supporting the maintenance activities, this report presents the results of an investigation with the primary goal of collecting information that can assist in future modifications and enhancements. The remainder of this report is divided into four (4) major sections. The immediately following section provides an evaluation of ADDS through a review of its inherent strengths and weaknesses. Included in this section is an enumeration and discussion of several positive as well as negative characteristics of ADDS and the generated documents. Section 3 presents modifications and enhancements to ADDS that are intended to eliminate (or reduce) the undesirable characteristics and introduce (or augment) positive code and documentation qualities. The discussion focuses on changes that (a) can be incorporated within the existing system, e.g. the changing of report formats, (b) require only minor system modifications, e.g. the introduction of new capabilities based on current system constraints, and (c) necessitate major system redesign and redevelopment. Section 4 provides a summary, and Section 5 describes possible avenues of future development.

2.0 An Evaluation of ADDS Documents

A major thrust of this research is the evaluation of documents generated by ADDS. Knowing that the synthesized documents are intended to support the maintenance activity provides a basis for identifying pertinent characteristics that are beneficial and desirable, e.g. traceability and consistency. In addition to assessing the presence of such characteristics, the manifestations and

implications of their absence are also considered in the evaluation process. Correspondingly, the following discussion is divided into two major parts reflecting these contrary perspectives. In particular, Section 2.1 discusses the merits of ADDS documents relative to desirable characteristics, whereas Section 2.2 presents a description of specific document deficiencies resulting from the absence of such characteristics.

2.1 Desirable Characteristics Attributed to ADDS-Generated Documents

ADDS generates documents to support maintenance activities. Based on the intended use of such documents, one can identify three desirable characteristics expected, and found to be present, in ADDS-generated documents: *traceability, consistency,* and *completeness.* Moreover, knowing that these documents are being automatically generated, a fourth characteristic one expects to find is *flexibility.* Unlike the first three, flexibility is a dynamic rather than static characteristic that is introduced through the synthesis process. The following four sections define and discuss each of these characteristics relative to documents generated by ADDS.

2.1.1 Traceability

Document traceability can be defined as the ease with which one can relate document information to the external object that is referenced. More intuitively, if a particular system function is described in a document, one should be able to identify directly that function in the source code as well as in higher level documents describing the need for that function. In addition to external traceability, internal document traceability is also desirable. That is, the connection between high level descriptions and their elaboration (or refinement) *within* a document should be easily discernable.

In general, documents generated by ADDS possess a high degree of traceability. Consider, for example, documents that reference various system-level structures [NSWC1, NSWC2, NSWC3].

On a very high level, ADDS provides a System Structure Report that "presents the processing hierarchy of the C/D program, all C/D modules, and the channel program and CSR collections" [NSWC1]. Because this report is synthesized from the source code, the descriptive items consist of actual program, CSR and module names. Subsequently, traceability to the source code is self-evident.

At the next lower system-level structure, traceability is achieved through the enunciation of procedure names. That is, the Procedure Calling Hierarchies Report [NSWC2] "presents the calling hierarchy for all C/D Entrance Procedures" [NSWC1].

Finally, at the variable level, ADDS exploits variable names within the source code to produce two documents that enable users to investigate relationships among data items. The first document describes data item definitions as they relate to outer level data structures within which the items are defined [NSWC3]. In reality, this document is a set of three reports that provide a further partitioning of data items based on whether an item is defined as common data, local data, or temporary data. In addition to providing the actual source code name, each report lists "the type of object named (as defined in the database) and how the preceding name at the next higher level relates to it" [NSWC1]. The second document providing variable level information is the Set/Used Report [NSWC4]. Within this report, each data item in the C/D Baseline 2 computer program is listed alphabetically. For each data item, the report provides the name of every "routine" that sets (updates) or uses (employs) that data item. In this context, a "routine" is defined to be a tactical module procedure, an ADEP module, a common service routine or channel program.

Based on the above descriptions, documents achieve traceability through the use of actual data names used in the system source code. We note, however, such an approach only supports traceability to objects external to a document, i.e. the system source code. Traceability within each document is not readily apparent.

2.1.2 Consistency

Consistency is another desirable document characteristic. In general, document consistency is marked by harmonious regularity in the use of *terms* and document *format*. From the perspective of ADDS-generated documents, consistency is achieved in two ways. First, consistency in the use of terms is ensured through the dictates of the PSL/PSA naming conventions and the CMS-2 Coding and Commenting Standards [NSWC5]. The coding and commenting standards define system-wide guidelines for naming modules, procedures and data items. Similarly, PSL/PSA standards dictate a precise set of rules and naming conventions when defining software objects and relationships among those objects. Because such standards are enforced throughout the software development process, and because ADDS generates documents from a PSL/PSA information database derived from products of the developmental process, consistent terminology is assured throughout and among synthesized reports. The second way consistency is achieved is through the use of accordant presentation formats, automatically synthesized and based on *predefined* document structures. For example, hierarchical relationships are illustrated through indentation, and labels for sections of information are uniformly positioned.

Together, the above mentioned elements characterize a unified process that insures consistency within and among documents generated by ADDS. This consistency significantly contributes to the readability and usability of such documents.

2.1.3 Completeness

Rather than "completeness", the term "conditional completeness" is probably more appropriate in describing the third positive characteristic of ADDS documentation. The dictionary definition of "completeness" is "having all the necessary parts." Within ADDS, documents are generated

according to a set of criteria. For example, the *Procedure Calling Hierarchy* report "presents all C/D computer Entrance Procedures in the C/D Baseline 2 computer programs." Because the actual Baseline 2 source code is automatically analyzed, and because the analytical process has proven reliable over time, we accept such reports as being complete - at least relative to the *query criteria* and *baseline* analyzed. This last qualification, however, mandates the "conditional completeness" categorization. That is, the ADDS documents generated under the current set of selection/specification criteria for Baseline 2 *are* complete. Maintenance activities, however, revolve around a changed or modified version of the Baseline 2 source code. Because the ADDS database reflects only information derived from Baseline 2.0 source code and does not incorporate any later modifications to that baseline, we say that relative to (or *conditioned* on) the information database, ADDS-generated documents are complete.

Although a fixed baseline insures that all maintenance personnel have a common reference point from which to work, it is also important to know exactly how changes will impact the currently running (modified) system. Regarding this crucial aspect, ADDS-generated documents are, unfortunately, deficient. The deficiency, however, does not lie in a document's completeness but in its currency. The implications of non-current documentation is discussed later in this report.

2.1.4 Flexibility

Although not a quality or characteristic one normally associates with documentation, document flexibility is highly desirable and inherent to reports generated by ADDS. Document flexibility can be defined to be the ease with which one can create new documents (or tailor existing ones) based on criteria selected and specified by the user. Because ADDS is an automated document synthesis system, flexibility is realized at two distinct levels: the content/format level and the user specification level.

- On the content/format level, ADDS is designed to accept PSL/PSA specifications that dictate (a) both the format and content of new reports, and (b) modifications to existing reports. This task is further simplified by the fact that ADDS has been developed (and is maintained) "in house".

- On the user level, people who use the system can tailor reports to their individual needs. For example, instead of listing a procedural hierarchy for the entire system, users can list the hierarchy for a single process. This user flexibility comes from ADDS being an on-line, interactive system.

Although the flexibility of designing and tailoring documents through an on-line system is highly desirable, the full potential of such a capability has yet to be exploited by the users. This aspect, along with the other critical statements mentioned above, are more fully explained in the following section.

2.2 Undesirable Characteristics Attributed to ADDS-Generated Documents

Section 2.1 presents several desirable document characteristics (or qualities) present in ADDS-generated documents. In support of maintenance activities, those characteristics (traceability, consistency, conditional completeness and flexibility) significantly contribute to the usability of documents generated by ADDS. In assessing the usability of documents, however, two other qualities must be considered, i.e accuracy and adequacy. Document accuracy is realized through the *error-free* presentation of information. We note that with respect to usability, document accuracy differs from conditional completeness (discussed in Section 2.1) in that an accurate document presents an error-free, *status quo* description of a subject. Document adequacy, on the other hand, can be defined as the sufficiency of the information content in that document (or set of

documents) to support an intended use. For our purposes, "intended use" equates to the support of maintenance activities.

In general documents determined to be inaccurate and/or inadequate are typically characterized as possessing erroneous, or insufficient. More specifically, erroneous documents are inaccurate, while inadequate documents omit or only partially convey critical information. Unfortunately, current ADDS documents exhibit both erroneous and inadequate characteristics. The following three subsections discuss the accuracy and adequacy of ADDS documents relative to the conveyance of *erroneous* and *insufficient* information.

2.2.1 Erroneous Information

Erroneous information can be characterized by the unintentional deviation from truth or accuracy. With respect to documents generated by ADDS, this "deviation from truth" occurs because (1) the information database does not reflect the most current software configuration, and (2) documents are synthesized from database relationships that, although technically correct, imply non-existent relationships. Both deficiencies are discussed below.

The Continued Use of an Obsolete Database

When a system baseline is first established, an information database is built that reflects the relationships among system software objects. This database supports PSL/PSA information requests from which ADDS documents are generated. As system maintenance occurs, baseline source code is modified, and new versions of the executable code are distributed to the fleet. Missing from this activity, however, are corresponding updates to the information database reflecting the source code modifications. Consequently, as modifications to the baseline continue, reports generated from the initially established baseline quickly become obsolete and inaccurate.

Since the current database always reflects a system configuration based on the initial baseline, and because no facility is readily accessible for cross-referencing maintenance activities with the initial baseline configuration, the regeneration of ADDS documents is also futile. Simply stated, these documents become increasingly inaccurate and erroneous as the maintenance activity continues.

The Misrepresentation of PSL/PSA Object Relationships

An additional source of erroneous information stems from the misuse or misstatement of object relationships defined in the ADDS database. In particular, documents like the Set/Used report misrepresent variable/procedure relationships because the *overlay* feature of CMS-2 inadvertently introduces variable aliasing. Overlaying is a feature of the CMS-2 language that provides for the redefinition and renaming of memory locations common to multiple procedures. As illustrated in Figure 1, memory locations 100 through 105 are defined in procedure A as item V. Procedures B and C redefine (and repartition) the same set of memory locations as W/X and Y/Z, respectively. This redefinition allows one to reference the *same* memory location(s)

- through different *names*, and
- based on different *characteristics*.

For example, memory locations 104 and 105 are accessed through a reference to item V in procedure A, to item X in procedure B, and to item Z in procedure C.

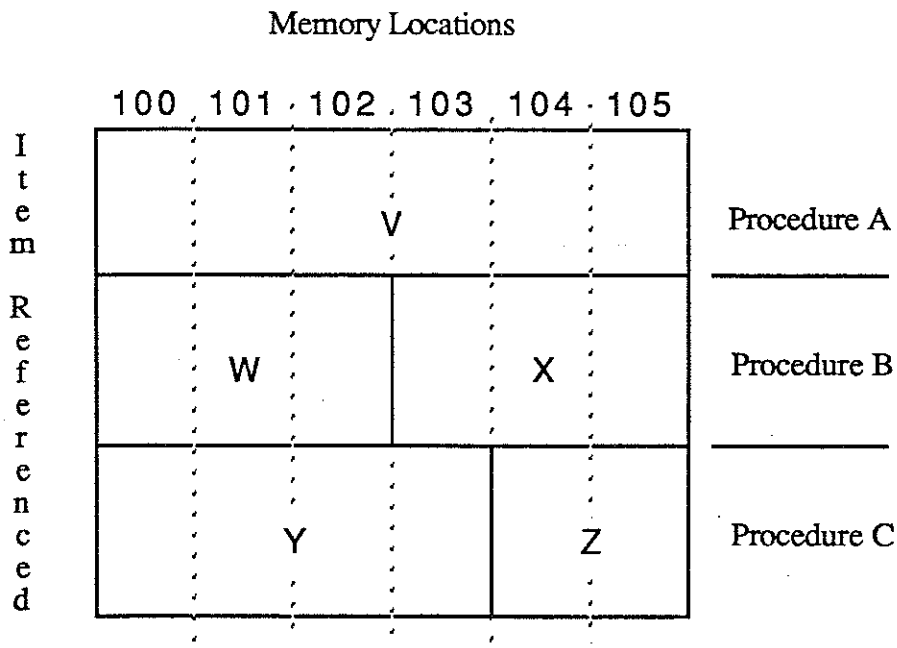


Figure 1: Redefinition Through Overlays

The problem in the generated documentation is that *all* of the names mapped to the same or overlapping memory locations are often reported when only one applies. For example, the Set/Used report will state that items V, X and Z are all used by procedure A. This type of misinformation complicates the maintenance activity and is clearly erroneous. If such information is to remain, then usage information should be qualified to state that

- item V is *directly* set/used, and that
- item V is *indirectly* accessed through references to items W,X in procedure B and items Y,Z in procedure C.

The first qualification is always needed. Reference to the second qualified set is crucial if procedures A,B and C execute concurrently or share information through common memory locations.

2.2.2 Insufficient Information

Insufficiency of information can be manifested in omissions and inadequacies. Inadequate information is characterized by documents that convey only parts of the necessary or desired material. That is, inadequate documents are often deficient in detail. Such documents are usually difficult to understand and can lead to disastrous misinterpretations. The following subsections illustrate and discuss several aspects of ADDS-generated documents that indicate various levels of inadequacy.

2.2.2.1 Inadequate Information

The Need for Meaningful Titles and Descriptions

Meaningful titles and descriptions significantly contribute to understanding the intent and purpose of a document. Ideally, a document begins with an informative title followed by a description of its purpose and content. For documents that are voluminous or provide repetitive information, page headers displaying the document title are often helpful. Several ADDS reports, however, do not utilize this feature. For example, a description of the Set/Used report only states that:

> For each data item in the C/D Baseline 2 computer program, this report presents the tactical module procedures, ADEP modules, common service routines, and channel programs which set (update) or use (employ) it. The data items are listed in alphabetical order [NSWC4].

In addition to needing greater detail, ambiguous (or confusing) terms like set/used, update/employ should either be avoided or explained. In particular, a user might want to know the relationship

between update and employ. Another report that ignores the above guidelines is the System Message List [NSWC6]. This report lacks any descriptive synopsis.

The Power of Contextual Information

The appropriate use of contextual information is another measure of the adequacy of information. Contextual information provides a reference point through which one gains perspective. Contextual information (and perspective) is especially critical to documents that list volumes of repetitive data. For example, if one is enumerating all variables within a set of procedures, appropriately placed procedure names provide context, and subsequently, perspective. An example of an ADDS document needing contextual information is the Procedure Calling Hierarchies reports [NSWC2]. A representative excerpt from this report appears below:

```
1 CALERR
        2 CALIOE
                3 CCSXX2A-C4

Level Count   Level Count   Level Count
  1     1       2     1       3     1
```

Certainly a relevant question is: *In what module or SYS-PROC do these procedures reside?* Implicit in this question is yet another: *In moving from one procedure to another, have SYS-PROC or module boundaries been crossed?* Contextual information can provide answers to such questions and significantly facilitate the maintenance activity.

The Avoidance of Confusing (Database) Terminology

The users of ADDS documentation maintain CMS-2 code and are familiar with terms and expressions inherent to the CMS-2 language. Similarly, the same set of users focus their activities around understanding and maintaining the various functional elements of the AEGIS software system. Again, the users are comfortable with terms related to those functions. In concert with the above, maintenance documents are most appropriately worded with terms and phrases reflecting CMS-2 and/or system element applications. In several instances, however, ADDS documents employ PSL/PSA terminologies in critical places, and subsequently, lead to confusion. For example, in the Design Description Document (Volume 1) [NSWC1] the PSL/PSA terms *employs* and *utilizes* are found. Based on the excerpt below, one might ask the following questions. *Is there any difference between employs and utilizes? If so, are the users aware of the distinction?*

```
DEFINE PROCESS   CAL12A;
      •
   EMPLOYS:      CTCALTNG/DELETE-C/D, CTCALTNG/XPOS-C/D,
      •
   UTILIZES:     CALOUT
```

A similar example can be seen found in the Set/Used report [NSWC4] with respect to the terms *employed* and *updated.*

```
DEFINE ELEMENT:  CALAAMP-C/D;
   UPDATED BY:   CAL823, CAL86B, CALPOS, CALSUB,
                 CALVID, CAM1DC, CAM5DC, CAMPUS;
   EMPLOYED BY:  CALAMP, CALM6B, CAM5DC;
```

For one who is not familiar with the PSL/PSA terminology the following questions are appropriate. *What is the relationship between the employed and updated terms? Is one a subset of the other?* Another logical question, and one conveying confusion, might be: *If a data item is updated by a routine, then should it not also be employed by that routine?* The above illustration appears to contradict this logical assumption.

Finally, a third illustration, extracted from the Common Database Structure report [NSWC3], shows PSL/PSA terms and relationships being used as supplemental information.

```
1 CALB21P1_LOCDD-C/D          SET
  2 CALTCSRW-C/D              ENTITY (COLLECTION OF)
    3 CALTCSRW/YA16S6-C/D     ELEMENT (CONSISTS OF)
```

Note the use of PSL/PSA object names: SET, ENTITY and ELEMENT, and the use of PSL/PSA relationship phrases: COLLECTION OF and CONSISTS OF. Although these terms and phrases are intended to convey relationships among CMS-2 objects, confusion and misunderstanding can occur if the user does not understand the definitional differences among SET, ENTITY and ELEMENT as well as the semantic difference between COLLECTION OF and CONSISTS OF. Ideally, users should not have to translate between CMS-2 and PSL/PSA terminology. The ADDS reports should exploit the terminology familiar to the user. PSL/PSA terminology should be used only when it is known to be understood by the document user, and then, only when it augments or complements existing information.

The Documentation of Design Decisions

Based on the focus of a document, certain categories of information are expected to be present. In particular, documents describing the design of software components should reflect:

- the decisions that are made in arriving at a final design,
- the underlying assumptions made, and
- alternate designs considered.

Such information is particularly helpful to maintenance personnel. For example, new personnel can more easily learn the steps in the development process, all maintenance personnel can see the progression of thought that is particular to the given code, and finally, developers are kept apprised of the underlying assumptions that might affect decisions concerning new features or enhancements. Additionally, knowledge about alternate designs and implications of their incorporation can have significant impact if current underlying assumptions change. In particular, a design alternative previously considered might be the better alternative under new conditions.

Relative to the above discussion, the Data Base Design Document (DBDD) and the Program Description Document (PDD) lack crucial information concerning current design rationale, basic assumptions, and alternate design considerations. This situation arises because critical portions of the DBDD and PDD are automatically generated from the AEGIS source code by the AEGIS Source Code Preprocessor (ASCP). We note that the automatic generation capability does not cause the loss of desirable (and crucial) information but rather the problem lies with the (limited) availability of information. That is, the AEGIS source code exhibits characteristics and design qualities of "what is", excluding past design decisions and assumptions.

Accessibility to Pertinent Information

Finally, an item that indirectly contributes to the inadequacy of information concerns user accessibility to the information base. Beyond standard reports, users often have to leave their place of work and travel to another floor or even to another building to access desired information. This is especially true for users wanting access to the interactive ADDS facilities currently available. In essence, the desired information is available, but facilities providing access to that information through the on-line ADDS system are inadequate.

2.2.2.2 Omitted Information

Less-suitable but more restrictive than erroneous or inadequate information, is the exclusion of required knowledge about a subject , i.e. omitted information. The following subsections discuss several instances of omitted information and indicate, through examples, the synthesis of corresponding reports based on information *currently* available to ADDS.

The "Affecting" Relationship Among Variables

An important aspect of the maintenance activity is the identification of a set of variables that can affect the value of another variable. This type of information is particularly important if one has isolated a software problem to a specific statement and variable within that statement. In [WEIM84], Mark Weiser discusses the technique of "program slicing" and illustrates how this technique can provide the above-mentioned information. For example, suppose one is given the

following code segment:

```
(1)    a := b + c
(2)    if (a > 0) then
(3)        d := a + 1
(4)    else
(5)        e := 55.3
(6)    f := sin (g + a)
(7)    h := d + 5.
```

One format that illustrates the affect of one variable on another might be:

Variable Affected	Variable(s) Affecting
a	b, c
b	----
c	----
d	a, b, c
e	----
f	g, a, b, c
g	----
h	d, a, b, c

Based on this type of information, a person performing maintenance activities can trace backwards through code examining only those statements and variables that impact the identified problem statement. In other words, if one determines that the variable *h* is being assigned an incorrect variable at statement (7), then accordingly, only statements (3) and (1) require investigation.

The above report format is for illustrative purposes only. More informative report formats might be based on the reporting of variables (and corresponding line numbers) affecting a specific variable on a selected source line statement. Because the AEGIS source code is currently included

in the ADDS information base, reports like the above are possible. An expanded discussion on "program slicing" as applied to AEGIS source code is presented later in this report.

Globally Accessible Data

Much of the AEGIS system design and corresponding software development is predicated on communication through global variables. For such a crucial aspect, however, no reports reflecting the usage of globally accessible information is available to maintenance personnel. A major consideration in maintaining software is the impact of changing one software component and inadvertently affecting another component. This problem is well recognized and so pervasive that it has been given the term: "ripple effect". For example, consider the two following procedure code segments, both of which are assumed to access the global variable *a*.

Procedure 1

```
•
if a <= 0 then
  a := 1;
•
```

Procedure 2

```
•
y := b / a

•
```

Now, suppose the requirement specifications for Procedure 1 change such that *a* can assume a value of zero (0), e.g. the statement, a := 1, is changed to a := 0. The obvious impact on Procedure 2, i.e. division by zero, must be recognized and addressed. Clearly, information relating procedure (and module) access to globally defined variables can provide the additional relationships necessary to minimize detrimental ripple effects.

The Decoding of Variable Names

Within the AEGIS domain, conventions for creating variable names, systems names, and so forth are standardized and well defined. These conventions are documented in Section 3.3 of the *CMS-2 Coding and Commenting Standards* [NSWC5]. More specifically, the standards define subfields within a name and dictate the contents of each field based on the intended usage and characteristics associated with the name. For example, suppose the standards dictate the following naming convention for variables:

field 1	=>	system name	3 spaces
field 2	=>	bay name	3 spaces
field 3	=>	variable type (l, g, t - local, global or temporary)	1 space

Correspondingly, the variable MBRS2BL would be decoded as a variable used in the Missions Broadcast Remote (MBR) system, located in the Standard Second Bay (S2B), and that it is a Local variable (L).

Although no examined ADDS documents currently appear to exploit this additional source of information, the intrinsic value is readily apparent. The value of decoded names is particularly important to inexperienced maintenance personnel or to personnel who are examining a system for the first time. Decoded variable names might be appropriately included in current reports (for contextual information) or perhaps in a quick reference type of report.

Cross-Referencing Variable Definitions and Corresponding Declarations

Within the AEGIS software development process, the Common Database approach supports the centralized *definition* of program variables. Software modules that need access to such variables

"include" the appropriate dataset (or SYS-DD). The inclusion of a dataset in a software module constitutes the *declaration* of all variables defined within the included dataset. During maintenance activities, knowledge as to

- which dataset contains the definition of specific variables, and
- what datasets are included in which procedures, SYS-PROCS and Modules

are helpful in performing modifications, assessing ripple effects caused software changes, and so forth. Subsequently, when working on a problem, maintenance personnel should not need to search through source files to find where a variable is defined or declared. Such information should be available through a standard reference guide. In support of this need, ADDS should generate an alphabetic listing of all variables (or a user designated subset) and where they are defined and/or declared.

3.0 Improvements to the Automated Design Description System (ADDS)

This section discusses improvements that can be made to the current ADDS system. These improvements focus on correcting deficiencies discussed in the previous section and are divided into three major categories:

- improvements based on the *existing* ADDS configuration,
- improvements requiring only *minor changes* to ADDS, and
- improvements necessitating *major modifications* to ADDS.

Each of these categories is described more fully in the following subsections.

3.1 Improvements Based on the Existing ADDS Configuration

This section describes several ways that the documentation currently being generated by ADDS can be improved *without* changing the existing system. The modifications suggested below entail the alteration of existing reports and the synthesis of new ones. The suggested changes to existing reports are intended to enhance their usability. The generation of new reports focuses on filling a perceived information gap. All improvements described in this section can be implemented within the current ADDS system.

Globally Accessible Data

As mentioned in Section 2.2.3, much of the AEGIS system design and communication philosophy is predicated on the use of globally accessible data. In this regard, a new report that highlights global variable access is desired. Problems created by global access to variables can be obscure and, if not detected, can effect disastrous consequences. Based on a specified set of procedures, SYS-PROCS, and/or modules, this new report should list global variables accessed, indicate the accessing component and convey how the global variables are being used, e.g. non-modification reference or update. Again, consider the following code:

Procedure 1	**Procedure 2**
•	•
if a <= 0 then	y := b / a
a := 1;	
•	•

With respect to Procedures 1 and 2, assuming that *a* is a global variable, the following information is appropriate:

Global Variable Ref'd	Referencing Procedure	Referencing Method	Line Number
a	Procedure 1	Reference only	(5)
		Update	(6)
	Procedure 2	Reference only	(5)
•	•	•	•

The Decoding of Variable Names

Because variable and system names follow a predefined set of definitional guidelines, .i.e. those defined in the CMS-2 Coding and Commenting Standards [NSWC5], and because an understanding based on the decoding of a name can provide insights as to its functionality, a report listing *specified* names and their decoded attributes is useful. Such a report might assume the following format:

Variable Name	Code		Interpretation	
MBRS2BL	MBR	=>	System:	Mission Broadcast Remote
	S2B	=>	Bay:	Standard Second Bay
	L	=>	Variable Type:	Local

Cross-Referencing Variable Definitions and their Corresponding Declarations

A document that provides information indicating where variables are defined and who has declared them for usage can assist maintenance personnel in (a) locating the appropriate dataset

(SYS-DD or LOC-DD) for gaining access to a variable and its definition, and (b) identifying all procedures and/or SYS-PROCS currently using selected variables. Because the number of variables defined within the Common Database is extensive, the capability for users to specify a selected set of variables is advisable.

Based on a specified set of variables, an alphabetized listing similar to the following might be appropriate.

Variable	Defining Dataset	Declaring Item
AA	SYS-DD-ABC	Procedure X
		Procedure A
		SYS-PROC-Z
B	SYS-DD-XX	Module DDD
		Procedure M

In formatting such a report, however, one must insure that exact meanings are known. For example, one might ask: *Is Module DDD's appearance in the list because Procedure M is defined in Module DDD and Procedure M actually includes SYS-DD-XX?* The precise conditions under which a declaring item can appear must be clearly and unambiguously stated in the document description.

The Need for Meaningful Titles and Descriptions

A needed improvement to current documentation generated by ADDS is the integration of meaningful titles and descriptions. One report, [NSWC1], contains descriptive information that introduces a series of documents. Although some description is better that none, the descriptions provided are terse and *lack substantial detail.* The authors assume that because [NSWC1] briefly outlines each report generated by ADDS, i.e. [NSWC1, NSWC2, NSWC3, NSWC4, NSWC5,

NSWC6], descriptions were necessary within each report. Our opinion, however, is that every report should contain a description outlining its intended use and functionality. Moreover, because the users of the documents are primarily maintenance personnel, the document description should reflect a corresponding perspective. Additionally, the title of the report should appear on every page; and, for documents reporting information about multiple data items, the current data item should also be listed at the top of the page.

The Power of Contextual Information

Finally, certain contextual information should be included in the ADDS reports. For example, in the Procedure Calling Hierarchy report [NSWC2] contextual information indicating the lexical scope of a procedure relative to its including SYS-PROC and Module can add perspective to the interpretation of the report. Such information is available, illustrates the multi-dimensional relationship that actually exists among procedures, and consequently, should be provided to the users. As an example, an excerpt from a reformatted page of the Calling Procedure Hierarchy report [NSWC2] might look something like the following:

Calling Hierarchy	**Including**	
	Module	**SYS-PROC**
1 CALERR	AAAA	BBBB
2 CALIOE		
3 CCSXX2A-C4		CCCC

Level	Count	Level	Count	Level	Count
1	1	2	1	3	1

The above excerpt indicates that Procedure CALERR of SYS-PROC BBBB in Module AAAA calls CALIOE in the same SYS-PROC and Module. CALIOE, however, calls CCSXX2A-C4 in SYS-PROC CCCC, but still within the same module as CALIOE. Note also that the contextual information is printed only as change occurs. Such actions are deemed appropriate because the contextual information, after all, plays the secondary role.

Summary of Improvements Under The Current System

The improvements described in the above section address several of the deficiencies outlined in the Section 2.2. These improvements, however, do not resolve all of the deficiencies. Within the first problem category, erroneous information, neither of the problems are resolved. In relation to the second problem category, inadequate information, proposals for removing the first two deficiencies are discussed. This involves

- the use of meaningful titles and descriptions in all reports, and
- the inclusion of additional contextual information.

Under the third problem category, omitted information, suggestions for resolving all but one of the deficiencies are discussed. Effectively, the following reports need to be added to the ADDS documentation:

- a report describing what variables affects what other variables,
- a report about what procedures access global variables,
- a report on decoded variable names, and
- a cross-reference report listing where variables are defined and/or declared.

In addition to removing several deficiencies of the current documentation, the proposed changes will also have a beneficial effect on the existing positive system characteristics. In particular, the additions of contextual information, reports for variables affecting other variables, variable name decoding and define/declare information can contribute toward enhancing traceability.

3.2 Minor System Enhancements

The following subsections discuss improvements to ADDS that necessitate minor modifications and enhancements to the existing system. The types of system modifications include the integration of new software products and an expansion of the database query/retrieval language.

Program Slicing Through Syntactic Analysis

Section 2.2.3 provides an introduction to the concept of "program slicing" [WEIM84] and illustrates how that technique is used to support the maintenance activity. The term, program slicing, comes from the conceptual partitioning (or slicing) of a programs based on the set of statements that can effect the value of a variable. Consider again the example given in Section 2.2.3 and our quest for determining all variables that affect the value of variable *h*.

Program Statements	Program Slice	Variables Affecting Value of *h*
(1) a := b + c	(1) a := b + c	b,c
(2) if (a > 0) then		
(3) d := a + 1	(3) d := a + 1	a
(4) else		
(5) e := 55.3		
(6) f := sin (g + a)		
(7) h := d + 5	(7) h := d + 5	d

From the "slice" one sees that in statement 7 variable *d* contributes to the value of *h*. In turn the value of *d*, and indirectly the value of *h*, is affected by the value of *a* in Statement (3). Moreover, the variable *a* is a function of variables *b* and *c*. Effectively, the variable *h* is directly and indirectly affected by variables *a,b,c* and *d*.

The synthesis of slicing information and the generation of a report similar to the above can be achieved in two ways:

- the "in-house" construction of a source code analyzer whose semantic actions extract the pertinent information , or
- through modifications to the CMS-2 compiler.

An in-house analyzer, similar to the one currently being investigated by Bob Brown at CSC, should be considered first because (a) NSWC will have direct control over the analyzer and (b) versions of the syntax analyzer can easily be generated to perform other forms of source code analysis helpful to the maintenance activities, e.g. static analysis [ARTJ81] and symbolic execution [CLAL85].

The Inclusion of ZyRECORDS

ZyRECORDS is a product of ZyLAB Corporation (Chicago, IL) and supports the extraction of information based on *record* structures. ZyRECORDS is similar to ZyINDEX (also a product of ZyLAB) in that textual search patterns can be specified, but where the search patterns are confined to specified record structures. The power of ZyRECORDS comes from its ability to select textual *segments* based on qualified record structures. For example, structured segments of source code can be selectively retrieved based on specified record format *and* contextual information. That is,

suppose one desires a listing of all SYS-PROCS that include the definition of the dataset SYS-DD XXX. One requests a search on the source code, restricted to SYS-PROCS (whose structure has been predefined to ZyRECORDS) and containing the string "SYS-DD XXX". Based on this specification, the corresponding report should extract all appropriately attributed SYS-PROCS *and* the encompassing declarations, definitions and procedures.

The capabilities of ZyRECORDS can also be applied to document analysis. For example, document abstract, table of contents, and section formats can be defined as structured records with fields. Accordingly, the user can then extract logically *complete* units of text based on structure and/or based on qualified string patterns within fields or records. For example, the user might select all paragraphs that reference a particular named function.

Because ZyRECORDS is tool that has been developed to work in tandem with ZyINDEX, a product already included in ADDS, the integration of ZyRECORDS should require only minor modifications. More specifically, the ZyRECORDS enhancement is a capability best suited for integration into the Micro-ADDS component of ADDS.

An Expansion of PSL/PSA Capabilities

The last minor enhancement to ADDS involves modifications to the PSL/PSA definition and query system. In particular, new capabilities should be included that provide for

(a) the renaming of current PSL/PSA objects and relationships, and
(b) an expansion of the number of objects and relationships currently definable within the PSL/PSA system.

These changes will allow the database managers to appropriately define objects and relationships consistent with CMS-2 and element terminologies. In turn, such changes will promote the synthesis of reports that more precisely convey the *actual* objects and relationships among components fundamental to AEGIS systems, and will significantly enhance document readability and understandability. The suggested changes can also be applied to the Overlay problem discussed in Section 2.2.1. That is, PSL/PSA relationships can be defined to capture variable aliasing and to recognize multiple definitions of the same memory location.

Summary of Improvements based on Minor Enhancements

The minor enhancements suggested above resolve several deficiencies discuss in Section 2.2. First, the expansion of the PSL/PSA capabilities provides for the resolution of

- the Overlay problem discussed under Erroneous Information (Section 2.2.1), and
- the use of confusing database terminologies as presented in Section 2.2.2.

In particular, modifications to PSL/PSA will promote the use of terminologies consistent with objects and relationships inherent to the CMS-2 language, and subsequently, provide for the synthesis and conveyance of variable name aliasing and the "overlaying" of memory locations.

The minor improvements suggested above are also expected to further enhance several positive aspects of ADDS. First, both the ZyRECORDS and the PSL/PSA enhancements improve traceability. ZyRECORDS can be used in conjunction with ZyINDEX to relate objects in both documents and the AEGIS source code. Through the PSL/PSA enhancements, consistent terminology can be used to define and select objects and relationships throughout all major (and minor) functions of the AEGIS system. Second, ZyRECORDS contribute to the flexibility of

ADDS because record structures can be defined for both source code and document text, which in turn, provides for the extraction of local as well as cross-referenced information.

3.3 Major System Modifications

The third category of improvements requires substantial system modifications. The necessity and perceived benefits of such changes, however, justify the proposed modifications. In addition to addressing one major deficiency of the current ADDS system, the proposed changes also provide for new levels of information acquisition and dissemination.

The Synthesis of Current Documentation

The first major change proposed for ADDS focuses on the fundamental need for maintaining documentation that reflects both the established baseline and applied modifications. Currently, the ADDS database captures only objects and relationships defined by the initial system baseline, and is not changed until the next major baseline is established. Subsequently, as modifications are applied to the current baseline, the ADDS database and documentation become outdated. Both the database and documentation should always reflect the currently "accepted" system, where "accepted" implies the system defined by the current baseline with integrated modifications.

The ability to produce current documents can be achieved in several ways. First, the ADDS database can be rebuilt at specified intervals and new documents generated. There are, however, the issues of (a) computer time necessary to rebuild the database and regenerate reports, and (b) the manpower needed to distribute new reports and collect old ones. Second, a secondary database can be built which contains changes made to the system baseline since the last generation of the primary ADDS database. Obviously, the report generation routines would have to be modified to access and consider information in both databases before synthesizing a document. In addition to

complications that might be introduced through major baseline changes, such an approach has the potential side-effect of slowing down response times for interactive queries and report synthesis. The third approach to maintaining current documentation involves the continual updating of the ADDS database to reflect each and every modification to the system baseline. This approach might be achieved through (a) rebuilding parts of the database affected by the change, or (b) the setting up an indirection mechanism (like pointers) that "effect" database modifications.

In selecting any of the above approaches (or perhaps a hybrid of the above), one must also consider the following consequential issues:

- access must be given to computational facilities that have at their disposal the requisite information for capturing baseline modifications and integrating those modifications into the ADDS database,
- such facilities must commit the manpower and computing resources required for the *periodic* task of updating the ADDS database, and
- additional computing facilities and manpower must be secured for generating and distributing the new documents.

Finally, based on the approach selected,

- one must determine the priority placed on interactive query processing, and the adverse impact secondary databases and pointer systems can have on the information retrieval process.

Software Configuration Management System

The second major change is directed more towards enhancing the maintenance support environment rather than the Automated Design Description System. ADDS does, however, benefit indirectly. The suggested change is the integration of a Software Configuration Management System (SCMS) into the environment that currently supports maintenance activities. Such a system can benefit the maintenance activities in two ways. First, benefits can be directly derived from added capabilities like source code revision control and the automatic tracking of error reports and modification requests. Second, ADDS can and should be modified to produce reports that incorporate information from *both* the existing ADDS database and the new information base generated by SCMS.

The selected Software Configuration Management System should provide

- a Source Code Control System (SCCS) [ROCM75,TICW85],
- a Module Interconnection Language (MIL) and corresponding description system [PRIR82,TICW79],
- a subsystem loading language [FELS79] and
- a Modification Report (MR) system [KNUD76].

These four elements, individually discussed below, encapsulate the primary virtues of an SCMS: version control, interconnection specification, system building and maintenance tracking.

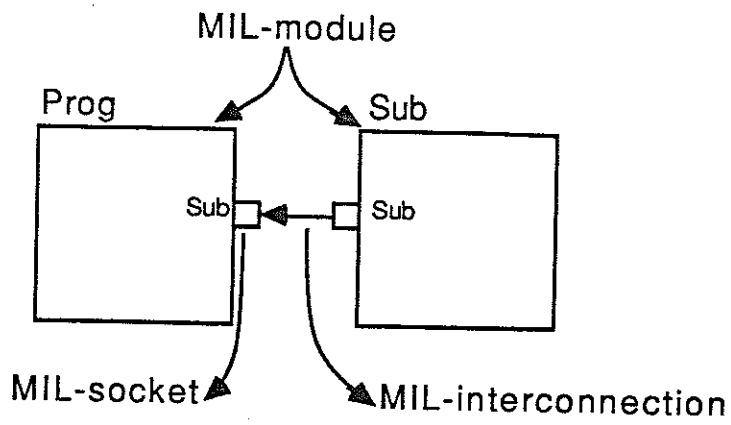
Fundamental to any Software Configuration Management System is a Source Code Control System. The primary task of an SCCS is to track and maintain changes to source code and documentation. That is, an SCCS automatically records all source code and corresponding

document modifications. From a maintenance perspective, an SCCS provides the ability to (a) track program changes, (b) determine who made the changes, why, and when, and (c) reconstruct earlier versions of software components. Such capabilities are highly desirable when performing maintenance activities and can have a significant, positive impact on personnel productivity. Although modifications to the AEGIS system are currently being recorded, that process is manual, time consuming, and error prone.

Another element needed in the Software Configuration Management System is a Module Interconnection Language (MIL). The development of MILs has been directed toward defining the conceptual framework of how system modules interface. In particular:

> "a MIL is a language for describing software definition module structure. Its main function is to establish the accessibility of resource names, i.e. identifiers, procedure names, type names, etc., among modules, and to assist in binding resource names to the modules which provide and need those resources [PENM79]."

As illustrated below, one utilizes three MIL components in describing relationships between two modules: *the MIL-module,* the *MIL-socket* and the *MIL-interconnection.*



A MIL-module represents the definition of a module or data structure. The name of any given MIL-module is made visible through the use of a MIL-socket. The MIL-socket depicts the interface between a MIL-module and its environment, and describes pertinent interface characteristics. The MIL-interconnection is a directed arc connecting two sockets and represents the access to a MIL-module and the direction of access. In the above example, Sub is a resource that is accessible to Prog [PNEM79].

Relative to maintenance activities, a MIL-based description of modules and their relationships to each other can provide quick access to dataflow information and pertinent interface characteristics. An added advantage is that through module interface languages the characterization of modules and module interfaces can be easily automated. In turn, such a system can provide *interactive* access to inter-module communication characteristics and interconnection validation based on a pre-defined set of connection rules.

Thirdly, the SCMS should include a "subsystem loading language". System loading languages are executable and make all of the necessary checks to build a system. For example, in [FELS79] Feldman describes the *Make* subsystem loading language found in the UNIX[1] environment. The benefit of a loading language is the speedup of building a system or part of a system. They are

[1] Unix is a trademark of AT&T.

similar to a MIL in that they work at the system level. In fact, some MILs provide loading capabilities [TICW79]. When using loading languages, if certain databases are needed before a program can be created, the loading language builds them automatically.

A major advantage in using a loading language is the realization of an automated process by which a system or application is created. In particular, concerns related to how one configures a system, which program versions to include, and where one places the resulting executable files are no longer considerations of the programmer or maintenance personnel . Such details are defined through the loading language and resolved by the system.

Finally, the SCMS should include an interactive Modification Report system. An MR system maintains a log of modification requests, fixes, enhancements, and so forth. Typically a Modification Request might contain: the name of the owner of the request, the request number, the date the request was initiated, the creator of request, the application affected by the request, the address of creator, the phone number of creator, the actual software module affected by the request, the request type, an indication as to the severity of problem, a summary of problem, a full description of problem, the request's status, deadline for resolving the request, comments pertinent to the request, and finally, a description of the how the request was addressed and resolved [KNUD76].

Additionally, an MR system can generate status reports describing who has what to do, which features have a lot of work remaining, what are the major problems outstanding, and so on. Such information is useful not only to maintenance personnel, but to managers as well. In particular, maintenance personnel can be kept apprised as to what needs to be done and with what priority. Managers, on the other hand, can access the same information and make staffing decisions relative to perceived problem difficulties, severity levels and established priorities.

Much has been written about Software Configuration Management Systems and the specialized aspects discussed above. For further information interested readers are referred to [COND86].

Summary of Major Modifications

The first improvements described in this section address a crucial deficiency discussed in Section 2.2.1, i.e. the continued use of a non-current database. Although necessitating an additional commitment to computing and manpower resources, the suggested remedies do provide for a database that reflects objects and relationships found in the system software currently used by the fleet. Accordingly, maintenance reports and documentation shall reflect such currency. The second suggested change, the integration of a Software Configuration Management System, is not aimed a correcting any particular deficiency, but simply to enhance the maintenance environment. An SCMS provides capabilities crucial to adequately supporting diverse and demanding maintenance activities. These capabilities include source code version control, a language for precisely describing module interface characteristics, a system synthesis language and a modification tracking and reporting system. In general, the modifications outlined in this section will contribute to document/code traceability, provide for more current documentation and significantly enhance the maintenance environment.

4.0 Summary and Conclusions

The primary objectives of this research effort has been to identify both the strengths and weaknesses Automated Design Description System (ADDS), and to suggest appropriate methods for resolving those deficiencies. Although several positive system and document characteristics stem from the automated approach to report generation, the same approach brings with it a myriad of deficiencies. Reflecting this duality, the suggested modifications emphasize the removal of current deficiencies while improving positive aspects of the system and documentation.

Throughout the investigation, a major emphasis has been placed on ascertaining the usability of documents and reports generated by ADDS. Several positive characteristics and qualities, e.g. traceability, completeness and consistency, are identified and discussed in Section 2.1. Section 2.2 presents perceived deficiencies and inadequacies related to the current set of documents produced through ADDS. For presentation purposes, the discussion of document deficiencies is partitioned into three major sections. This partitioning reflects undesirable document characteristics contributing to (a) erroneous information, (b) inadequate information and (c) omitted information. The suggested remedies address all but one of the perceived deficiencies and are presented in relation to the amount of modification impact they might have on the existing ADDS system. That is, in Section 3 the order of discussion is based on improvements that (a) can be integrated into the current ADDS configuration, (b) necessitate only minor changes, and (c) require major enhancements to the system.

For all deficiencies but one, a detailed description of suggested corrections are discussed. The one deficiency not addressed concerns the conveyance of software design decisions in the Program Description Document (PDD) and the Data Base Design Document (DBDD). In particular, both documents lack details relating the whys and hows of the evolution of system design decisions. This deficiency stems from the use of the AEGIS Source Code Preprocessor (ASCP) in synthesizing critical sections of the PDD and DBDD. The automated process of extracting information is not questioned, but instead, the legitimacy of using the AEGIS source code as the database. We hypothesize that such a database primarily reflects " what is" , not past decisions leading to the current system configuration.

In summary, several usability elements *are* missing from the documents generated by ADDS. This study suggests several report modifications and system enhancements that can enhance the correctness and usability of the synthesized documents. An integral part of instituting improved

document quality and an enhanced Automated Design Description System, however, is addressing and resolving the critical issue of system *purpose*. That is, the decision of whether ADDS is to play an active or passive role in supporting maintenance activities must be made. As detailed in the companion report, *Automated Design Description System (A Synopsis Based on User Interviews)*, the purpose of the system is a major factor in selecting design alternatives and system modifications like the ones discussed in this report.

5.0 Future Research Directions

The system and document enhancements suggested in this report are intended to raise the quality and usability of information generated from ADDS. Determining whether is does, however, can only be substantiated through document quality assessment. In general, the analysis of documentation is difficult because of its unstructured nature. Nonetheless, preliminary research findings support evidence that Document Quality Indicators (DQIs) exist and can be used in an assessment process focused on determining document quality. A document quality indicator (or DQI) is defined to be a variable whose value can be determined through direct analysis of a document characteristic and whose evidential relationship to one or more quality factor(s) is undeniable. For example, the understandability of a document is directly related to its readability. Readability is measurable through the direct analysis of sentence structure. Hence, if one desires an indication as to the understandability of a document, one measure might be based on the analysis of sentence structures.

Clearly, sentence structure is only one of many factors contributing to document understandability. This realization, along with the accepted importance of quality documentation, motivates a future investigation focusing on the identification, synthesis and exploitation of document quality indicators. In the identification process, issues regarding characteristic composition must be addressed. That is, what are the identifiable characteristics of DQIs. The

synthesis process entails relating DQIs to qualities like readability and comprehensibility. The proposed relationships, however, must be undeniable. The exploitation of DQIs involves the formulation of reliable and valid metrics, that is, metrics that are *consistent* in their measured results and accurately *reflect* the quality or property being assessed. Additional questions that must also be resolved include the following:

"Do the intended user of a document, its purpose and focus affect underlying DQIs?"

"Can the assessment of DQIs be automated?"

An investigation of the issues mentioned above, and a search for answers to the above questions is currently underway.

References

[ARTJ81] Arthur, J. and Ramanathan, J., "Design of Analyzers for Selective Program Analysis," *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 1, January 1981, pp. 39-59.

[BOEB79] Boehm, B., "Software Engineering - As It IS," Proceeding of the 4th International Conference on Software Engineering, September 1979, Munich, Germany, pp. 11-21.

[BOEB76] Boehm, B., "Software Engineering," IEEE Transactions on Computers, Vol. C-25, December 1976, pp. 1226-1241.

[CLAL85] Clark, L. and Richardson, D., "Applications of Symbolic Evaluation," *The Journal of Systems and Software*, Vol. 5, NO. 17, February 1985, pp. 15-35.

[COND86] Conde, D., "Bibliography on Version Control and Configuration Management," *ACM SIGSOFT Software Engineering Notes*, Vol. 11, No. 3, July 1986, pp. 81-84.

[FELS79] Feldman, S., "Make - A Program for Maintaining Computer Programs," *Software - Practice and Experience*, Vol. 9, No. 4, April 1979, pp. 255-265.

[KNUD76] Knudsen, D., Barofsky, A. and Satz, L., "A Modification Request Control System," *Proceedings of the 2nd International Conference on Software Engineering*, October 1976, San Francisco, CA, pp. 187-192.

[LIEB78] Lientz, B., "Issues in Software Maintenance," *ACM Computing Surveys*, Vol. 15, No. 3, Sept 1983, pp. 271-278.

[NSWC1] CND B/L 2 Design Description Document, Volume 1.

[NSWC2] CND B/L 2 Design Description Document, Volume 2.

[NSWC3] CND B/L 2 Design Description Document, Volume 3.

[NSWC4] CND B/L 2 Design Description Document, Volume 4.

[NSWC5] Computer Programming Standards, CPS-09, May 1987.

[NSWC6] ADS Baseline 3 Phase 2 System Message List.

[PNEM79] Penedo, M. and Berry, D., "the Use of a Module Interconnection Language in the SARA System Design Methodology," *Proceedings of the 4th International Conference on Software Engineering*, September 1979, Munich, Germany, pp.294-307.

[PRIR82] Prieto-Diaz, R. and Neighbors, J., "Module Interconnection Languages: A Survey," Technical Report, Department of Computer Science, University of California Irvine, August 1982.

[ROCM75] Rochkind, M., "The Source Code Control System," *IEEE Transaction on Software Engineering*, Vol. SE-1, No. 4, December 1975, pp. 364-370.

[TAUR77] Tuasworthe, R., *Standardized Development of Computer Software*, 1977, Prentice-Hall, Englewood Cliffs, NJ.

[TICW85] Tichy, W., "RCS - A System for Version Control," *Software - Practice and Experience*, Vol. 15, No. 7, July 1985, pp. 637-654.

[TICW79] Tichy, W., "Software Development Control Based on Module Interconnection," *Proceedings of the 4th International Conference on Software Engineering*, September 1979, Munich, Germany, pp. 29-41.

[WASA76] Wasserman, A., "On the Meaning of Discipline in Software Design and Development," In: P. Freeman and A. Wasserman (eds), *Tutorial on Software Design Techniques*, IEEE Computer Society, Long Beach CA.

[WEIM84] Weiser, M., "Program Slicing," *IEEE Transaction on Software Engineering*, Vol. SE-16, No. 4, July 1984, pp. 352-357.

PRELIMINARY

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Limited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Systems Research Center SRC 88-002 Blacksburg, Virginia 24061	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Systems Research Center 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Surface Warfare Center
6c. ADDRESS (City, State, and ZIP Code) Virginia Tech 320 Femoyer Hall Blacksburg, Virginia 24061	7b. ADDRESS (City, State, and ZIP Code) Dahlgren, Virginia 22448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Surface Warfare Center 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N60921-83-G-A165 B0034
8c. ADDRESS (City, State, and ZIP Code) Dahlgren, Virginia 22448	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Prospects for Automated Documentation Analysis in Support of Software Quality Assurance

12. PERSONAL AUTHOR(S)
James D. Arthur, Richard E. Nance and K. Todd Stevens

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM 6/15/87 TO 1/25/88	January 1988	44

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

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

This report discusses the strengths and weaknesses of ADDS. ADDS is an automated document synthesis system that exploits the benefits of reverse engineering. Its generated documents are primarily used to support the maintenance activity. Based on the content and format of these reports recommendations are made to improve the overall effectiveness of ADDS. These recommendations include the modifications/addition of reports and corresponding changes to ADDS.

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

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE