# The Evaluation of Software Quality: An Empirical Approach to Validation*

*James D. Arthur and Richard E. Nance*

**TR 92-51**

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

October 18, 1992

**Cross-listed as Systems Research Center report SRC-92-006.*

Technical Report SRC-92-006*

# The Evaluation of Software Quality: An Empirical Approach to Validation

## An Interim Report for Period 1 February – 15 June 1992

*James D. Arthur*
*Richard E. Nance*

Systems Research Center
and
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0251

July 1992

* Work supported by the Joint Logistic Commanders, Computer Resource Management Group through the Systems Research Center under Basic Ordering Agreement N60921-89-D-A293

Cross referenced as Technical Report TR-92-51, Department of Computer Science, Virginia Tech

## ABSTRACT

This interim report outlines activities related to the Software Quality Assessment Project funded by the JLC/CRM. Reported activities reflect investigative efforts performed at the Systems Research Center (Virginia Tech) and at the Melpar Division of E-Systems. Accordingly, this report discusses (1) the development and installation of a document analyzer, (2) the development and refinement of document quality and process indicators, and (3) our approach to data collection. An outline of planned activities for the next reporting period is also presented.



**CR Categories and Subject Descriptors:** D.2 **[Software Engineering]:** D.2.1 **[Requirements Specifications]**–*Methodologies and Tools;* D.2.2 **[Tools and Techniques]**–*Design, Management.*

**Key Words and Phrases:** Software quality assessment, documentation analyzer, document quality indicators, process indicators, software tools, software development methodologies, validation.

# The Evaluation of Software Quality: An Empirical Approach to Validation

## 1. Introduction

This interim report outlines activities related to the Software Quality Assessment Project funded by the JLC/CRM under the Basic Ordering Agreement N60921-89-D-A293. Reported activities reflect investigative efforts performed at the Systems Research Center (Virginia Tech) and at the Melpar Division of E-Systems. Personnel conducting the investigation are Dr. Richard E. Nance (Co-Principal Investigator), Dr. James D. Arthur (Co-Principal Investigator), Edward V. Dorsey (Graduate Student) and Arcel Castillio (Graduate Student). The reporting period is 1 February through 15 June 1992.

Per CDRL C.1.4 (Task 4) of the Statement of Work we provide in this report: (1) a description of progress during the reporting period, (2) an outline of planned activities for the next reporting period, and (3) a synopsis of the current expense profile.

## 2. Progress Description

The focus of our current work has been three-fold: document analyzer development and installation, indicator development and refinement, and data collection.

### 2.1 Document Analyzer Development and Installation

The prototype document analyzer tool (DAT) is completed and now operational on site at E-Systems. The DAT executes in both SUN Unix and Apple/AUX environments. Although E-Systems has a SUN system that is compatible with the one at the Systems Research Center, we have chosen to perform document analysis on the project's MACII, also residing at E-Systems. This decision is consistent with the desire to maintain a "low-profile" at E-Systems and to minimize our reliance and impact on E-System's development facilities. Currently, DAT extracts data elements needed to compute 10 of the original 32 proposed documentation quality indicators. Additional work is being directed toward enhancing the reporting facilities of the DAT. The Users Manual, provided as Appendix A, details the operational aspects of the DAT. It also provides a synopsis of the input and output formats of the tool.

## 2.2 Indicator Development and Refinement

Indicator development and refinement proceeds in three directions:

(1) Work on the 66 code indicators is in the refinement stage. On-site test data is being collected for indicator refinement and for the purpose of verifying the correctness of the analyzer and report generator.

(2) Preliminary identification and metric definition associated with the 32 document quality indicators is complete. Development documents are being obtained from E-Systems so that indicator refinement and verification of the data extraction process can begin. We expect to begin the refinement and verification processes in July 1992.

(3) The refinement and redefinition of process indicators has been a major effort during this period. Currently, ten process indicators are being tested on-site for applicability and data availability. Appendix B provides a listing of the ten process indicators; however, the metric formulation is not provided since some remain under development, with their final form yet to be decided. Four process indicators relate OPA attribute to requirements volatility, i.e. the disruptive tendency to add, delete, and modify requirements after completion of the Software Requirements Review, at which time requirements are intended to be fixed. The claim that four attributes are adversely affected by changing requirements might seem surprising, but those who have experienced the effects on software-intensive projects are not likely to express disagreement. Cohesion, coupling, early error detection and traceability can all suffer the consequences from changing requirements. Also related to the requirements definition phase is the detection of defects later in the development process (following SRR).

The effects of personnel departures or reassignment, described in the literature as staffing instability, are grouped with other sources of uncertainty that lower the quality of software. Labeled *Development Instability*, the process indicator includes sources such as shifts in target hardware or language, changes in the software development environment, and shifts in management policy along with the personnel effect.

Two indicators reflect the ongoing use of software development folders (files) as promoting traceability and the visibility of behavior. The software quality assurance, configuration management and test activities are assessed in the indicators of the SQA infrastructure and conformance with the test model.

## 2.3 Data Collection Activities

Data is being periodically collected from two major sources: (1) BGPHES source code currently under configuration management, and (2) the software development folders (SDFs) corresponding to selected BGPHES CSCs.

Currently, three Snapshots of selected CSCs have been taken. Dates and CSC components are as follows:

| | |
|---|---|
| 8 Apr | POLO, POUT and POWI, |
| 21 May | POLO, POUT, POWI and PORE, |
| 15 Jun | POLO, POUT, POWI, PORE, PODB and POGR. |

The resulting sets of data will provide insights as to how well we can predict software quality in the early stages of the development process. Data extraction techniques have proved robust and dependable, with only slight modifications necessary in one instance to handle conditional compilation constructs.

The second major data collection effort addresses process assessment: the review of selected SDFs for validation of software requirements and the capability for meeting process indicator needs. A crucial aspect of the SDF review process is attending code walkthroughs. Code walkthroughs observed during the reporting period include:

| | | |
|---|---|---|
| 5 Feb | POUT | System Utilities |
| 13 Feb | POGR | Graphics |
| 27 Feb | POPI | PIC Interface |
| 31 Mar | PODB | POS Database |
| 14 Apr | POUT | (second walkthrough) |
| 30 Apr | ICLO | Controller Log |
| 4 Jun | PODM | MSTDF Database |

Future efforts will focus heavily on SDF reviews.

Although not explicitly related to the data collection activities, VTSRC personnel participated in the following activities to gain a better understanding of the software development process and to advance understanding of the validation project:

| | |
|---|---|
| 19 Feb | Discussion of on-site progress with CDR Moore, R. Parker, C. Garner, R. Strong and others at SPAWAR. |
| 13 Mar | Briefing to John Salasin (DARPA and SEI), John Whitehead (NAVSEA), Dr. Raghu Singh (SPAWAR), Ms. Norma Stopyra (SPAWAR) and others at CP5 in Crystal City. |
| 17 Mar | Attending Major Program Review for BGPHES XN-2 at MELPAR Headquarters in Falls Church. |
| 18 Mar | Presentation of research background and objectives of current on-site investigation to collective meeting of Software Quality Assurance Mangers for all E-Systems divisions at Tyson's Corner. |
| 29 Apr | Attending meeting on integration test at MELPAR, Fairfax, which dealt with test specifications, plans and procedures. O. von Bredow, C. Chausse, R. Strong (CSC), J. Finch, C. Sabat, N. Verna, among others in attendance. |
| 6 May | Attending meeting of the Systems Design Review Group to consider issues arising from Software Trouble Reports generated during the Integration Test builds. |
| 3 Jun | Attending SDRG Meeting. |

## 3. Planned Activities

Investigative activities are expected to focus on data collection, validation and metric refinement. In particular, we expect to place an emphasis on (1) verifying the correctness of the documentation analyzer, (2) reviewing SDFs for validation data, and (3) examining additional process documents and investigating process trends to support the acquisition of data necessary for the computation of other process indicators.

Because deployment is scheduled for the first quarter of 1993, VTSRC personnel will also begin to identify those requirements and activities necessary to effect a smooth transition of the validation effort from a development posture to one of deployment.

## 4. Current Project Expenditures

Appendix C provides a profile of project expenditures beginning February 1991 and ending June 1992. Overall, project expenditures have been less than projected. As illustrated, the most

significance discrepancy lies in the travel category. Three factors have contributed to reduced spending:

(1) On-site activities were projected to start in February 1991; the actual start date, however, was 1 October 1991.

(2) To date, only one VTSRC person has been needed on-site at any given time. As more CSCs are completed and as SDFs continue to mature, the demand for on-site personnel is expected to increase.

(3) The Virginia Tech Systems Research Center negotiated a reduced daily room rate with Comfort Inn, Fairfax.

## 5. Projected Requirements for Task Completion

Our projected work schedule and project costs assume deployment of BGPHES during the first quarter of 1993. Based on this assumption, no change is anticipated in the projected level of funding needed to complete the validation effort. VTSRC is adding an additional graduate student, Randy Love, to assist in validation activities. Edward Dorsey, the author of the Document analyzer, is expected to complete the requirements for graduation in December 1992.

# Appendix A

# The Document Analysis Tool (DAT)

# Documentation Analysis Tool (DAT)

# Operations Manual

Table of Contents

I. User's Section ..... 1
I.1. What is DAT? ..... 1
I.2. Formatting of Input Files ..... 6
I.3. Running DAT ..... 6
I.4. DAT Output ..... 8
II. Technical Section ..... 11
II.1 Installation of DAT ..... 11
II.2 Piecewise Execution of DAT ..... 12
II.3.1 Document Data Extractor ..... 12
II.3.2 Document Preprocessor ..... 12
II.3.3 Document Analyzer ..... 13
II.4 Debugging Information Provided by DAT ..... 13
II.4.1 Table of Contents (TOC) Repository ..... 14
II.4.2 Keyword Repository ..... 15
II.4.3 Index Repository ..... 16
II.4.4 Glossary Repository ..... 16
Appendix A
File Descriptions ..... A1
Appendix B
File Formats ..... B1
Appendix C
Constants ..... C1
Appendix D
Troubleshooting Guide ..... D1
Appendix E
Universal Documentation Format Definition ..... E1

# I. User's Section

The following portion of the manual deals with user-oriented topics of the documentation analysis tool (DAT). The topics covered in this section include:

- what is DAT?
- input file format
- program execution

## I.1. What is DAT?

DAT stands for Documentation Analysis Tool, and is used to render a quality assessment of requirements and design specifications for computer software products. A simplified representation of what DAT does is shown in Figure 1.

Basically, the original document is processed by both a document data extractor and a document preprocessor. The data extractor generates keyword data for use by the document analyzer; the preprocessor transforms the original document to a universal document format, for processing by the document parser. The document parser produces intermediate representations of vital document components. Using the intermediate representations, the document analyzer renders a report of the quality of the original document. A more detailed explanation of this process is given in the technical portion of this manual.

DAT performs its assessment based on the presence of documentation *properties*. Each property measured by DAT is evidential of a documentation *attribute*. Together, an attribute-property pair for a *documentation quality indicator* (DQI). DQIs are in turn evidence that certain software engineering *principles* were applied to produce the document analyzed. The Objectives Principles Attributes (OPA) Framework, identifies linkages between software engineering *objectives*, *principles*, and *attributes*, enabling the results of the DAT analysis to be used to determine the extent to which desired objectives have been met by the documentation of a software product. The linkages between objectives, principles, and attributes are displayed in Figure 2.

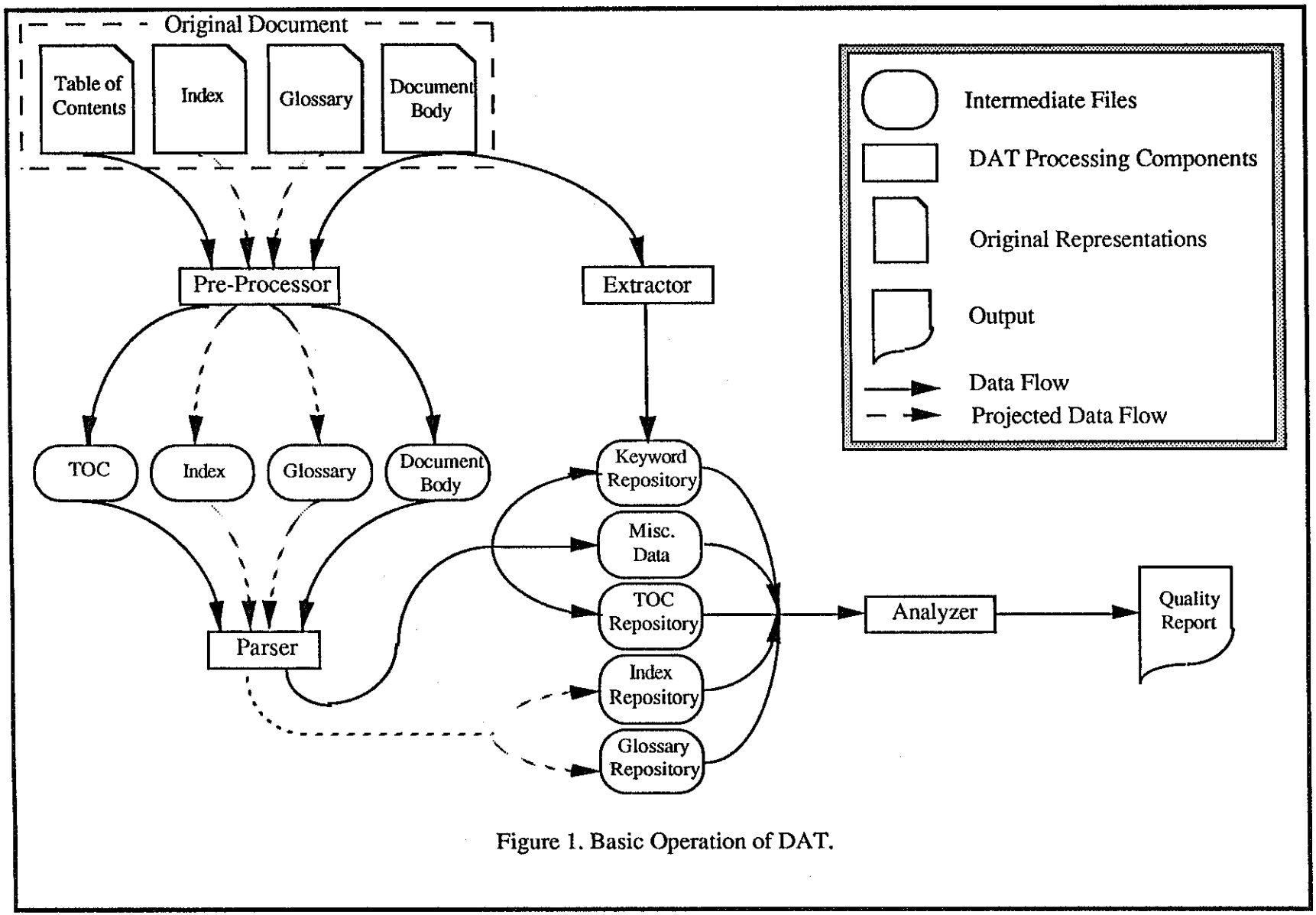



Figure 1. Basic Operation of DAT.

The document properties assessed by DAT are described in Table 1.

Table 1. Documentation Properties Assessed by DAT.

| Property | Definition | Measurement Approach |
|---|---|---|
| Locational Accuracy of Table of Contents (TOC) | The accuracy with which the TOC notes the starting locations of document sections. | For each TOC entry, the section number, section title and page number are stored in the TOC repository. When a section header is encountered within the document body, the page on which it occurs is compared to the page listed in the TOC. The ratio of correct TOC entries to the total number of TOC entries is calculated. |
| Correctness of TOC Nomenclature | The correspondence between section titles within the TOC and their counterparts within the document body | For each TOC entry, the section number, section title and page number are stored in the TOC repository. When a section header is encountered within the document body, the title in the heading is compared to the corresponding title listed in the TOC. The ratio of correct TOC entries to the total number of TOC entries is calculated. |
| Completeness of TOC | The completeness of coverage of document sections by the TOC. | The number of TOC entries is counted and stored. After the entire document body has been analyzed, the ratio of TOC entry to total number of sections within the document body is calculated. |

Table 1. Documentation Properties Assessed by DAT.

| Property | Definition | Measurement Approach |
|---|---|---|
| Acronym Usage | The proper use of acronyms within the document, including definition at initial use and consistency of context in which the acronym is used. | Each acronym is checked to determine if (1) it was defined at its initial usage, and (2) it is used in a consistent manner (see keyword context consistency). The average over all acronyms is calculated. |
| Completeness of Glossary | The presence of document keywords within the glossary as entries. | Using the keyword repository, the glossary is examined to determine the number of keywords present as entries in the glossary. For each keyword in the glossary, the keyword frequency is added to a running total. The final measure is based on the ratio of the glossary score to the total possible score (i.e., the sum of the keyword frequencies). |
| Order of Glossary | The use of an alphabetical ordering system in the glossary. | The UNIX utility *diff* is used to generate a comparison between the actual glossary order and the order of the glossary as generated by the UNIX utility *sort*. The ratio of the number of ordered entries to the total number of entries is calculated. |
| Locational Accuracy of Index | The accuracy with which the index notes the pages on which certain terms are located. | For each entry in the index, the term of interest and the locations in which it occurs are stored in the index repository. The ratio of correct entries to total number of entries is calculated, using he frequency of the term in quest to be used as a weighting scheme. |

Table 1. Documentation Properties Assessed by DAT.

| Property | Definition | Measurement Approach |
|---|---|---|
| Order of Index | The use of an alphabetical ordering system in the index. | This measure is calculated in a manner similar to Order of Glossary, only the index entries are used. See Order of Glossary. |
| TBD/TBS Frequency | The frequency of TBD/TBS statements within the document body relative to a predefined acceptance rate. | The frequency of TBD/TBS ("to be defined..."/"to be specified...") phrases is calculated, The total number of such phrases is then made relative to a predefined tbd/tbs acceptance rate, which accounts for the number of sections in the document. |
| Missing/Incorrect References | The frequency of references to nonexistent sections or sections inappropriate to the reference with respect to context. | Each intersectional reference is checked to determine if (1) the section being referenced is present, and (2) the context of the referenced section ids similar to the current section. If either of these is false, the reference is counted as missing or incorrect. The ratio of missing/incorrect references to the total number of references is calculated. |
| Reference Appropriateness | Of those references deemed to be appropriate, the degree to which the average reference ids appropriate. | If a reference is *not* missing/incorrect (see Missing/Incorrect References), hen the degree of similarity of the referenced section to the referencing section is calculated. The total similarity over all sections is calculated. |

## I.2. Formatting of Input Files

In order to be interpretable by DAT, all input files must conform to a few formatting criteria. Most of this formatting is performed automatically by the document preprocessor, but before any component of DAT is executed on an input file, an explicit manual check must be made to insure that the document conforms to the following format:

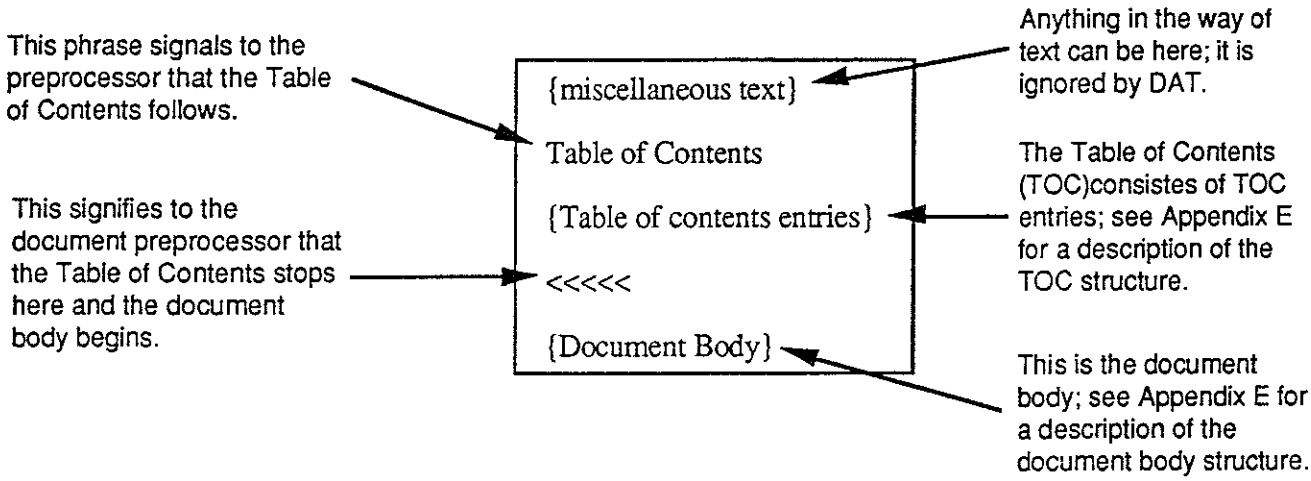



Figure 3. Input File Format.

The most important manual alteration is the insertion of the end of table of contents symbol, "<<<<<". Note that if the table of contents is absent, the phrase "Table of Contents" must also be placed in the file, on the line immediately preceding the symbol "<<<<<". By inserting these phrases/symbols, the document parser can recognize a null table of contents and continue parsing. Otherwise, errors in interpretation occur, resulting in invalid measures at best.

Proper formatting (manual and through application of the preprocessor) prepares the document text file for analysis.

## I.3. Running DAT

Once DAT has been installed and the input files are available, DAT can be executed. Execution of DAT in a UNIX environment proceeds through one of two options: (1) script-based execution, or (2) piecewise execution. Since piecewise execution is useful only in a troubleshooting/debugging capacity, it is described in detail in the technical section of this manual (II.2).

## I.3.1 Script-based Execution (docalyze)

During installation, an executable script named *docalyze* should have been installed; if not, install it now. *Docalyze* contains a series of UNIX commands which call the appropriate components of DAT at the appropriate times, with the correct input. The resulting output is manipulated so that generated assessment reports bear names corresponding to the input file assessed. *Docalyze* also enables multiple file processing with a single command, either by listing the filenames explicitly, or through the use of wildcards (limit of 9 files at a time). In order to use *docalyze*, give a command of the following form:

**docalyze file1 [file2] [file3] [file4] [file5] [file6] [file7] [file8] [file9]**

If all of the input filenames contain a common expression, a wildcard expression may be used. Consider the input files:

input1 input3 input5 input7
input2 input4 input6 input8

The following command analyzes all eight of these files:

**docalyze input***

The resulting output files from this command (assuming error-free execution) would be:

input1.report input3.report input5.report input7.report
input2.report input4.report input6.report input8.report

Note that erroneous output may be generated. If the report files generated by *docalyze* do not conform to the example report (see I.4 Output), then an unrecoverable error occurred during execution, and the output is invalid. Such cases may warrant piecewise execution to determine the source of the error (see II.2 Piecewise Execution).

Due to the ability to process multiple files easily, the script-based execution method is the recommended method for processing large amounts of data. Figure 4 demonstrates the display sent to the screen as a result of analyzing the file "demo".

```
docalyze  demo

Documentation Analysis Tool (DAT)
Developed at the Systems Research Center, Virginia Tech by E.V. Dorsey

NOTE: this utility assumes the definition of the evironmental variable
      DAT_DIRECTORY, specifying the location of the DAT components:
      * extract - document data extractor
      * prep - document preprocessor
      * doc_assess - document analyzer

DAT_DIRECTORY is currently  /u2/sqaevd/ada/parser

All output is placed in the resident directory of this script.

Eliminating unreadable characters from demo...
Converting page breaks within demo...
Performing data extraction on demo...

Keyword Extractor for use with the Documentation Analysis Tool (DAT).

Developed by E.V. Dorsey and J.D. Arthur at the Systems Research Center,
Virginia Tech, Blacksburg, Virginia, 1991.

The extract utility creates/overwrites the following files:
    DAT_keywords - keyword information file
    Other intermediate files which are removed at program termination,
    all of the form DAT_temp[1-6].

NOTE: this utility assumes the definition of the evironmental variable
      DAT_DIRECTORY, specifying the location of the DAT object code.

Currently, the variable DAT_DIRECTORY is  /u2/sqaevd/ada/parser
Extracting data from demo ...
Ensuring line length throughout demo ...
Completed acronym extraction for demo ...
Completed frequency-based extraction for demo ...
Completed stopword-based extraction for demo ...
Data Extraction of demo complete

Preprocessing demo...

Analyzing demo - verbose output will be in file DAT.verbose...
Analysis of demo complete.
Stripping results from input and placing in file demo.report...
Done.
```

Figure 4. Screen Display Generated by *docalyze*.

## I.4. DAT Output

In the report generated by DAT, each of the measures performed by DAT is displayed with

information pertinent to the calculation of the property, where appropriate. Also displayed is the number of syntax errors from which the document parser recovered in the course of interpreting the document. If the number of errors is high, there may be need to question the accuracy of the measures given (see Appendix D: Troubleshooting Guide), as the analyzer may have incorrectly interpreted large quantities of the document due to formatting problems.

Figure 5 is an example of the report generated by *docalyze*, using the file *demo* as input.

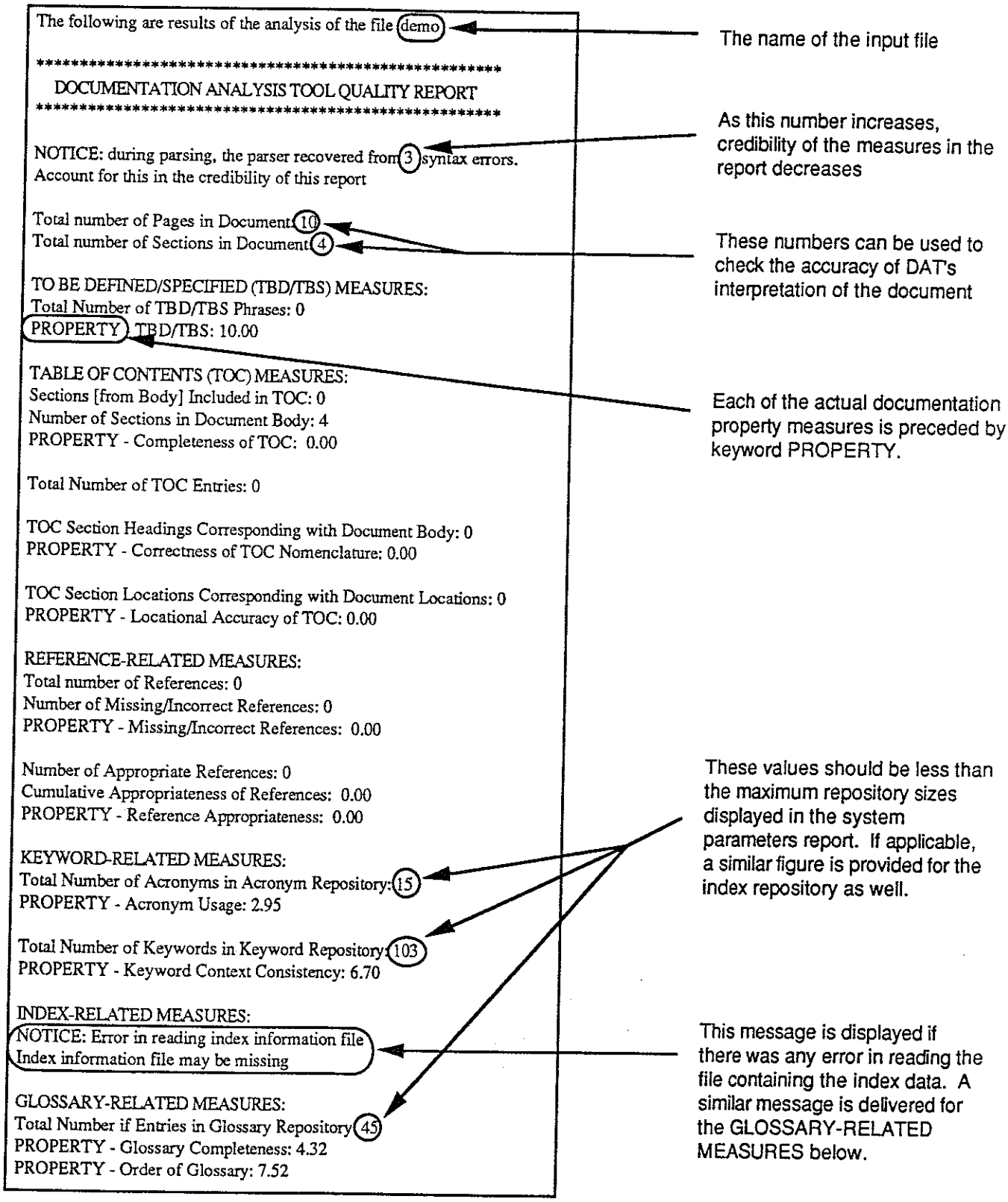



Figure 5. DAT Analysis Report.

# II. Technical Section

The following portion of the manual describes the more technical, non-user issues concerning the documentation analysis tool (DAT). Included as technical issues are:

- software installation
- piecewise execution of DAT, and
- investigation of errors & debugging of software.

Each of these topics are covered in the sections that follow.

## II.1 Installation of DAT

The components of DAT are designed to run within a single UNIX directory. Since DAT is implemented in Ada, the directory chosen may need to be established as an Ada directory by whatever standards are set by the Ada compiler in use. Thus, the simplest way to install DAT is to perform the following procedure:

1. If *aflex* (or *alex*) and *ayacc* are not currently available on the system, place the source code for *aflex* and *ayacc* onto the system and compile it. The executable utilities can be placed anywhere on the system, as long as they are executable from the DAT directory.

2. Place the following classes (see Appendix A: File Descriptions for details on these file classes) of files into the DAT directory:
   - Source Code
   - Compilation Tools
   - Utility Specifications
   - *docalyze*

3. Execute the following compilation tools to compile the components of DAT:
   - **make**
   - **ppmake**
   - **extractor_make**

A listing of the DAT directory should now yield all the files installed in step 2, and the executable versions of the document data extractor , preprocessor and document analyzer.

4. Place the input documents on the system, if necessary.

## II.2 Piecewise Execution of DAT

In cases where run-time errors occur, it may be necessary to analyze a document one step at a time to determine the cause of the error. DAT consists of three components:

(1) document data extractor,
(2) document preprocessor, and
(3) document analyzer.

The order in which (1) and (2) are executed is not important, but BOTH of these components must be executed before the execution of the document analyzer (3).

### II.3.1 Document Data Extractor

To create the data files needed by the document analyzer, the document data extractor (1) must be executed, using the document text file as input. The command to do this is:

**extract input_file**

where *input_file* is the name of the document text file. The extractor will create a data file, *DAT_keywords* , and place it in the DAT directory. This file will later be used by the document analyzer component of DAT.

### II.3.2 Document Preprocessor

The document analyzer is not capable of interpreting a wide range of formats, so the original document text file must first be preprocessed. To preprocess the document text, issue the following command:

**prep < input_file > input_file.pp**

where *input_file.pp* is the preprocessed version of the original document, *input_file*. Any name can be assigned to the preprocessed file, as long as the same file is used as the input to the document analyzer (see II.3.3 Document Analyzer).

### II.3.3 Document Analyzer

Once the data files have been generated and the input file has been preprocessed, the document is ready for analysis. To analyze a document, use the following command:

**doc_assess < input_file.pp > output_file**

As with the document preprocessor, the actual names of the input and output files do not matter, as long as the input file is the result of a successful execution of the document preprocessor. Note that when using the piecewise execution method to analyze a document, the resulting output contains information in it other than the analysis results (i.e., debugging information). The analysis report is always the final text in the output file. If the analysis report is absent, an execution error occurred.

## II.4 Debugging Information Provided by DAT

Since DAT is a prototype, the need for debugging will probably arise as new documents are analyzed. Also, the measures generated by DAT should be checked for some time to assure that unforeseen anomalies do not cause inaccurate results. To facilitate these actions, DAT produces a file, *DAT.verbose*, when executed using the *docalyze* command (otherwise, the same information is contained within the output of *doc_assess*). *DAT.verbose* contains information regarding each sentence it parses and the location of such sentences within the original document (see Figure 6); *DAT.verbose* also contains information on the several information repositories produced during execution, including the:

- Table of Contents (TOC) Repository
- Keyword Repository
- Index Repository, and
- Glossary Repository

The information provided by *DAT.verbose* on each of these repositories is described below.

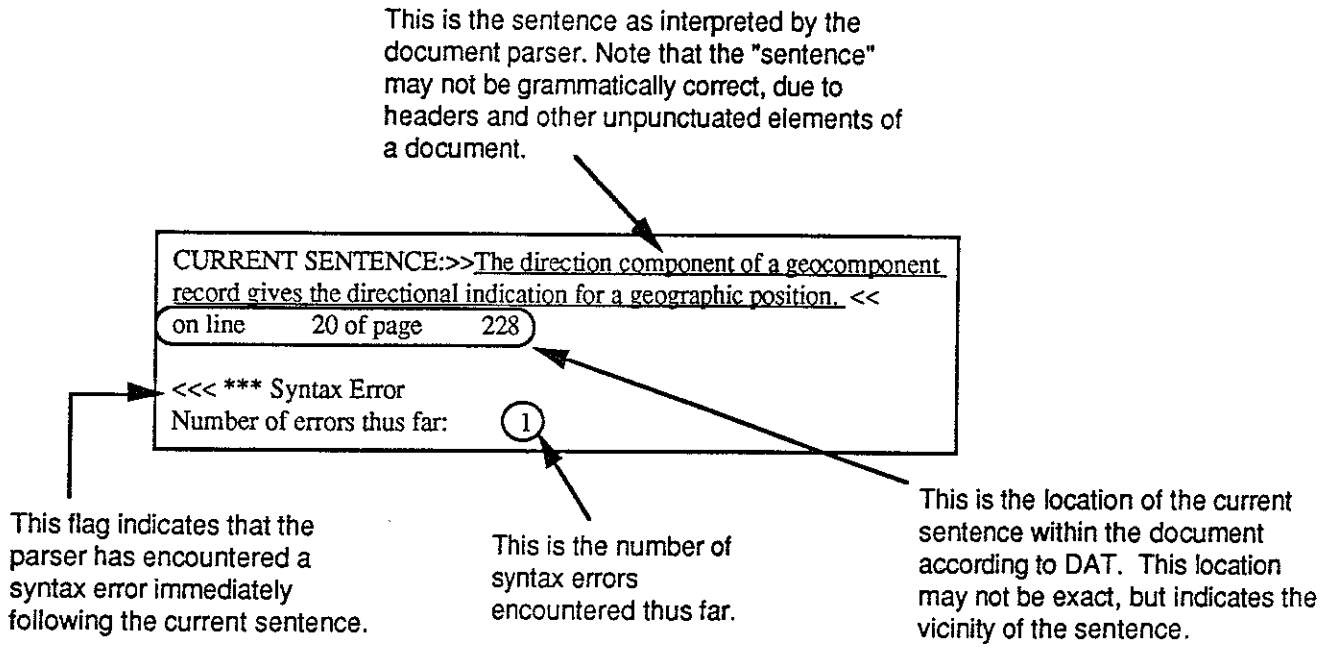



Figure 6. How to Interpret the Data in *DAT.verbose*.

## II.4.1 Table of Contents (TOC) Repository

The TOC repository is an intermediate representation of the Table of Contents of the document. By searching *DAT.verbose* for the string "TOC Repository", the display of DAT's interpretation of the TOC can be seen. The data is displayed as <section number> <section title> <page number>; for example:

II.4.1 Table of Contents Repository 14

would be the entry representation for this section. A page number of zero (0) indicates that no page number was present in the TOC for that section, or that an error in formatting prevented successful determination of the page number by the document preprocessor.

Inspection of the TOC repository can reveal that an error in preprocessing led to a number of TOC entries being misinterpreted or omitted altogether. More on debugging can be found in the table in Appendix D: Troubleshooting Guide.

## II.4.2 Keyword Repository

The keyword repository is probably the single most important information repository, as several measures rely upon its data both directly and indirectly. The display of the keyword repository in *DAT.verbose* is in the following format:

<entry #> <keyword> <# of occurrences> <rationale> <sample> <locations> <contexts>

<entry #> - the position of the keyword record in the keyword repository; this number is simply an index into the array holding the keyword information records.

<keyword> - the lowercase character string representation of the keyword.

<# of occurrences> - the number of times the document data extractor recognized the keyword's presence in the original document. This measure may be slightly inaccurate at times, but the measures using it are relative, so little (if any) error is introduced as a result.

<rationale> - the reason why the term is considered a keyword. The possible values are:

- FREQ - the term has a frequency within the prescribed thresholds.
- ACRO - the term is an acronym.
- TOC - the term was present in the Table of Contents.
- ERR - some error was encountered in reading the rationale field from the file *DAT_keywords*.

<sample> - this field has the value "SAMPLE" if the keyword is part of the sample used in calculating measures, "-----" if not part of the sample.

<locations> - this is a list of the pages on which a keyword occurs, and at the present time probably suffers from inaccuracy, due to the inability to determine position within the document precisely.

<contexts> - the contexts in which the keyword is used. This field is displayed ONLY if the keyword is part of the sample.

### II.4.3 Index Repository

For each entry in the index, the word and the pages on which it is purported to occur are displayed.

### II.4.4 Glossary Repository

For each entry in the glossary, the word defined in the glossary is displayed.

# Appendix A: File Descriptions

DAT uses several prefixes/suffixes to distinguish the different files it uses/generates. A list of these naming conventions follows, with descriptions of the files named in such a manner:

| **DAT Output** | |
|---|---|
| DAT.verbose | This file is generated each time a document is analyzed and contains some useful debugging information, as well as the analysis results (approximately the last 50 lines of the file). This file is ***overwritten*** each time DAT is executed, so it must be copied to another file if the results are to be saved. |
| <fn>.report | This file contains the results of the analysis of the file <fn>. It is generated when a document is analyzed using the command *docalyze*. |
| DAT_acronyms<br>DAT_keywords | These files contain data on the keywords and acronyms within a document. These data are used by DAT to perform certain analyses upon the current input document. DAT_acronyms and DAT_keywords are generated by the preprocessor and are ***overwritten*** each time the preprocessor is executed. |
| DAT_glossary<br>DAT_glossary_pp<br>DAT_index | These files contain data on the glossary and index within a document. This data is used by DAT to perform certain analyses upon the current input document. At the current time, these files must be manually generated. |

| Source Code | |
|---|---|
| doc_lex.a<br>doc_lex_dfa.a<br>doc_lex_io.a | These files are the source code for the lexical analyzer portion of the documentation analyzer. They are generated by the lexical analyzer generator *aflex* from the specification file *doc_lex.l* . |
| doc.a<br>doc_shift_reduce.a<br>doc_goto.a<br>doc_tokens.a | These files are the source code for the parser portion of the documentation analyzer. They are generated by the parser generator *ayacc* from the specification file *doc.y*. |
| pp_lex.a<br>pp_lex_dfa.a<br>pp_lex_io.a | These files are the source code for the lexical analyzer portion of the document preprocessor. They are generated by the lexical analyzer generator *aflex* from the specification file *pp_lex.l*. |
| acronym.a<br>frequencyfilter.a<br>stopfilter.a<br>tocfilter | These files are all required by the document data extractor (see *extract* under *Executables*) for execution. They can be compiled by using the command *extractor_make* (see *extrator_make* under *Compilation Tools*). |
| dat.a<br>glossary.a<br>index.a<br>keywords.a<br>phrases.a<br>references.a<br>requirements.a<br>sections.a<br>sections.a<br>toc.a<br>words.a | These are the source code files for the main body of the documentation analyzer. They can be compiled using the command *make*, as the file *makefile* contains the proper specifications for the compilation of these files. The resulting executable is the file *doc_assess* (see *doc_assess* under *Executables*). |

| | |
|---|---|
| **Utilities** | |
| aflex (alex) | This is the lexical analyzer generator developed at UC Irvine. |
| ayacc | This is the parser generator developed at UC Irvine. |
| **Compilation Tools** | |
| ppmake<br>extractor_make | These files are executable scripts that issue the commands to compile the source code for the document preprocessor (see *prep* under *Executables*) and the document data extractor (see *extract* under *Executables*), respectively. |
| makefile | This is the specification file utilized by the UNIX *make* utility to construct the main body of DAT, *doc_assess*. Evokes *aflex* and *ayacc* if necessary to create the lexical analyzer and parser, as well as compile and link all involved files. |
| **Executables** | |
| doc_assess | This is the executable version of the main body of DAT. It can be called separately or evoked as a result of the *docalyze* command. |
| extract | This is the executable version of the document data extractor. It can be called separately or evoked as a result of the *docalyze* command. |
| prep | This is the executable version of the document preprocessor. It can be called separately or evoked as a result of the *docalyze* command. |
| docalyze | This is an executable script which evokes the various components of DAT, creating *DAT.verbose* and *<fn>.report*, where <fn> represents the name of the input file supplied. This is the recommended way to analyze documentation using DAT. |
| **Utility Specifications** | |
| doc_lex.l | This is the *aflex* specification file for the lexical analyzer used by *doc_assess*, the main body of DAT. |
| pp_lex.l | This is the *aflex* specification file for the lexical analyzer used by *prep*, the document preprocessor.. |
| doc.y | This is the *ayacc* specification file for the document parser used by *doc_assess*, the main body of DAT. |

# Appendix B: File Formats

In making adaptations to the existing components of DAT, the various file formats must be maintained, unless the I/O routines are adjusted accordingly. To facilitate maintenance of file formats, the following information is provided on the formatting constraints placed on the files involved in using DAT.

## Keyword Data

Filename: DAT_keywords

Line Format:
{SPACEs}{INTEGER}{WORD_LENGTH CHARACTERs}{rationale}{NEWLINE}

Notes

- The SPACEs at the beginning of each line are *optional.*
- An INTEGER is a fixed point number.
- WORD_LENGTH is a constant used by DAT (see Constants); for example, if WORD_LENGTH is 40, then 40 characters are in this field, padded by SPACEs, if necessary.
- A CHARACTER is an element of the ASCII character set.
- The value of rationale is either "frequency" or "acronym", that defines the basis for the term's being a keyword.

## Acronym Data

Filename: DAT_acronyms

Line Format:
{WORD_LENGTH CHARACTERs}acronym{NEWLINE}

Notes

- WORD_LENGTH is a constant used by DAT (see Constants); for example, if WORD_LENGTH is 40, then 40 characters are in this field, padded by SPACEs, if necessary.
- The string "acronym" represents the same information as the rationale field in the Keyword

Data file.

## Index Data

Filename: DAT_index

Line Format:
{WORD_LENGTH CHARACTERs}{INTEGER}SPACE{INTEGER}...{NEWLINE}

Notes

- WORD_LENGTH is a constant used by DAT (see Constants); for example, if WORD_LENGTH is 40, then 40 characters are in this field, padded by SPACEs, if necessary.
- The INTEGER field represents the page(s) on which the word occurs. Note that there may be several entries here, as long as each is delimited by at least one SPACE.

## Glossary Data

Filename: DAT_glossary

Line Format:
{WORD_LENGTH CHARACTERs}{NEWLINE}

Notes

- WORD_LENGTH is a constant used by DAT (see Constants); for example, if WORD_LENGTH is 40, then 40 characters are in this field, padded by SPACEs, if necessary.

## Glossary Order Data

Filename: DAT_glossary_pp

Line Format: This file is generated by re-directing the I/O of a UNIX *diff* operation. The sequence of commands to generate this file follow:

```
sort DAT_glossary > sort.temp
diff DAT_glossary sort.temp > diff.temp
grep -h -v "[<>a]" diff.temp > DAT_glossary_pp
rm sort.temp diff.temp
```

# Appendix C: Constants

The following table displays the constants used to regulate the operation of DAT. The package in which each constant is declared is supplied; to determine which file a package is in, add the '.a' filename extension to the package name (e.g., package sections is in the file section.a).

Adjustments to the values of these constants can be made to effect changes in DAT operation with respect to information repository capacities, data file names, and predefined threshold values for various assessments.

| Constant Name | Package of Declaration (add .a for filename) | Purpose/Definition |
|---|---|---|
| word_length | words | The maximum number of characters a variable of type *word* can contain. NOTE: a change to this variable must be made in files used by the document data extractor - see in-line documentation for details. |
| max_section_depth | sections | The maximum depth to which sections can be nested, i.e., 1.2.3.4 has a nesting depth of 4. |
| null_id_value | sections | Section ID numbers are initialized to this value. |
| max_toc_entries | toc | The maximum number of table of contents entries that can be represented by the TOC information repository. |
| max_toc_loc_err | toc | The number of consecutive location errors encountered in the TOC information repository before DAT undergoes locational self-correction. |
| keyword_context_ similarity_threshold | keywords | If the similarity value between two contexts of a keyword's usage are above this value, then the contexts are considered similar enough to be combined into a single context for purposes of keyword context consistency assessment. |

| Constant Name | Package of Declaration (add .a for filename) | Purpose/Definition |
|---|---|---|
| toc_freq_value | keywords | This value is used as the occurrence frequency for terms that are considered keywords by virtue of being in the Table of Contents, as the actual occurrence frequency is indeterminable. |
| max_keywords | keywords | The maximum number of keyword records that can be represented by the keyword information repository. |
| max_acronyms | keywords | The maximum number of acronym records that can be represented by the acronym information repository. |
| keyword_filename | keywords | The name of the file containing the keyword data, generated by the document data extractor. |
| mark_increment | keywords | Determines the sample size relative to keyword based measures. The higher this integer is in value, the smaller the resulting sample. |
| max_index_entries | index | The maximum number of index entry records that can be represented by the index information repository. |
| index_filename | index | The name of the file containing the index data to be used by the document analyzer. |
| max_glossary_entries | glossary | The maximum number of glossary entries that can be represented by the glossary information repository. |
| glossary_filename | glossary | The name of the file containing the glossary data to be used by the document analyzer. |
| glossary_pp_filename | glossary | The name of the file containing the glossary ordering data to be used by the document analyzer. |

| Constant Name | Package of Declaration (add .a for filename) | Purpose/Definition |
|---|---|---|
| reference_similarity_ threshold | references | The minimum similarity value that must exist between a reference context and the content of the referenced section for the reference to be considered "appropriate". |

# Appendix D: Troubleshooting Guide

| Symptom | Action |
|---|---|
| High number of syntax errors raised by document parser. | 1. Search *DAT.verbose* for instances of the string "Syntax Error"; note the locations in the document in which the errors occur (see Figure 6).<br>2. Inspect the file *DAT.prep* for the cause of the errors. |
| Fatal error in the document analysis phase (i.e., when *doc_assess* is executing). | 1. Inspect *DAT.verbose* to determine at what point within the document the analyzer crashed.<br>2. Inspect *DAT.prep* in the area of the fatal error (see step 1) for possible faulty preprocessing. A possible error is the failure to start/end a grammar nonterminal with the appropriate delimiter. |
| Fatal error in preprocessor. | 1. Inspect *DAT.prep* to determine the location of the error.<br>2. Inspect the original input file for the existence of unreadable characters.<br>3. If unreadable characters are found, a UNIX *tr* command can be used to filter out the offending character within the file *docalyze*. |
| High number of location self-correction by DAT. | 1. Be wary of an unusually low value for the Locational Accuracy of TOC measure.<br>2. Be wary of an unusually low value for the Locational Accuracy of Index measure. |

| Symptom | Action |
|---|---|
| Zero value for Keyword Context Consistency or Acronym Usage measures. | 1. Inspect the value of the number of keywords (acronyms) in the keyword (acronym) repository given in the DAT report file. If it is zero, then there may be a problem in the document data extractor, see step 2.<br>2. Verify the existence of the file *DAT_keywords*.<br>3. Inspect *DAT.verbose* for the keyword repository display, and inspect its contents. |
| Low score for any Table of Contents (TOC) measure. | 1. Inspect the value of the number of TOC entries given in the DAT report file for correctness.<br>2. Inspect the contents of the TOC repository in *DAT.verbose* for anomalies. |
| Any fatal error in DAT execution. | 1. Inspect the input file to determine its conformance to the prescribed format (see Figure 3). |

# Appendix E: Universal Documentation Format Definition

This appendix contains a BNF definition for the documentation format expected by the documentation parser. The function of the document preprocessor is to transform a document into this universal documentation format. The following definition should be used in constructing new preprocessors for differently formatted documents. In interpreting the following definition:

| | |
|---|---|
| **bold typeface** | denotes case-insensitive literals |
| ALL CAPITALS | denotes tokens returned from the lexical analyzer |
| lowercase | denotes nonterminals defined within this grammar |

document ⇒ table-of-contents
⇒ document document-body

document-body ⇒ section
⇒ document-body section

table-of-contents ⇒ TOC-START
⇒ toc-entry
⇒ table-of-contents toc-entry
⇒ table-of-contents phrase
⇒ table-of-contents TOC-STOP

toc-entry ⇒ TOC-ENTRY-START ID-NUMBER phrase DOT-STRING INTEGER TOC-ENTRY-STOP
⇒ TOC-ENTRY-START ID-NUMBER phrase INTEGER TOC-ENTRY-STOP
⇒ TOC-ENTRY-START ID-NUMBER phrase TOC-ENTRY-STOP
⇒ TOC-ENTRY-START ITEM-NUMBER phrase DOT-STRING INTEGER TOC-ENTRY-STC
⇒ TOC-ENTRY-START ITEM-NUMBER phrase INTEGER TOC-ENTRY-STOP
⇒ TOC-ENTRY-START ITEM-NUMBER phrase TOC-ENTRY-STOP
⇒ TOC-ENTRY-START TOC-ENTRY-STOP

section ⇒ SECTION-START section-header
⇒ section paragraph
⇒ section list
⇒ section phrase
⇒ section SECTION-STOP

section-header ⇒ SECTION-HEADER-START ID-NUMBER phrase SECTION-HEADER-STOP

paragraph ⇒ PARAGRAPH-START
⇒ paragraph sentence
⇒ paragraph PARAGRAPH-STOP

sentence ⇒ SENTENCE_START phrase SENTENCE-STOP

phrase ⇒ word | phrase word
⇒ section-reference | phrase section-reference
⇒ punctuation | phrase punctuation
⇒ acronym-initial-usage | phrase acronym-initial-usage
⇒ INTEGER | phrase INTEGER
⇒ phrase ID-NUMBER

list ⇒ LIST-START
⇒ list list-item
⇒ list paragraph
⇒ list phrase
⇒ list LIST-STOP

list-item ⇒ LIST-ITEM-START
⇒ list-item ID-NUMBER
⇒ list-item ITEM-NUMBER
⇒ list-item paragraph
⇒ list-item phrase
⇒ list-item LIST-ITEM-STOP

section-reference ⇒ **section** ID-NUMBER
⇒ ( **section** ID-NUMBER )
⇒ ( ID-NUMBER )

acronym-initial-usage ⇒ ( ACRONYM )

punctuation ⇒ PUNCT-TOKEN
⇒ (
⇒ )

word ⇒ **section**
⇒ **and**
⇒ **shall**
⇒ HYPHENATED-WORD-TOKEN
⇒ TBD-TBS
⇒ ACRONYM
⇒ WORD-TOKEN

# Appendix B

# Process Indicators

**Process Property:** Traceability with Software Development Files (or Folders) (SDFs)

**Impact of Property:** The absence of a Software Development Files (or Folders) prohibits the documentation of the history of the component. The SDF provides ready access or reference to the software elements representing the status of the component and how that status has been realized. Without the SDF, the trail of documentation to explain the current status could be very difficult to locate and to understand.

**OPA Entity Affected:** Traceability (+)

**Rationale:** A SDF should be created as the result of the initial designation of the component, should describe all significant events pertaining to the component, should detail any and all changes made throughout the development process, and should reflect the current status of all parts of the component. Ancestral documents not a part of the SDF should be listed and referenced (with loctional guidance).

**Measurement Approach:** The approach should reflect the role of the SDF in documenting the progression of the software component with recognition of any deficencies that inhibit traceability:

(1) not existing,
(2) created after the designation of the component thus lacking data prior to that time,
(3) no log or failure to log all significant events, particularly events related to configuration control,
(4) missing specifications (requirements, preliminary design, detailed design, program design, code), or lacking references to their location,
(5) missing results of inspections, walkthroughs, unit and integration tests, or lacking references to their location,
(6) ontaining incorrect parts, specifications, results, data, references, etc.

**Metric:**

**Indicator:**

**Process Property:** Loss of Cohesion from Requirements Volatility

**Impact of Property:** The allocation of requirements to a software component after Software Specification Review(s) [DOD-STD-2167A, 1988] (SSR) has the potential of reducing the cohesiveness of that component by having to associate functions that otherwise would not have been grouped together if recognized earlier in the development process. Deletion of requirements tends to have less negative effects, but can, through redesign, adversely affect the cohesiveness of a component.

**OPA Entity Affected:** Cohesion (-)

**Rationale:** Requirements decomposed following functional or hierarchical perspectives are adversely affected by changes. Either the addition or deletion of requirements can bring abou;a loss in functional continuity or an imbalance in levels of an hierarchy. Previously determined design boundaries might not accommodate the new requirements. Depending on when the addition occurs, testing can be more difficult. The deletion of requirements reduces the functionality of the components affected. This contraction of functionality can reduce the component size sufficiently to make alternative assignments appear more attractive than was originally the case or encourage elimination of the component entirely.

**Measurement Approach:** The effect on cohesion is primarily decisional, i.e. the extent of the loss depends on the degree to which the component is impacted. Adding or deleting many requirements causes major loss of cohesiveness. Modifying requirements is similarly disruptive, but can be motivated by the intent to simplify the design.

**Metric:**

**Indicator:**

**Process Property:** Potential Disruption in Traceability Caused by Requirements Volatility

**Impact of Property:** The allocation of requirements to a software component after SDR† can force changes in the relationships among components that must be described in the design specifications produced in prior stages.

**OPA Entity Affected:** Traceability (-)

**Rationale:** Changes in requirements necessitate the backtracking to prior stages in the development process. A natural tendency is to forego the revision of prior specifications, which disrupts the desired path linking the implementation (code or documentation) to its conceptual roots.

**Measurement Approach:** The effect on traceability is considered to emanate from the elapsed time since the SDR. The greater this period, the higher the potential for traceability to be disrupted. A temporal weighting reflects this view which can be tempered by the significance of requirements changes.

**Metric:**

**Indicator:**

---

† Definitions of acronyms taken from [DOD-STD-2167A 1988, pp. 12-13]
SDR = System Design Review
SSR = Software Specification Review
PDR = Preliminary Design Review
CDR = Critical Design Review

**Process Property:** Reverification to Meet Requirements Volatility

**Impact of Property:** The allocation of requirements to a software component after System Design Review (SDR)† forces a reverification from the preliminary (high-level) design forward to the current point in the development process.

**OPA Entity Affected:** Early Error Detection (-)

**Rationale:** Performing reverification, especially at a level representative of a much earlier stage in the development process, has the tendency to lack the rigor that typifies the temporally consistent verification. Further, the detection of errors at this point has lost the advantage realized by correction before progressing further with development based on incorrect decisions.

**Measurement Approach:** Consider the effect of changes in requirements from both a magnitude and timing perspective. The magnitude can be expressed as the ratio of unchanged requirements to total, both allocated and derived. The timing effect is reflected as increasing with the length of time following SRS. Note that the deletion of requirements, while ostensibly reducing the test time, can require redefinition of test threads and changes in test procedures. Consequently, additions and deletions are treated identically.

**Metric:**

**Indicators:**

---

† Definitions of acronyms taken from [DOD-STD-2167A 1988, pp. 12-13]
SDR = System Design Review
SSR = Software Specification Review
PDR = Preliminary Design Review
CDR = Critical Design Review

**Process Property:** Requirements Defects Detection

**Impact of Property:** The discovery of defects in software requirements following Software Specification Review (SSR) is indicative of a weakness in the review process that permits errors to propagate beyond requirements specification into the design activities. Correction after design is underway mandates respecification and redesign with an attendant potential loss of quality.

**OPA Entity Affected:** Early Error Detection (-)

**Rationale:** Interim requirements reviews, culminating with the SSR, are intended to discover defects in the specification of software requirements. Defects discovered following SSR, that do not emanate from changes in requirements, reflect ineffectiveness in the requirements review. The later the discovery of the defect, the greater the potential for affecting the amount and extent of redesign, recode, and retest that must be done.

**Measurement Approach:** The number of requirements defects discovered versus the total number of allocated and derived requirements must be considered. However, the duration between the SSR and the defect discovery must be a factor since the later the discovery, the greater the potential for having a negative impact on quality. The measurement must be applied to individual software components.

**Metric:**

**Indicator:**

**Process Property:** Requirements Volatility Generating Added Coupling

**Impact of Property:** The allocation of requirements to a software component after SSR has the potential of introducing added coupling with existing components.

**OPA Entity Affected:** Coupling (-)

**Rationale:** Requirements decomposed following functional or hierarchical perspectives are adversely affected by changes. The addition or deletion of requirements can introduce necessary data sharing and communications that couple components beyond the degree redefinable if the requirements were specified originally. Existing design boundaries might not accommodate the new requirements, forcing linkages among components. Such linkages can complicate unit and integration testing. Less likely is the addition of coupling through the deletion of requirements; however, dependencies could be created in this way.

**Measurement Approach:** Requirements changes can cause broad adverse effects. The decisional influence exerted on coupling is reflected by the level of the requirements change since the "ripple effect" tends to propagate downward. (Picturing the decomposition tree helps to understand this explanation of the approach).

**Metric:**

**Indicator:**

**Process Property:** Development Instability

**Impact of Property:** Instability in development can stem from changes in:

(1) personnel responsible for component,
(2) target hardware platform,
(3) target language or language translator,
(4) software development environment, and
(5) project and organizational management or policy.

**OPA Entity Affected:** Early Error Detection (-)

**Rationale:** Any form of instability can lead to discontinuities in the development process that: (1) cause an inordinate number of errors, making it difficult to detect and remove all of them, or (2) undermine and weaken the error detection capabilities resident in the process.

**Measurement Approach:** The relationships are potentially so complex that validating a functional form is out of the question. Thus, a crude, simple approach is used. Each form of instability is allocated two points, with a subtraction of one for each change affecting the component.

**Metric:**

**Indicator:**

**Process Property:** Test Model Conformance

**Impact of Property:** Adherence to the stipulated test model lends credibility to the claim that testing has been performed to the level projected in the Software Development Plan or the Software Engineering Manual. Deviations from the model undermine the perception that the programming activities are being properly verified as specifications of components are transformed into code.

**OPA Entity Affected:** Visibility of Behavior (+)

**Rationale:** While the test model includes all phases of testing (Unit, Integration, System Acceptance and Operational Evaluation), the initiation of System Acceptance Test marks the transition from development to deployment *for quality assessment.* In other words, the predictive nature of design indicators at this point gives way to the strict assessment for an acquisition decision that would employ management indicators. With focus on Unit and Integration Test phases, the assessment should reflect: (1) extent and precision of the test model, (2) conformance with the model in test specifications and test procedures, and (3) adherence in the execution of all phases of test. Obviously, (1) is global to all software components, but indicative of the degree to which quality can be assessed through the testing activities. Both (2) and (3) need to be viewed in their relation to (1), for an attempt to correct deficiencies *implicitly* in (1) introduces confusion in the testing process. Note that testing affects products rather late in the development process; therefore, indication of process quality is limited. Further discussion of the rationale is given in CM 92-001: Testing in the Assessment of Software Quality.

**Measurement Approach:** Measure each of the three facets above, reflecting "goodness" through the answers to a series of questions addressed to each facet. A score is accumulated for each facet and a normalization places the total indicator value on the [0,10] scale.

**Metric:**

**Indicator:**

**Process Property:** Software Quality Assurance Infrastructure

**Impact of Property:** Concern for software quality should pervade the entire development process from requirements specification to product delivery. SQA is the primary means of effecting process improvement.

**OPA Entity Affected:** Visibility of Behavior (+)

**Rationale:** Software quality assurance must be recognized as an organizational responsibility and from that perspective a project responsibility. High quality can be achieved only with recognition that quality must be designed "into" the product, with designated responsibility for assuring that this happens, and with defined procedures to carry out the necessary functions [Dhillon 1987, pp. 167-179].

**Measurement Approach:** Beginning simply with recognition, increment to reflect the degree to which SQA is imbedded within the development process as defined by the organization and as used in the project.

**Metric:**

**Indicator:**

**Process Property:** Use of Software Development Files (Folders) (SDFs)

**Impact of Property:** The absence of a Software Development File (or Folder) inhibits a clear picture of the history of the component. Lack of meticulous attention to the use of the SDF can defeat its purpose as readily as not having one. An inaccurate or incomplete SDF negates the ability of a product component to demonstrate proper maturation, and undermines the goal of process improvement through feedback from product assessment.

**OPA Entity Affected:** Visibility of Behavior (+)

**Rationale:** A SDF should be created as the result of the initial designation of the component, should describe all significant events pertaining to the component, should detail any and all changes made throughout the development process, and should reflect the current status of all parts of the component.

**Measurement Approach:** The approach should reflect all practical inadequacies of a SDF:

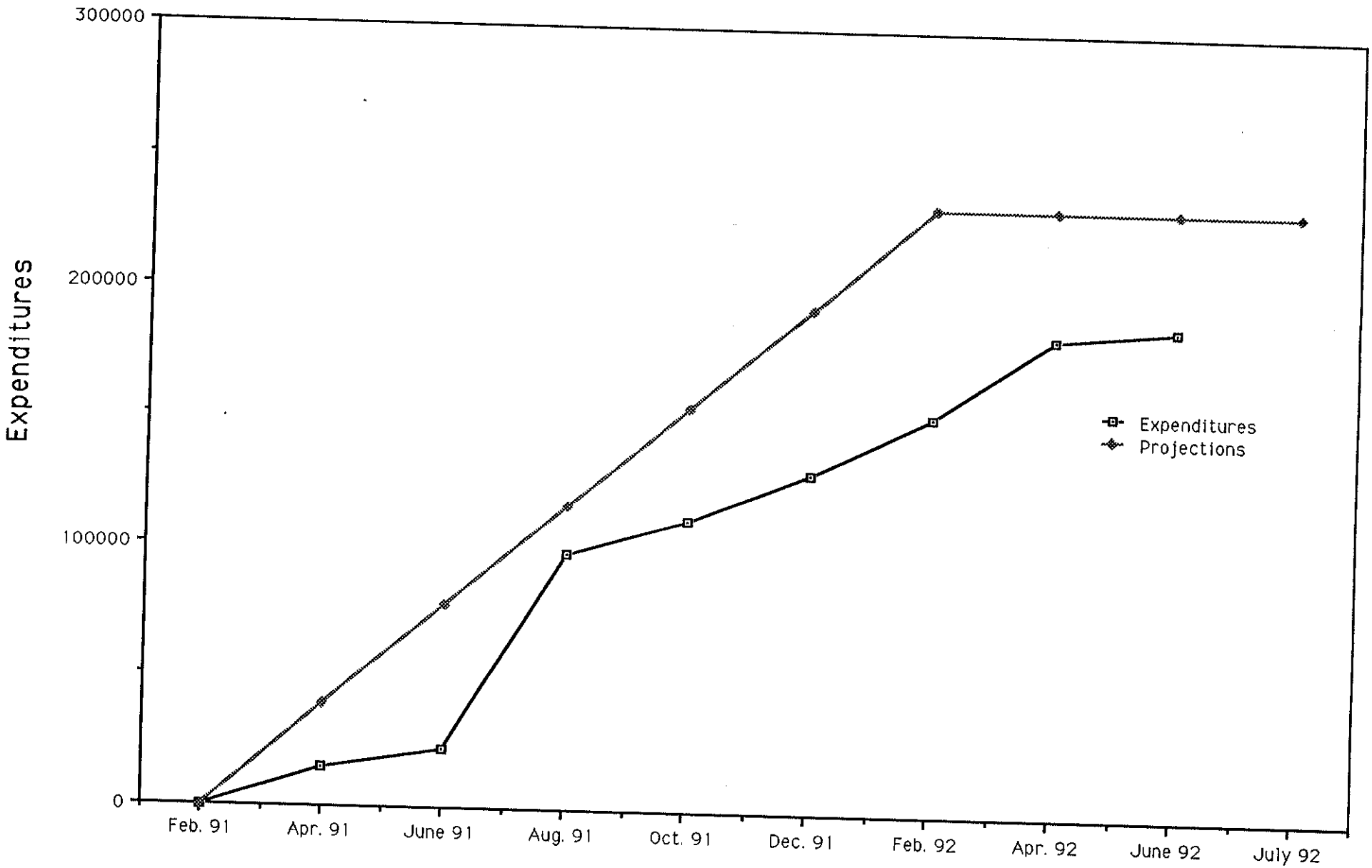
(1) not existing,
(2) created after the designation of the component thus potentially omitting information,
(3) no log or failure to log all significant events,
(4) missing parts (sub-components),
(5) missing specifications (requirements, preliminary detailed design, program design, code),
(6) missing test specifications, procedures, or results,
(7) missing results of inspections, walkthroughs, unit and integration tests,
(8) missing project management data such as schedules, milestone charts, certifications, etc.
(9) containing incorrect parts, specifications, results, data, references, etc.

**Metric:**

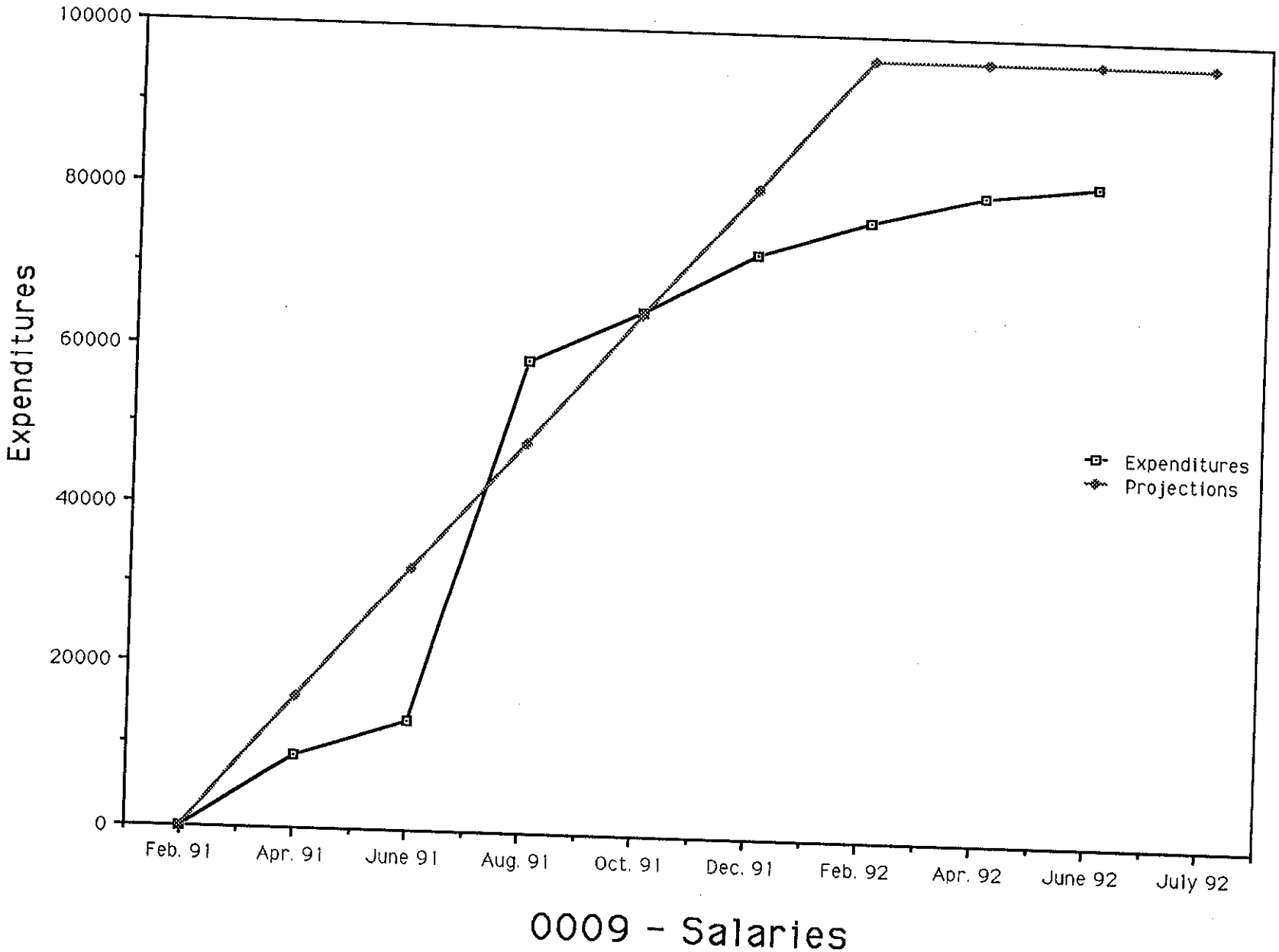
**Indicator:**

# Appendix C

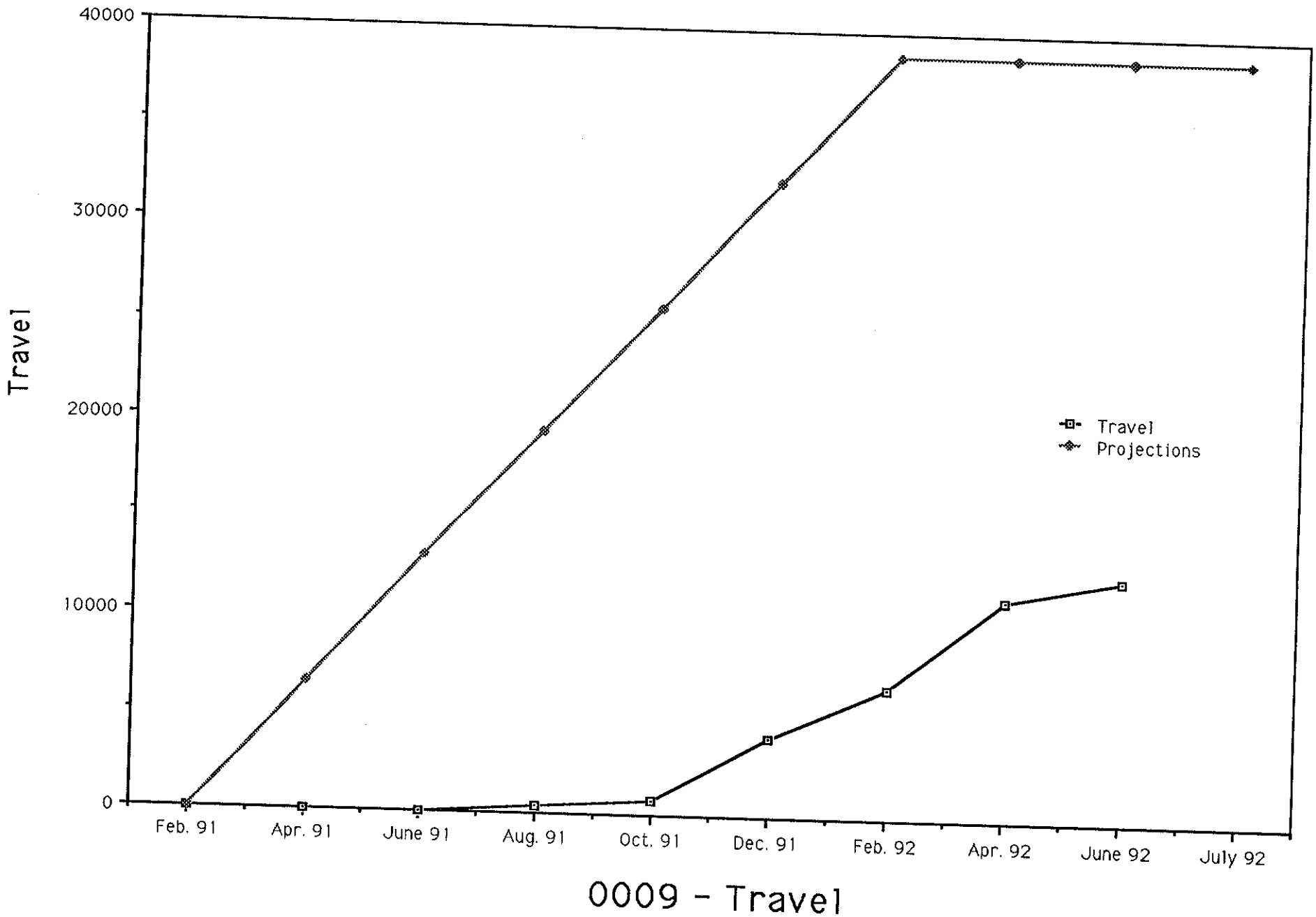
# Expenditure Profiles




0009 - The Evaluation of Software Quality (Total)




0009 - Salaries


40000
30000
20000
10000
0
Travel
Feb. 91
Apr. 91
June 91
Aug. 91
Oct. 91
Dec. 91
Feb. 92
Apr. 92
June 92
July 92
Travel
Projections
0009 - Travel

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Unlimited |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>Systems Research Center SRC-92-006 | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>Naval Surface Warfare Center E-Systems, Melpar Div. |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Systems Research Center<br>6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION<br>Naval Surface Warfare Center E-Systems, Melpar Division |
| 6c. ADDRESS (City, State, and ZIP Code)<br>Virginia Tech<br>Blacksburg, Virginia 24061-0251 | 7b. ADDRESS (City, State, and ZIP Code)<br>Dahlgren, VA 22448-5000 University Center<br>Ashburn, VA 22011 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Logistics Commanders/Computing Resource Managers<br>8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c. ADDRESS (City, State, and ZIP Code)<br>Space and Naval Warfare Systems Command<br>Washington, DC 20363-5100 | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO. |

11. TITLE (Include Security Classification)
The Evaluation of Software Quality: An Empirical Approach to Validation - An Interim Report for Period 1 February - 15 June 1992

12. PERSONAL AUTHOR(S)
James D. Arthur and Richard E. Nance

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Interim | FROM 2/1/92 TO 6/15/92 | July 1992 | 54 |

16. SUPPLEMENTARY NOTATION

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Software quality assessment, documentation analyzer, document quality indicators, process indicators, software tools, software development methodologies, validation |
| | | | |
| | | | |

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

This interim report outlines activities related to the Software Quality Assessment Project funded by the JLC/CRM. Reported activities reflect investigative efforts performed at the Systems Research Center (Virginia Tech) and at the Melpar Division of E-Systems. Accordingly, this report discusses (1) the development and installation of a document analyzer, (2) the development and refinement of document quality and process indicators, and (3) our approach to data collection. An outline of planned activities for the next reporting period is also presented.

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION | |
|---|---|---|
| ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS | | |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE (Include Area Code) | 22c. OFFICE SYMBOL |