DOCUMENTATION: A GROWING NEED... A NEW TOOL

Marcia A. Metcalfe

ABSTRACT

The Software Design and Documentation Language (SDDL) has proven to be an effective, automated documentation tool. This paper presents the purpose, timing, and components of documentation. SDDL is introduced and related to the different timing scenarios; its capability to provide reasonable documentation is discussed and demonstrated by the use of several examples.

I. INTRODUCTION

Documentation has typically been viewed as the drudgery of software development. Weinberg refers to documentation as the castor oil of programming [1]; Kleine says that if design is Cinderella, then certainly documentation is her ugly sister [2]. And it logically follows that documentation became seen in this light since, usually, by the time the documentation stage was reached, the main characters were tired of the program and ready to move on to another project.

However, as more and more programs come into existence, and the need for meaningful documentation also increases, it will be necessary to correct this image if reasonable documentation is to be produced. Without it, the task of understanding, updating, and/or modifying a program becomes increasingly difficult (if not impossible) and costly.

This paper is organized in the following manner: the purpose, timing, and various components of documentation are discussed. Next, the automated tool, SDDL, is presented and related to the different timing scenarios; its capability to provide meaningful documentation is discussed and demonstrated through the use of several examples. Finally, conclusions are drawn.

A. PURPOSE OF DOCUMENTATION

The primary purpose of documentation is to provide undistorted communication between the parties interested in the software, both present and future [3]. Documentation should convey such information as:

- What the program does; how it functions
- How the data/information is represented
- How the various segments/routines relate to one another

This information becomes increasingly valuable as modification is required, and the original developer(s) is no longer available or has forgotten program details [4].

B. TIMING OF DOCUMENTATION

Three documentation timing scenarios will be addressed. The first, "before the fact" documentation, usually presents the program specifications and/or design. The second scenario, "concurrent" documentation, occurs throughout the various phases of software development and provides a working vehicle to prevent distortion of ideas, promotes project control, captures design changes, and permits the orderly development of software. It is useful to the development programmer as well as to the maintenance programmer. The third scenario, "after the fact" documentation typically records the history of development, demonstrates that the program works, and provides a means for maintenance [5].

C. COMPONENTS OF DOCUMENTATION

Given that different users need different information, and that no one document could probably ever provide all the information which in the future may be required, the following documentation components have been identified to accommodate the changing requirements of documentation over the program's lifecycle.

From the very early stages of software design forward, several components become essential. One such component is a high-level description (overview) of the program in prose. It should include such information as what function the program is being designed to perform and what its limitations
and assumptions are. As the design progresses, the data structure becomes more defined and the algorithms/procedures which operate on the data are identified and developed. Therefore, data structure diagrams and a calling sequence diagram which shows the interrelationships of the procedures become two additional basic components of documentation.

As the design matures and coding begins, other components surface. These include the procedure (job control runstream) necessary to execute the program. It should contain a written explanation of the various steps, as well as the input/output declarations, definitions, and allocations. Another component includes descriptions of the data files which contain examples of reasonable data in terms of its mode and size/length. Source code listings, preferably with cross reference tables become an invaluable component. Machine specifications can also be documented at this time.

By the time the program code is completed and the usual documentation phase begins, current versions of the previously-mentioned components coupled with actual data files, a sample test case, with output, and references to other related documents (i.e., user's guide, design document, supporting technical papers) will provide a full complement of meaningful documentation.

II. SDDL: A TOOL

The Software Design and Documentation Language which was developed by Henry Kleine [6] at the Jet Propulsion Laboratory of the California Institute of Technology, has as its main objective communication. It facilitates communication between all the characters in the software development process (i.e., managers, customers, designers, development programmers, maintenance programmers, and the machine). SDDL automates the documentation task; it processes input (expressed in natural language or source code) and produces formatted, software documentation.

Further, methodologies have been developed for displaying representations of data, project management, and the direct processing of source code. Automatic features provided by SDDL include a table of contents, module reference tree (calling sequence diagram), and cross reference listing. User-defined features include specific cross reference listings and SDDL keyword definitions.

The remainder of this section will present SDDL's capability to address the changing documentation requirements in a program's lifecycle, and relate them through the various timing scenarios.

A. SDDL AS A DOCUMENTATION TOOL FOR THE DESIGN PHASE ("BEFORE THE FACT")

SDDL has been used successfully to design both SIMSCRIPT and FORTRAN programs. The processing of natural language statements allows a high-level description of the program's function, limitation, and assumptions, which does not have to meet typical programming language syntactical requirements. Figure 1 [7] demonstrates this capability. Capturing the physical representation of the data is also facilitated by SDDL. Figure 2 [7] illustrates this capability. Figure 3 [7] shows a refined design of a data structure.

Automatic features of the SDDL processor include:

- a table of contents
- a module reference tree (forward-calling sequence diagram)
- a module cross reference listing
- logic error detection

Figure 4 [7] is a segment of an automatically-generated module reference tree; it provides information regarding the interrelationships of the various, identified program procedures. A glossary of terms can also be facilitated by the SDDL processor.

B. SDDL AS A DOCUMENTATION TOOL FOR THE CODING PHASE ("CONCURRENT")

SDDL provides a working vehicle which facilitates the coding phase of software development (see Figure 5 [7]). This is the point at which coding conventions can be adopted to allow for the direct processing of source code; and the external data file representation can be documented. Procedures for flagging revisions can be instituted; and user-specified cross reference listings (for global variables, data files, footnotes, etc.) can assist the development programmer. Project management techniques (including a calendar of events, milestones, and progress charts) can be incorporated into the document for use during this phase.

C. SDDL AS A TOOL FOR THE DOCUMENTATION PHASE ("AFTER THE FACT")

Two methodologies for using SDDL during the documentation phase have been developed. The first is the direct processing of SIMSCRIPT, and other high-level languages, source code through SDDL; the second is using the SDDL processor to generate a supporting document to existing source code listings.
1. The Direct Processing of SIMSCRIPT Source Code

Figure 6 is the result of processing SIMSCRIPT source code through SDDL. The document formatting features enhance both the clarity and flow of control in this routine. User-defined cross reference listings can be generated at this point to capture machine portability considerations and I/O devices. The automatically-generated SDDL features provide additional information about the source code.

2. As a Supporting Document for an Existing Program

SDDL can be used to document existing software written in any programming language. Figure 7 [8] shows SDDL being used to capture the physical data representation of an existing FORTRAN program. Meaningful identifiers have been added to clarify the cryptic descriptors; and their instances can be gathered in a user-specified cross reference table. Additionally, variable mode and units of measure have been supplied.

Figure 8 [8] illustrates SDDL being applied to capture the structure/algorithm, at a high level, of an existing FORTRAN program. Natural language statements lend clarity to the routine description; automatic document formatting features lend flow of control visibility.

III. CONCLUSIONS

SDDL can take some of the "drudge" out of documentation by capturing meaningful information during the various phases of software development as well as by transferring the burden to the computer. It allows for details, usually recalled from someone's memory at the end of a project, to be recorded as they occur during the project. This single, automated medium facilitates the communication between the various members in the software development process over time, thereby providing a two-dimensional documentation tool (i.e., between people, over time).

Additionally, SDDL provides a framework for implementing documentation standards. It skews the documentation effort away from the end (when developers are very busy verifying, debugging, and testing) and toward the beginning of the project (when developers are less busy).

When used in a "concurrent" documentation mode, SDDL provides a working vehicle which begins as a designer's tool, then becomes a development programmer's tool, and finally emerges as a maintenance programmer's tool. When used in an "after the fact" documentation mode, SDDL can produce a document in support of existing source code which adds clarity and visibility into the program's actions. Further, SDDL can generate documentation directly from source code.

REFERENCES


LINE 481 PROGRAM OBJECTIVES
482 ************************************************************************************************
483 × SAMIS III IS AN ECONOMIC MODEL OF A HYPOTHETICAL U.S. INDUSTRY TO MANUFACTURE SILICON SOLAR
484 × MODULES, WHICH ARE USED TO GENERATE ELECTRICITY DIRECTLY FROM SUNLIGHT BY THE PHOTOELECTRIC EFFECT.
485 ×
486 × IT IS INTENDED THAT THE SAMIS III PROGRAM FACILITATE STANDARDIZED COMPARISON OF THE RELATIVE
487 × ECONOMICS OF COMPETING MANUFACTURING PROCESSES. IT IS ALSO INTENDED THAT IT FACILITATE ASSESSMENT
488 × OF COMPLETE SEQUENCES OF PROCESSES WITH RESPECT TO THE LOW-COST SOLAR ARRAY (LSA) PROJECT
489 × GOALS. FURTHER, IT IS INTENDED TO PROVIDE INFORMATION THAT WILL HELP IN DETERMINING FRUITFUL
490 × AREAS OF RESEARCH.
491 ×
492 × THE INPUTS TO THE 'SAMIS III MODEL FALL INTO SEVERAL GROUPS:
493 ×
494 × A) DESCRIPTIONS OF THE ECONOMIC CHARACTERISTICS OF EACH MANUFACTURING PROCESS/MACHINE
495 × 1) PROCESS PARAMETERS (PRODUCT PRODUCED, RATE, DUTY CYCLE, ETC)
496 × 2) EQUIPMENT COST FACTORS
497 × 3) FACILITIES AND PERSONNEL REQUIREMENTS (PER MACHINE)
498 × 4) BYPRODUCTS PRODUCED AND UTILITIES AND COMMODITIES REQUIRED (PER MINUTE)
499 × 5) PRODUCTS USED IN THE PROCESS (AND THE ASSOCIATED YIELDS)
500 ×
501 × B) DESCRIPTION OF THE TECHNOLOGICAL AND ECONOMIC STRUCTURE
502 × 1) OF FIRMS IN THE INDUSTRY
503 × 2) OF PROCESSES IN EACH FIRM
504 ×
505 × C) STANDARD DATA
506 × 1) PRICES OF PERSONNEL, COMMODITIES, ETC. AS FUNCTIONS OF QUANTITIES
507 × 2) INDIRECT REQUIREMENTS AS FUNCTIONS OF QUANTITIES
508 × 3) RELATIONSHIPS FOR ESTIMATING INITIAL CAPITAL
509 × 4) INFLATION RATES AND OTHER ECONOMIC PARAMETERS
510 ×
511 × D) RUN TIME DATA
512 × 1) RANGE OF DEMANDS FOR PHOTOVOLTAIC POWER
513 × 2) RANGE OF ANOTHER PARAMETER TO BE VARIED (TO BE IMPLEMENTED IN A LATER RELEASE)
514 × 3) "SWITCH" SETTINGS (SUCH AS THE INTEGRAL.MACHINES.FLAG)
515 ×
516 × FROM DESCRIPTIONS OF THE MANUFACTURING PROCESSES, DETERMINISTIC EQUATIONS DESCRIBING THE
517 × MANUFACTURING-COSTS OF EACH PROCESS, AND BUSINESS COSTS OF EACH FIRM HOUSING ONE OR MORE OF THESE
518 × PROCESSES, THE MODEL ESTIMATES THE PRICES THAT MAY REASONABLY BE EXPECTED FOR SOLAR MODULES AND
519 × ANY RECOGNIZABLE INTERMEDIATE PRODUCTS THAT ARE USED IN THEIR MANUFACTURE. COST ELEMENTS ARE
520 × ALLOCATED TO EVERY PROCESS.
521 ×
522 × BY PERFORMING SENSITIVITY ANALYSES OF VARIOUS PARAMETERS INVOLVED IN THE MODEL, AND BY
523 × ANALYZING THE EFFECT ON PRICE OF DIFFERENT INDUSTRY CONFIGURATIONS, INFERENCES CAN BE DRAWN WITH
524 × RESPECT TO RESEARCH PRIORITIES AND THE EFFECTS OF GOVERNMENTAL AND INDUSTRIAL ACTIONS.
525 ×
526 × WHILE SAMIS III WAS DESIGNED FOR THE NASCENT SOLAR ARRAY INDUSTRY, THE METHODOLOGY IS NOT
527 × INDUSTRY DEPENDENT. APPLICATION TO MANUFACTURERS IN OTHER INDUSTRIES CAN BE ACHIEVED BY RATHER
528 × MODEST AUGMENTATION OF THE STANDARD DATA.
529 ×
530 × END
531 ×

Figure 1. SDDL Processing High-level, Natural Language Statements
DATA_STRUCTURE NOMENCLATURE

DATA STRUCTURES ARE DESCRIBED IN TERMS OF ENTITIES, WHICH ARE THE OBJECTS OUT OF WHICH THE MODEL IS CONSTRUCTED; ATTRIBUTES, WHICH DESCRIBE VARIOUS CHARACTERISTICS OF THE ENTITIES; AND SETS, WHICH DESCRIBE STRUCTURED RELATIONSHIPS AMONG THE ENTITIES.

ATTRIBUTES CONTAIN VALUES, USUALLY NUMERIC, BUT SOMETIMES ALPHABETIC OR ALPHANUMERIC. SOME OF THESE VALUES ARE CHARACTERISTIC OF THE PARTICULAR ENTITY THEY DESCRIBE (AS, FOR INSTANCE, THE EXPENSE ITEM: PRICE vs. QUANTITY.TABLE). THIS KIND OF ATTRIBUTE IS IDENTIFIED IN THE DATA STRUCTURE BY AN S (TO MEAN "SAVED" OR "STATIC") AT THE RIGHT HAND MARGIN. VALUES OF THESE ATTRIBUTES ARE STORED WITH THE ENTITY EVERY TIME THE ENTITY IS SAVED ON FILE. OTHER ATTRIBUTE VALUES ARE CALCULATED DURING THE COURSE OF A RUN, AND ARE NOT STORED AS PART OF THE ENTITY DESCRIPTION. THIS KIND OF ATTRIBUTE IS IDENTIFIED BY A D (TO MEAN "DYNAMIC" OR "DERIVED") AT THE RIGHT MARGIN.

SETS ARE "OWNED" BY ENTITIES AND IN CONSEQUENCE, ARE VERY MUCH LIKE ATTRIBUTES. THEY ARE REFERENCED IN THE DATA STRUCTURE LIKE ATTRIBUTES, EXCEPT THAT THEIR NAMES ARE PRECEDED BY THE WORD "SET," THE KIND OF ENTITIES WHICH "BELONG TO" THE SET FOLLOW THE SET NAME AND ARE INDENTED.


IN ADDITION TO THE S OR D DESCRIBED ABOVE, EACH ATTRIBUTE IN THE DATA STRUCTURE DESCRIPTIONS HAS ITS UNITS OF MEASURE AND ITS MODE. THE FOLLOWING CODES ARE USED FOR THE MODES:

(I) INTEGER
(R) REAL
(T) TEXT
(U) UNIQUE TEXT
(C) SINGLE CHARACTER
(V) REAL VECTOR
(Z) 2 BY N TABLE
(M) REAL MATRIX
(T) TEXT VECTOR
(I) INTEGER VECTOR


Figure 2. High-level Data Representation
Figure 3. Lower-level Data Structure Diagram
Figure 4. Automatically-generated Module Reference Tree
Figure 5. Refined Design of a Routine with Project Management Information, Footnotes, and Various Automatic Formatting Features
Figure 6. Actual SIMSCRIPT Source Code Enhanced by the SDDL Processor
Figure 7. SDDL Used to Document Data Structures in an Existing FORTRAN Program
LINE
358 PROGRAM MAIN PROGRAM
359
360 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
361 &
362 & THE MAIN PROGRAM ESSENTIALLY IS USED TO CALL THE PRINCIPAL SUBROUTINES
363 &
364 & $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
365
366 COMMON (ENTRY, SCAR, CONSFT, SETAT, TSTD, TOUTP, TPURC,
367 ROUTE)
368
369 ***** NOTE ***** NO VARIABLES ARE TRANSFERRED IN THE MAIN LINE SUBROUTINES IN
370 "CALL" STATEMENTS* ALL VARIABLES ARE TRANSFERRED IN THE LABELLED COMMONS
371
372 373 LABEL: START
374 NEW INITIALIZE_VARIABLES (INLINE)----------------------------------------( 12)
375
376 PRINT WELCOME MESSAGE
377
378 CALL SUBROUTINE_QUERY TO INPUT CAR PARAMETERS----------------------------------( 13)
379
380 LABEL: NEW_CIRC, RECALCULATE, DRIVE CYCLE
381 CALL SUBROUTINE_DRIVE TO SPECIFY DRIVING CYCLE PARAMETERS------------------( 16)
382
383 CALL SUBROUTINE_BEGIN TO PRINT VEHICLE PARAMETERS------------------------( 20)
384
385 IF CYCLE # THAT IS CONSTANT SPEED IS SELECTED
386 CALL SUBROUTINE_SCHEMA TO CALCULATE FOR CONSTANT SPEED----------------------( 23)
387 CALL SUBROUTINE_SCHEMA TO PRINT RESULTS----------------------------------------( 25)
388
389 ELSE ANY OF THE DRIVING CYCLES HAS BEEN SELECTED
390 CALL SUBROUTINE_SCHEMA TO CALCULATE FOR DRIVING CYCLE------------------------( 27)
391 CALL SUBROUTINE_SCHEMA TO PRINT RESULTS----------------------------------------( 33)
392
393 ENDIF
394
395 CALL SUBROUTINE_PERFORM TO PRINT PERFORMANCE REQUIREMENTS IF FLAG IS SET------------------------( 35)
396
397 CALL SUBROUTINE_CHANGE TO ALLOW USER CHANGES-----------------------------------( 39)
398
399 *YIELD USER CHOICE
400
401 SELECT-CASE PER USER CHOICE
402 CASE 1: LOGOFF
403 CASE 2: RESTART
404 CASE 3: USER WANTS TO CHANGE DRIVING CYCLE
405 CASE 4: USER CHANGED AN INPUT PARAMETER
406 CASE 5: USER CHANGED AN INPUT PARAMETER
407 CASE 6: RECALCULATE, DRIVE CYCLE TO GET A NEW CYCLE
408 CASE 7: RECALCULATE, DRIVE CYCLE TO RECALCULATE THE DRIVING CYCLE
409 PARAMETERS BECAUSE THE VEHICLE PARAMETERS WERE CHANGED
410 ENDSELECT CASE
411
412 ENDPgPRORAM MAIN PROGRAM

Figure 8. SDDL Used to Document a Routine/Algorithm in an Existing FORTRAN Program