AD-A031 806 MITRE CORP BEDFORD MASS F/G 9/2
SPECIFICATIONS FOR SIMON, A SOFTWARE IMPLEMENTATION MONITOR.(U)
SEP 76 A E CORRIGAN, R J FLEISCHER F19628-76-C-0001
UNCLASSIFIED RADC-TR-76-288 NL

1 OF 2
AD A031806

RADC-TR-76-288
Final Technical Report
September 1976

SPECIFICATIONS FOR SIMON, A SOFTWARE IMPLEMENTATION MONITOR

MITRE Corporation

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

SPECIFICATIONS FOR SIMON, A SOFTWARE IMPLEMENTATION MONITOR

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

This report has been reviewed and is approved for publication.

APPROVED: Roger B. Panara

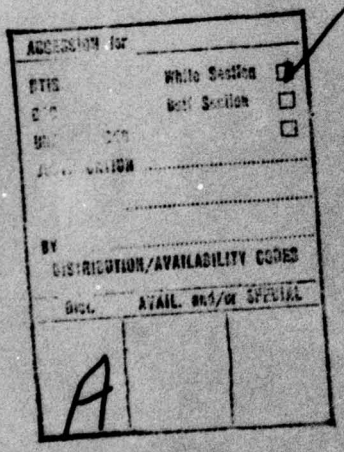
ROGER B. PANARA
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office



Do not return this copy. Retain or destroy.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-76-288	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): SPECIFICATIONS FOR SIMON, A SOFTWARE IMPLEMENTATION MONITOR.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. July 1975 — July 1976,
6. PERFORMING ORG. REPORT NUMBER: N/A	
7. AUTHOR(s): A. E. Corrigan, R. W. Spitler, R. J. Fleischer, Joseph E. Sullivan	8. CONTRACT OR GRANT NUMBER(s): F19628-76-C-0001 ESD
9. PERFORMING ORGANIZATION NAME AND ADDRESS: MITRE Corporation, Bedford MA	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 63728F, 55500801
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIM), Griffiss AFB NY 13441	12. REPORT DATE: September 1976
	13. NUMBER OF PAGES: 130
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Roger Panara (ISIM)
ESD Task Monitor: Capt Samuel L. Ruple

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Data Collection
Project Management

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

Specifications are given for a phototype of Simon, a tool for technical and managerial visibility during software development. Planned and possible extensions of the prototype are also presented, and some uses of the system are illustrated.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

235 050

EVALUATION

The objective of this work was to develop the specification for a software system to provide technical and managerial visibility into the software development process, and to provide for the systematic and consistent collection of data for research into factors affecting software quality and cost. The subject report is the design specifications for such a system and was used to build a prototype System Implementation Monitor (SIMON).

SIMON operates within, and helps to maintain a development environment that proceeds in an orderly way from specifications through design, implementation and test. It is recognized that this process is not necessarily linear and single-thread. With the concept of top-down design and implementation, it is normal for the design phase of one level of abstraction to generate specifications for lower levels, thus spawning further development cycles that may or may not proceed in parallel, in phase, with other development cycles.

SIMON makes use of data as available in each phase to track and project progress towards the emerging system, the quality of that system, and expenditures of resources. The system is technically oriented, that is the emphasis is on system size, complexity and reliability, while also accounting for traditional management factors such as costs, budgets and schedules.

Samuel L. Ruple

SAMUEL L. RUPLE, Capt, USAF
ESD Task Monitor

Roger P. Ramara

ROGER P. RAMARA
RADC Project Engineer

ACKNOWLEDGEMENTS

This report presents the results of work done in FY75 under MITRE Project 5220, "Advanced Systems Technology," Task A, "Software Quality." Sponsorship and technical direction of this task originated in the Software Sciences Section (R. Nelson, Chief), Information Processing Branch (F. Tomaini, Chief), within the Information Sciences Division of the Rome Air Development Center (RADC), United States Air Force. R. Robinson and D. White of RADC served as Project Engineers.

In addition to the authors of this report, L. Cheng and S. Morser contributed to the building of Simon. J. Clapp provided early technical direction of the task, and although her responsibilities have become broader she has continued to provide technical review at critical junctures.

On a continuing basis, N. Anschuetz has provided research services to keep the Simon team aware of related developments while immersed in the building effort. Finally, amidst the traditional crush of year-end business, M. Gallo typed the manuscript.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concl.)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Concl.)

LIST OF TABLES

SECTION I

INTRODUCTION

GENERAL GOALS

Simon is intended to serve two principal purposes: (1) technical and managerial visibility of the software development process, and (2) systematic and consistent collection of data for research into factors affecting software quality and cost. To these ends, Simon extracts and records certain information during the course of software design, implementation and test, providing status reports on request. As much of this information as possible is gathered automatically, because of the well-known difficulty of extracting timely, accurate and complete data from human programmers already immersed in the challenging task of producing correct programs.

The potential advantages of such a system, and the broad outlines of the features it should incorporate, have been described in several publications [Clapp and Sullivan, 1974; Clapp, 1974].

CONCEPT OF OPERATION

Simon operates within, and helps to maintain, a development environment that proceeds in an orderly way from specifications through design, implementation and test. It is recognized that this process is not necessarily linear and single-thread; on the contrary, with the concept of top-down design and implementation, it is normal for the design phase of one level of abstraction to generate specifications for lower levels, thus spawning further development cycles that may or may not proceed in parallel, in phase or out of phase, with other development cycles.

A level of abstraction, sometimes called a "module" or "function cluster," is herein called a "subsystem," for want of a better term that also avoids the unwanted connotations of those other terms. Thus a subsystem is a basic unit for tracking purposes, though a subsystem may comprise still more fundamental units, such as individual programs or subroutines. The development cycle of any one such subsystem proceeds through the following phases:

Phase	Product(s)
Initiation	Specifications Estimates of Required Resources
Design	Improved Specifications Improved Estimates

Phase	Product(s)
	Subsystem Design Test Design
Implementation	Improved Estimates Improved Design Improved Test Design Untested Subsystem
Test	Final "Estimates" (equalling the actuals) Tested Subsystem

These distinctions among these phases are not always sharp unless made so by arbitrary definition. (In fact, under one development approach, the design and implementation phases are not distinguished at all.) Nevertheless, those who produce software and those who manage that production must monitor both the use of resources and the quality of the emerging product throughout the cycle, and it is generally useful to take account of the fact that each phase is characterized by different kinds of activity and different items to be examined.

During each phase, Simon makes use of data as available in that phase to track and project progress towards the emerging system, the quality of that system, and expenditures of resources. The system is technically oriented, that is the emphasis is on system size, complexity and reliability, while also accounting for traditional management factors such as costs, budgets and schedules.

BACKGROUND OF THE PRESENT SYSTEM

During the past year, a prototype of Simon has been implemented. The domain of this prototype is somewhat narrower than the full set of possible applications of the Simon concept. First, Simon could in principle be applied at any organizational level. For example, there is no reason that a "subsystem" could not be a major software item such as a compiler; a natural unit of management might then be an entire group and the "manager" might be a contract monitor or second-level manager. In fact, however, the prototype is specifically geared to the needs of the first-level programming group and its immediate manager. A subsystem is then a level of abstraction appropriate for an individual or at most a few individuals to implement. Second, Simon could in principle be applied to any language and system, but in fact the prototype is specifically

tailored to the GCOS/JOVIAL environment.

Also, even within this restricted domain, the present working prototype of Simon is a modest subset of the capabilities originally envisioned, although the flexibility to incorporate other facilities is part of the design. The intent is to extend it gradually, while obtaining direct experience in its use.

An overview of the organization of the prototype is afforded by Fig. 1. The boxes denote major units (actually separate GCOS activities), all of which (except for the compiler itself) make up Simon. The connecting arrows denote information flow, generally by means of files or reports. Specifically: the user (programmer) supplies source code to the precompiler, which generates (reformatted) source code for the compiler and data to be entered into a permanent data base via a transaction processor. After the compiler runs, certain additional information is extracted by a postcompiler for entry into the data base. The user (programmer or manager) may also enter data into the data base "manually," i.e. through the transactor. Reports are prepared from the data base for the user at his request. (Certain reports and listings are also prepared for the user directly by the non-report modules.)

Most of the system is coded in COBOL using IDS (Integrated Data Store), chosen because it affords a sophisticated file system for the data base. The precompiler is coded in JOVIAL.

SCOPE AND PLAN OF THIS REPORT

This report is intended to serve as an external specification of the system, both as already implemented in prototype form (Section II) and in fuller form (Section II plus the extensions discussed in Section IV). In order to help clarify the intent and meaning of the specifications, examples of the use of Simon in context are discussed in Section III.

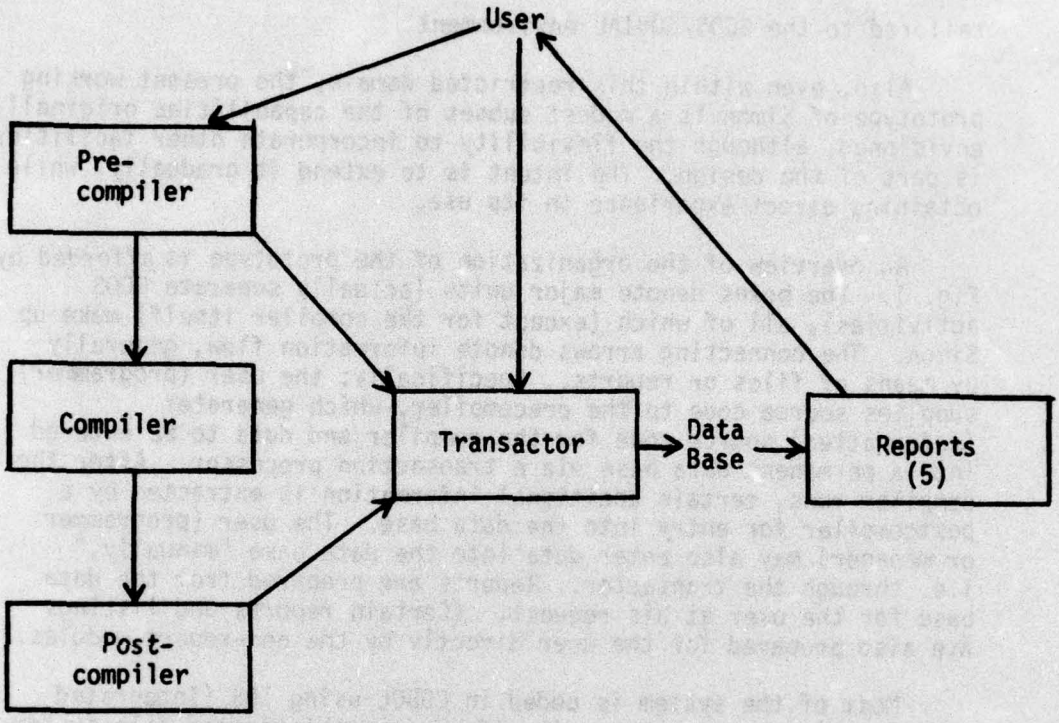


Figure 1

Simon Organization Schematic

SECTION II

PROTOTYPE SPECIFICATIONS

INTRODUCTION

This section details the specifications for the Simon prototype as it is currently implemented. The specifications are given from a user's point of view, so that a user may have an accurate and comprehensive understanding of the Simon system and a general view of how the system is to be used.

Each of the major subsystems is described separately. First those subsystems which supply inputs to the Simon data base - the Precompiler, Postcompiler, and Transactor subsystems - are specified. Next the five Simon reports, which provide the output functions from the data base, are described. These reports are: System Structure, Estimates Vs. Actuals, Project Schedules, Subsystem and Program Status, and Errors and Discrepancies. Within these specifications, explanatory notes on the use of a particular subsystem are included whenever such notes are felt to be necessary or helpful in understanding the subsystem's functions. More extensive notes on using Simon are given in Section III.

In addition to the subsystem specifications, and to provide the user with a more complete understanding of the Simon prototype, a description of the logical structure of the data base is included in Appendix I.

PRECOMPILER SPECIFICATIONS

Functions

The Simon Precompiler analyzes JOVIAL source code prior to compilation and prepares the code for compilation. The primary functions performed by the Precompiler are the formatting of JOVIAL source code, the calculation of various measures of source code length and complexity, and the calculation of other data from the source code.

The formatting function of the Precompiler is intended to highlight the program's control flow and thereby to increase the program's comprehensibility. The formatting is accomplished mainly through a pattern of indentation which serves to isolate and emphasize lines of code which are executed as one block. Each line of source

code is indented a number of columns which is equal to the current level of indentation times the indentation amount, a parameter which is supplied by the user. The level of indentation is initialized to Ø; certain JOVIAL statements then cause the level of indentation to be incremented or decremented before printing the next statement. A list of the indentation rules can be found in Table I.

In addition to the control flow indentation, the following formatting rules are obeyed:

1. Continuations of a statement to a new output line are printed at a level of indentation which is one greater than that of the first line of the statement;

2. Comments are printed on separate lines, at an indentation level which is one greater than that of the following statement;

3. A line is skipped before and after every internal procedure;

4. Statement labels are highlighted by left-shifting them a certain number of columns (again a user-supplied parameter) from the current indentation level; the rest of the statement then begins at the current indentation level or immediately after the printed label, whichever is greater.

Examples of most of the above formatting rules may be found in Figure 2.

As has been noted, the user is given some control over the formatting rules through a set of input parameters to the Precompiler. The user is also given the option of specifying whether or not the Precompiler is to expand JOVIAL "DEFINE" names in the manner of a macro expansion, a function which is not provided by the JOCIT compiler and which may be useful for debugging purposes. The user need not specify any of the input parameters as defaults are provided. A list of the possible input parameters and their defaults is given in Table II. Figure 2 shows a program in its unformatted, formatted, and formatted with expanded DEFINES forms.

Along with its formatting functions, the SIMON Precompiler detects some syntax errors and prints out appropriate error messages. These error messages are interspersed among the source code which is printed out for the user, with each error message applying to the source statement following it. Table III lists the major error

Table I

Indentation Algorithm for Formatting JOVIAL Source Code

Type of Statement	Change in Level of Indentation	Conditions for Change
BEGIN*	no change	if BEGIN immediately follows an IF, IFEITH, or ORIF
	add 1	otherwise
CLOSE	add 1	
END*	no change	if END corresponds to an IFEITH-ORIF sequence
	subtract 1	otherwise
FOR	no change	
IF	add 1	
IFEITH	add 1	
ORIF	add 1	
PROC	add 1	
PROGRAM	add 1	
START	no change	
START PROC	add 1	
all other statement types	subtract 1	if statement immediately follows an IF, IFEITH, or ORIF statement (without intervening BEGIN)
	no change	otherwise

*BEGIN and END are considered as separate statements for formatting purposes.

Sample Program: Unformatted

```
START PROC R'TYPE'NEXT(TOKEN'PTR=TTYPE)$
''RECOGNIZE.R'TYPE REAL CHENG''
DEFINE BEGIN'CASE ''IFEITH''$
DEFINE CASE ''ORIF''$
DEFINE OTHERWISE ''ORIF 1''$
DEFINE END'CASE ''END''$
DEFINE LIST ''I 18 S''$
DEFINE INT ''I 18 S''$
DEFINE BOOL ''B''$
DEFINE TRUE ''1''$
DEFINE FALSE ''Ø''$
DEFINE WHILE ''IF (''$
DEFINE BEGIN'LOOP '') $ BEGIN''$
DEFINE END OF ''GOTO''$
DEFINE LOOP ''$ END''$
ITEM C'PTR INT$
ITEM TOKEN'PTR LIST$
ITEM C'TOKEN INT P 1$
ITEM TTYPE INT $
ITEM TOKEN'TYPE INT P 2$
ITEM FOUND BOOL$
C'PTR=LRETI(TOKEN'PTR,C'TOKEN)$
IF C'PTR GQ Ø$ BEGIN
C'PTR=C'PTR+1$
FOUND=FALSE$
GET'NEXT. WHILE(C'PTR LQ LLENG(TOKENS)) AND NOT FOUND BEGIN'LOOP
TTYPE=LRETI(LRETL(TOKENS,C'PTR),TOKEN'TYPE)$
BEGIN'CASE (TTYPE NQ XCOMMENT) AND (TTYPE NQ XCOMCONT) AND (TTYPE NQ
XEXPANDED'TERM) AND (TTYPE NQ XRECORD'END) AND (TTYPE NQ XD'CODE)$
FOUND=TRUE$
OTHERWISE$ C'PTR=C'PTR+1$
END'CASE
END'OF GET'NEXT LOOP
IF NOT FOUND$
TTYPE= Ø$
END
TERM$
```

Figure 2. Formatting Function of the Precompiler

Sample Program: Formatted

```
START PROC R'TYPE'NEXT(TOKEN'PTR=TTYPE)$
    ''RECOGNIZE.R'TYPE REAL CHENG''
  DEFINE BEGIN'CASE ''IFEITH''$
  DEFINE CASE ''ORIF''$
  DEFINE OTHERWISE ''ORIF 1''$
  DEFINE END'CASE ''END''$
  DEFINE LIST ''I 18 S''$
  DEFINE INT ''I 18 S''$
  DEFINE BOOL ''B''$
  DEFINE TRUE ''1''$
  DEFINE FALSE ''Ø''$
  DEFINE WHILE ''IF (''$
  DEFINE BEGIN'LOOP '') $ BEGIN''$
  DEFINE END'OF ''GOTO''$
  DEFINE LOOP ''$ END''$
  ITEM C'PTR INT$
  ITEM TOKEN'PTR LIST$
  ITEM C'TOKEN INT P 1$
  ITEM TTYPE INT$
  ITEM TOKEN'TYPE INT P 2$
  ITEM FOUND BOOL$
  C'PTR=LRETI(TOKEN'PTR,C'TOKEN)$
  IF C'PTR GQ Ø$
    BEGIN
    C'PTR=C'PTR+1$
    FOUND=FALSE$
    GET'NEXT.  WHILE(C'PTR LQ LLENG(TOKENS)) AND NOT FOUND BEGIN'LOOP
      TTYPE=LRETI(LRETL(TOKENS,C'PTR),TOKEN'TYPE)$
      BEGIN'CASE (TTYPE NQ XCOMMENT) AND (TTYPE NQ XCOMCONT) AND (TTYPE NQ
        XEXPANDED'TERM) AND (TTYPE NQ XRECORD'END) AND (TTYPE NQ XD'CODE)$
        FOUND=TRUE$
      OTHERWISE$
        C'PTR=C'PTR+1$
      END'CASE
      END'OF GET'NEXT LOOP
      IF NOT FOUND$
        TTYPE=Ø$
      END
  TERM$
```

Figure 2. Formatting Function of the Precompiler (Cont.)

Sample Program: Formatted With Expanded Defines

```
START PROC  R'TYPE'NEXT (TOKEN'PTR=TTYPE)$
    ''RECOGNIZE.R'TYPE REAL CHENG''
  DEFINE BEGIN'CASE ''IFEITH''$
  DEFINE CASE ''ORIF''$
  DEFINE OTHERWISE ''ORIF 1''$
  DEFINE END'CASE ''END''$
  DEFINE LIST ''I 18 S''$
  DEFINE INT ''I 18 S''$
  DEFINE BOOL ''B''$
  DEFINE TRUE ''1''$
  DEFINE FALSE ''Ø''$
  DEFINE WHILE ''IF (''$
  DEFINE BEGIN'LOOP '') $ BEGIN''$
  DEFINE END'OF ''GOTO''$
  DEFINE LOOP ''$ END''$
  ITEM C'PTR I 18 S$
  ITEM TOKEN'PTR I 18 S $
  ITEM C'TOKEN I 18 S P 1 $
  ITEM TTYPE I 18 S $
  ITEM TOKEN'TYPE I 18 S P 2 $
  ITEM FOUND B $
  C'PTR=LRETI(TOKEN'PTR, C'TOKEN) $
  IF C'PTR GQ Ø $
    BEGIN
    C'PTR = C'PTR + 1 $
    FOUND = Ø$
    GET'NEXT.  IF ((C'PTR LQ LLENG(TOKENS))AND NOT FOUND)$
      BEGIN
      TTYPE=LRETI(LRETL(TOKENS,C'PTR),TOKEN,TYPE)$
      IFEITH (TTYPE NQ XCOMMENT) AND (TTYPE NQ XCOMCONT) AND (TTYPE NQ
        EXPANDED'TERM) AND (TTYPE NQ RECORD'END AND (TTYPE NQ XD'CODE)$
        FOUND = 1$
      ORIF 1$
        C'PTR=C'PTR + 1 $
      END
      GOTO GET'NEXT $
      END
    IF NOT FOUND $
      TTYPE=Ø$
    END
  TERM$
```

Figure 2. Formatting Function of the Precompiler (Concl.)

Table II

User-Supplied Input Parameters to the Precompiler

Parameter Names*	Data Type	Explanation	Default Value
CMPLX,C	boolean	whether or not to compute the complexity measure of the program	true
CMPTYPE,CT	2 character code	which complexity measure to use	C2
EXPAND,X,EX,E	boolean	whether or not to expand JOVIAL DEFINE names when formatting the source code	false
INDENT,I	integer	when formatting the source code, the number of columns to indent for every level of indentation	2
LABEL,L	integer	when formatting the source code, the number of columns to left-shift labels in order to highlight them from the rest of the statement	Ø
LMARGIN,LMI	integer	left margin of the input (source code) file	1
LMARGOUT,LMO	integer	left margin of the output (formatted source code) file	1
PAGESIZE,PAGE,P	integer	number of lines per page of the formatted source code file	Ø (no automatic page ejects are caused by the Precompiler)

Table II (Concl.)

Parameter Names*	Data Type	Explanation	Default Value
RMARGIN,RMI	integer	right margin of the input (source code) file	72
RMARGOUT,RMO	integer	right margin of the output (formatted source code) file	72

*Several names are listed for each parameter. Any of these names may be used when specifying a value for that parameter.

Table III

Syntax Error Messages Produced by the Precompiler

Error Message	Explanation
CONTROL CARD IN SOURCE CODE	A dollar sign was found in the first column of the source file; this will signal a job control card to GCOS.
DOLLAR SIGN IN COMMENT	A dollar sign was found within a comment other than in the forms "($" or "$)".
END OF FILE WITHIN COMMENT	The end of the source file was encountered within a comment.
END-FILE WITHIN STATEMENT	The end of the source file was encountered in the middle of a JOVIAL statement.
END-FILE WITHIN SYMBOL	The end of the source file was found within a JOVIAL symbol (a "word" in the language).
ILLEGAL OCTAL CONSTANT	The symbol O( ) was found.
INCOMPLETE OCTAL CONSTANT	Octal constant was not completed by a close parenthesis.
INCOMPLETE STATUS CONSTANT	Status constant was not completed by a close parenthesis.
NO FOR VARIABLE IN FOR STMT	No FOR (loop) variable follows the FOR keyword.
NO LABEL FOLLOWING GOTO	(Obvious)
NO PROCEDURE NAME GIVEN	A "START PROC," "PROC" or "CLOSE" statement was found with no procedure name following.
NON-JOVIAL CHARACTER FOUND	A character not allowed by the JOVIAL language was found, outside of a comment.

Table III (Concl.)

Error Message	Explanation
PRIME LEADING NON-PRIMITIVE	An identifier was found which began with a prime (') but which was not a keyword.
STMT NOT IN IFEITH SCOPE	An ORIF was found which did not correspond to any IFEITH statement.
TWO DEFINES IN STATEMENT	Two DEFINE keywords were found in one statement.
TOO FEW DOTS	A string of the form ".." or ". C" (C any character) was found.
TOO MANY CLOSE PARENTHESIS	(Obvious)
TOO MANY DOTS	A string of four dots was found.

messages and any necessary explanations.

The second major function of the Precompiler is that of calculating measures of length and complexity of the JOVIAL source code. These measures are:

1. number of JOVIAL statements,

2. number of lines printed,

3. Halstead length - a count of the operators and operands in a program.

"Hooks," i.e. calls to stubs, have been left in the code for the calculation of a psychological measure of complexity, the C2 measure developed at MITRE [Sullivan, 1973], although this capability has not yet been implemented. These measures provide several indicators of a program's complexity, which is believed to be correlated with the program's comprehensibility and incidence of errors. The measures are ultimately stored in the data base and are used in the Subsystem and Program Status Report.

In addition to these measures, other data is calculated to be stored in the data base. The program's name, the name of the subsystem to which it belongs, and its type - whether a real program, a driver, a stub, or a COMPOOL - are obtained from a user-supplied comment in the source code on the input record directly following the START statement. This comment has the following format:

"<subsystem name>.<program name> <type> <programmer>"

where <program name> is a maximum of 6 characters, <subsystem name> is a maximum of 12 characters, <type> is either "REAL" or "R," "DRIVER" or "D," "STUB" or "S," or "COMPOOL" or "C," and <programmer> is a maximum of 30 characters. The date on which the precompilation is being executed is also calculated. Finally, the list of JOVIAL DEFINE names which was set up for formatting purposes is also kept, to be stored in the data base.

Inputs

The inputs to the Precompiler consist of one file containing the JOVIAL source code to be analyzed and one file containing the user's input parameters. The format of the input parameter file is that of a FORTRAN NAMELIST input, where the name of the NAMELIST is PARMS (see the Honeywell FORTRAN manual [Honeywell BJ67]). The list of possible input parameters is given in Table II; as was noted

above, the user need not specify any parameters as defaults are provided.

Outputs

The outputs from the Precompiler consist of one file containing the formatted source code, ready for input to the compiler, one file containing the computed data to be input by the Transactor subsystem into the Simon data base, and one file for the user containing a listing of the formatted source code and any syntax errors which were detected.

POSTCOMPILER SPECIFICATIONS

Functions

The Postcompiler accepts the output from the JOCIT compiler and analyzes these listings to obtain cross-reference information on the entire system under construction, to be retained in the Simon data base. This data is assembled and printed in the System Structure Report.

The following is a list of the data extracted by the Postcompiler from the compiler listings:

1. date of compilation,
2. name of the program and the name of the subsystem to which it belongs, obtained from the user-supplied comment within the JOVIAL source code (see PRECOMPILER SPECIFICATIONS),
3. a list of the COMPOOL (external) data items referenced, including for each item whether it is set, used, or both,
4. a list of all external procedures called by the program,
5. a list of all files referenced, including for each file its type (hollerith or binary, varying or rigid length records).

In some exceptional cases, no analysis is performed nor is the data base updated, and a message to this effect is printed. These cases are: (1) if the user-supplied identifying comment states that

the "type" of the program is either "STUB" or "DRIVER"; (2) if no identifying comment was found in the source code; and (3) if the XREF option was not specified for the JOCIT compiler activity.

Inputs

The input to the Postcompiler is the JOCIT compiler output (the P* file under the GCOS operating system).

Outputs

The outputs from the Postcompiler include one file for the user containing a printout of the original compiler output, along with any diagnostic messages printed by the Postcompiler, and one file containing the extracted data to be put in the data base by the Transactor subsystem.

TRANSACTION PROCESSOR

Major Function

The Transaction Processor is the only program in Simon that directly updates the data base. A transaction is a single request to add, delete or modify a single item or closely related small group of items. A run of the Transaction Processor typically processes a list of such transactions, each one taken in turn, to produce some larger net effect on the data base.

Table IV lists the various kinds of transactions provided. A fuller functional description of these is given in Appendix II, and input format details are set forth in Appendix III.

The source of a transaction may be either another processor within Simon ("A" for "automatic" in Table IV) or the user directly ("M" for "manual" in Table IV). In the former case, a file containing transactions in the required format is passed from the other processor (Precompiler or Postcompiler) to the Transaction Processor. In the latter case, input preparation is facilitated by forms that prompt the user for the required information while informing the keypuncher of the correct format. Samples of such forms are given in Figure 3 (a-n).

Output from the Transaction Processor consists of a report giving all header cards of processed transactions, any data cards that were to be traced, and any diagnostics that the Processor

Table IV

Transactions for the Data Base

Manual or Automatic Input*	Abbreviation	Name	Remarks
M (f)	DMD	Delete Module	Delete a module (program) from the data base.
M (e)	DPA	Delete Programmer Assignment	Deletes assignments as specified by programmer's name, subsystem assigned, test or design indicator, and reporting period when assigned.
M (g)	DSB	Delete a Subsystem	Deletes a subsystem from the data base.
M (d)	EDC	Enter a Discrepancy Report	
M (a)	EER	Enter an Error Report	
A	EPR	Enter Preprocessor Results	Takes the results of a precompilation and puts them into the data base if the precompiled module is "REAL" or a "COMPOOL."
A	EPS	Enter Post-processor Results	Same as above, but for the Postprocessor.
M (b)	IPR	Initialize the Project's Data Base	
M (c)	ISB	Initialize a Subsystem	
M (h)	IUA	Initialize or Update Actuals	Enters or corrects records of past expenditures.

*For manual inputs, letter in parentheses refers to appropriate form in Figure 3.

Table IV

Transactions for the Data Base (Concl.)

Manual or Automatic Input	Abbreviation	Name	Remarks
M (i)	IUE	Initialize or Update Estimates	Enters or alters current estimates of resources needed to produce a subsystem.
M (n)	IUI	Initialize or Update Interface Data	Used by the programmers to keep the information on general subsystem interactions current.
M (m)	IUP	Initialize or Update Programmer Assignments	Used to change the number of hours in an assignment or to add extra assignments. Any deletion of assignments is performed by the DPA function.
M (k)	UDC	Update a Discrepancy Report	To delete an outstanding discrepancy once it has been resolved.
M (j)	UPR	Update the Project Information	To update most information entered by the IPR transaction. Primarily intended to be used when changing some budgeted resource for a project.
M (l)	USS	Update a Subsystem's Status	For indicating which milestones are passed while completing work on a subsystem. Milestones include completing documentation, completing test plans, etc.

produces. There are three levels of diagnostics: An "ERROR" usually terminates processing at the current logical level and backs up to a point where processing can continue. A "WARNING" indicates a possible error and gives the default action that is being taken. And a "NOTE" simply gives informative messages about the processing. Table V lists all the diagnostics and their probable causes.

Checkpoint/Restart

The Transaction Processor is used in providing a checkpoint/restart facility for Simon. As a normal part of the processing of transactions, a program is run right after the Transaction Processor to append the transactions onto the end of a save file on disk. This file is periodically appended to a save tape. Thus at any point in a project's history all transactions which have been run against the data base are saved either on the tape or on the disk file. (That is, the disk file is a logical extension of the tape.)

Periodically, checkpoints, i.e. dumps of the data base, are taken by means of the system program QUTD. When this happens, a special header card giving the date of the checkpoint is appended to the disk file. These checkpoints allow convenient restoration of the data base.

In restoring the data base, if transactions are to be rerun against the data base, it is of course necessary to run the transactions in exactly the same order as that in which they were originally entered. Restoration of the data base then involves several steps. The IDS file is first restored to a checkpointed state. A transaction manipulator is then run against the saved transactions to extract those needed to produce an up-to-date corrected version of the data base. Table VI gives the commands of the manipulation that can be used. Any editing of transactions is performed at this point. Finally, the Transaction Processor processes these transactions to produce a current data base.

Implementation Note

Because the Transaction Processor is very large, it is actually two processors, one for manual and one for automatic input, and only the one appropriate for the kind of input being supplied is run at any given time. Only in a restart operation, where manual and automatic input may be mixed, is the large combined Processor used.

*START EER (1-10) [1]

ERROR FORM

(Date) (2-7) (Time) (8-11) (Initials) (12-14) [2]

I. How Manifested (16)
- ☐ Discrepancy Form #_____ 1
- ☐ Desk Checking 2
- ☐ Compiler Diagnostics 3
- ☐ Other System Diagnostics 4
- ☐ Test Results 5
- ☐ Other _____ 6

II. How Diagnosed (17)
- ☐ Obvious from Manifestation 1
- ☐ Logic Analysis 2
- ☐ Instrumented Tests 3
- ☐ Other _____ 4

III. Mental Level of Mistake (18)
- ☐ Not Programmer 1
- ☐ Motor 2
- ☐ Memory - 3
- ☐ Memory + 4
- ☐ Logic - 5
- ☐ Logic + 6

IV. Number of Occurrences (19)
- ☐ 1 1
- ☐ 2 2
- ☐ 3 3
- ☐ 4 4
- ☐ 5 5
- ☐ 6-10 6
- ☐ 11-15 7
- ☐ 16-20 8
- ☐ Over 20 9

V. When Occurred (20)
- ☐ In Original Code 1
- ☐ In Making Change 2
- ☐ In Adding Instrumentation 3
- ☐ In Correcting an Error 4
- ☐ Other_____ 5

Time Spent (if significant) _____

---

[3]

Items Involved

(P: = Program)
(S: = Subsystem)

	(8)	(Name of Item) (10-21)
_____		_____
_____		_____
_____		_____
_____		_____
_____		_____
_____		_____

Brief Description (if interesting):

Figure 3a. Error Report Form

*START IPR (1-10) 1

PROJECT INITIALIZATION

Project ID _____	(2-13)	2
Project Start Date ⌊⌋⌊⌋⌊⌋	(15-20)	
Project Stop Date ⌊⌋⌊⌋⌊⌋	(22-27)	
Number of Days in a Reporting Period _____	(29-30)	
Number of Terminal for the Project _____	(31-32)	
File Space for Project (in LLinks) _____	(33-37)	
Budgeted Computer Dollars _____	(38-47)	
Budgeted Other Dollars _____	(48-57)	
Budgeted Man Hours _____	(58-65)	

Figure 3b. Project Initialization Form

*START ISB (1-10) [1]

INITIALIZE SUBSYSTEM

SUBSYSTEM NAME _____ (2-13) [2]

DATE OF DEFINITION ⌊_|_|_|_|_|_⌋ (15-20)

Figure 3c. Subsystem Initialization Form

*START EDC (1-10) [1]

DISCREPANCY REPORT

I. Discrepancy ID

⌊_|_|_|_|_|_⌋ (2-7) (Date) ⌊_|_:_|_⌋ (8-11) (Time) ⌊_|_|_|_⌋ (12-14) (Initials) [2]

II. How Found (16)

- ☐ Code Reading 1
- ☐ System Diagnostic 2
- ☐ Test Results 3
- ☐ Other _____ 4

III. Description: (80 chrs) _____

_____ (1-80) [3]

Figure 3d. Discrepancy Report Form

*START DPA (1-10) [1]

DELETE ASSIGNMENT

REPORTING PERIOD START DATE ______ (2-7) [2]

REPORTING PERIOD STOP DATE ______ (can be blank) (9-14)

ASSIGNED SUBSYSTEM ______________ (16-27)

TEST OR DESIGN ASSIGNMENT ('T' or 'D') ____ (29)

PROGRAMMER'S NAME ______________ (31-65)

REPORTING PERIOD START DATE ______ (2-7) [2]

REPORTING PERIOD STOP DATE ______ (can be blank) (9-14)

ASSIGNED SUBSYSTEM ______________ (16-27)

TEST OR DESIGN ASSIGNMENT ('T' or 'D') ____ (29)

PROGRAMMER'S NAME ______________ (31-65)

REPORTING PERIOD START DATE ______ (2-7) [2]

REPORTING PERIOD STOP DATE ______ (can be blank) (9-14)

ASSIGNED SUBSYSTEM ______________ (16-27)

TEST OR DESIGN ASSIGNMENT ('T' or 'D') ____ (29)

PROGRAMMER'S NAME ______________ (31-65)

REPORTING PERIOS START DATE ______ (2 -7) [2]

REPORTING PERIOD STOP DATE ______ (can be blank) (9-14)

ASSIGNED SUBSYSTEM ______________ (16-27)

TEST OR DESIGN ASSIGNMENT ('T' or 'D') ____ (29)

PROGRAMMER'S NAME ______________ (31-65)

Figure 3e. Assignment Deletion Form

*START DMD	(1-10)	1

DELETE MODULE

MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2
MODULE NAME	_____	(2-7)	2

Figure 3f. Module Deletion Form

*START DSB	(1-10)	1

DELETE SUBSYSTEM

SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2
SUBSYSTEM NAME	_____	(2-13)	2

Figure 3g. Subsystem Deletion Form

*START IUA (1-10) 1

EXPENDITURES PER SUBSYSTEM
(for a particular reporting period)

SUBSYSTEM _____ (2-13) 2

STARTING DATE (of the reporting period) _____ (15-20)

MAXIMUM CORE SPACE (to run the subsystem) _____ (21-28)

PROGRAMMER HOURS

FOR DESIGN _____ (29-34)

FOR TESTING _____ (35-40)

TERMINAL HOURS

FOR DESIGN _____ (41-46)

FOR TESTING _____ (47-52)

COMPUTER EXPENSES

FOR DESIGN _____ (53 62)

FOR TESTING _____ (63-72)

OTHER EXPENSES _____ (8-17) 3

FILE SPACE FOR DESIGNING

TEMPORARY _____ (18-23)

PERMANENT _____ (24-29)

FILE SPACE FOR TESTING

TEMPORARY _____ (30-35)

PERMANENT _____ (36-41)

Figure 3h. Actuals Report Form

*START IUE (1-10) 1

ESTIMATED RESOURCES per subsystem
(or update)

SUBSYSTEM _____ (2-13) 2

NUMBER OF MODULES _____ (15-20)

MAXIMUM CORE SPACE (to run the subsystem) _____ (21-28)

PROGRAMMER HOURS

FOR DESIGN _____ (29-34)

FOR TESTING _____ (35-40)

TERMINAL HOURS

FOR DESIGN _____ (41-46)

FOR TESTING _____ (47-52)

COMPUTER EXPENSES

FOR DESIGN _____ (53-62)

FOR TESTING _____ (63-72)

OTHER EXPENSES _____ (8-17) 3

FILE SPACE FOR DESIGNING

TEMPORARY _____ (18-23)

PERMANENT _____ (24-29)

FILE SPACE FOR TESTING

TEMPORARY _____ (30-35)

PERMANENT _____ (36-41)

Figure 3i. Estimates Report Form

*START UPR	(1-10)	1

UPDATES TO PROJECT INFORMATION
(no entry implies no update)

NEW PROJECT ID _____	(2-13)	2
NEW PROJECT STOP DATE _____	(15-20)	
TOTAL BUDGETED COMPUTER DOLLARS _____	(22-31)	
TOTAL BUDGETED OTHER DOLLARS _____	(33-42)	
TOTAL BUDGETED MAN HOURS _____	(44-51)	
NUMBER OF TERMINALS _____	(53-54)	
TOTAL BUDGETED LLINKS _____	(56-60)	

Figure 3j. Project Update Form

*START UDC (1-10) 1

DISCREPANCY DISPOSITION

I. Discrepancy ID:

(must correspond to a DISCREPANCY REPORT ID)

(Date) (2-7) (Time) (8-11) (Initials) (12-14) 2

II. Nature of Discrepancy: (16)

- ☐ Error Form(s) #_____ 1
- ☐ Lapse in Communication 2
- ☐ Not a Discrepancy 3
- ☐ Other 4

Figure 3k. Discrepancy Update Form

*	S	T	A	R	T		U	S	S

(1-10) [1]

SUBSYSTEM MILESTONE STATUS UPDATE

SUBSYSTEM NAME ______________________________ (2-13) [2]

DATE __________ (15-20)

TEST PLAN DONE? (22)

- ☐ YES 'Y'
- ☐ NO 'N'
- ☐ NO CHANGE ' '

DOCUMENTATION DONE? (24)

- ☐ YES 'Y'
- ☐ NO 'N'
- ☐ NO CHANGE ' '

INCLUDED PROGRAMS TEST STATUS:

PROGRAM NAME ______________________________ (8-13) [3]

TEST STATUS: (15)

☐ NOT DONE 'N' ☐ PARTIALLY COMPLETE 'P' ☐ FINISHED 'T'

PROGRAM NAME ______________________________ (8-13) [3]

TEST STATUS: (15)

☐ NOT DONE 'N' ☐ PARTIALLY COMPLETE 'P' ☐ FINISHED 'T'

PROGRAM NAME ______________________________ (8-13) [3]

TEST STATUS: (15)

☐ NOT DONE 'N' ☐ PARTIALLY COMPLETE 'P' ☐ FINISHED 'T'

PROGRAM NAME ______________________________ (8-13) [3]

TEST STATUS: (15)

☐ NOT DONE 'N' ☐ PARTIALLY COMPLETE 'P' ☐ FINISHED 'T'

Figure 31. Subsystem Update Form

*START IUP (1-10) [1]

PROGRAMMER'S SCHEDULE
(or update)

AS OF ⌊⌋⌊⌋⌊⌋ (2-7) [2]

PROGRAMMER_____ (9-43)

REPORTING PERIOD START DATE ⌊⌋⌊⌋⌊⌋ (8-15) THRU ⌊⌋⌊⌋⌊⌋ (15-20) [3]

SUBSYSTEM ASSIGNED _____ (22-33) TEST OR DESIGN_____ (34)
("T" or "D")

NUMBER OF HOURS ASSIGNED IN PERIOD _____ (36-39)

REPORTING PERIOD START DATE ⌊⌋⌊⌋⌊⌋ (8-15) THRU ⌊⌋⌊⌋⌊⌋ (15-20) [3]

SUBSYSTEM ASSIGNED_____ (22-33) TEST OR DESIGN_____ (34)

NUMBER OF HOURS ASSIGNED IN PERIOD _____ (36-39)

REPORTING PERIOD START DATE ⌊⌋⌊⌋⌊⌋ (8-15) THRU ⌊⌋⌊⌋⌊⌋ (15-20) [3]

SUBSYSTEM ASSIGNED_____ (22-33) TEST OR DESIGN_____ (34)

NUMBER OF HOURS ASSIGNED IN PERIOD _____ (36-39)

REPORTING PERIOD START DATE ⌊⌋⌊⌋⌊⌋ (8-15) THRU ⌊⌋⌊⌋⌊⌋ (15-20) [3]

SUBSYSTEM ASSIGNED _____ (22-33) TEST OR DESIGN_____ (34)

NUMBER OF HOURS ASSIGNED IN PERIOD _____ (36-39)

REPORTING PERIOD START DATE ⌊⌋⌊⌋⌊⌋ (8-15) THRU ⌊⌋⌊⌋⌊⌋ (15-20) [3]

SUBSYSTEM ASSIGNED _____ (22-33) TEST OR DESIGN_____ (34)

NUMBER OF HOURS ASSIGNED IN PERIOD _____ (36-39)

Figure 3m. Programmer Assignment Form

*	S	T	A	R	T		I	U	I

(1-10) [1]

INTERFACE INFORMATION
(or updates)

SUBSYSTEM NAME ______________________ (2-13) [2]
(for which information is being specified)

CAN A DRIVER BE USED TO TEST LOWER LEVELS ________ (15)
('Y', 'N', or blank)

RELATED SUBSYSTEM ________ (8-19) [3]

IS THIS AN:
☐ ADDITION 'A' ☐ CHANGE 'C' ☐ DELETION 'D' (21)

RELATIONSHIP TO ABOVE SUBSYSTEM
☐ CALLED BY ABOVE 'CD' ☐ CALLS ABOVE 'CG' (23-24)
☐ SHARES DATA 'SD' ☐ NO CHANGE ' '

REQUIRE STATUS OF THIS SUBSYSTEM TO THE ABOVE
☐ MUST BE COMPLETED 'T' ☐ CAN BE STUB 'S' ☐ NO CHANGE ' ' (26)

RELATED SUBSYSTEM ________ (8-19) [3]

IS THIS AN:
☐ ADDITION 'A' ☐ CHANGE 'C' ☐ DELETION 'D' (21)

RELATIONSHIP TO ABOVE SUBSYSTEM
☐ CALLED BY ABOVE 'CD' ☐ CALLS ABOVE 'CG' (23-24)
☐ SHARES DATA 'SD' ☐ NO CHANGE ' '

REQUIRE STATUS OF THIS SUBSYSTEM TO THE ABOVE
☐ MUST BE COMPLETED 'T' ☐ CAN BE STUB 'S' ☐ NO CHANGE ' ' (26)

RELATED SUBSYSTEM ________ (8-19) [3]

IS THIS AN:
☐ ADDITION 'A' ☐ CHANGE 'C' ☐ DELETION 'D' (21)

RELATIONSHIP TO ABOVE SUBSYSTEM
☐ CALLED BY ABOVE 'CD' ☐ CALLS ABOVE 'CG' (23-24)
☐ SHARES DATA 'SD' ☐ NO CHANGE ' '

REQUIRE STATUS OF THIS SUBSYSTEM TO THE ABOVE
☐ MUST BE COMPLETED 'T' ☐ CAN BE STUB 'S' ☐ NO CHANGE ' ' (26)

RELATED SUBSYSTEM ________ (8-19) [3]

IS THIS AN:
☐ ADDITION 'A' ☐ CHANGE 'C' ☐ DELETION 'D' (21)

RELATIONSHIP TO ABOVE SUBSYSTEM
☐ CALLED BY ABOVE 'CD' ☐ CALLS ABOVE 'CG' (23-24)
☐ SHARES DATA 'SD' ☐ NO CHANGE ' '

REQUIRE STATUS OF THIS SUBSYSTEM TO THE ABOVE
☐ MUST BE COMPLETED 'T' ☐ CAN BE STUB 'S' ☐ NO CHANGE ' ' (26)

Figure 3n. Interface Information Form

Table V

Transaction Processor Errors

ERROR ID	LEVEL	TEXT	PROBABLE CAUSES
TP 01	ERROR	INVALID HEADER CARD ON TRANSACTION	Number in function field or letter in date field.
TP 02	ERROR	INVALID FUNCTION FIELD ON HEADER CARD	Function specified is not a legal one.
TP 03	ERROR	UNSUCCESSFUL ATTEMPT TO STORE RECORD	Information in terms of an IDS record was not able to be stored in the data base. This could be a result of duplicate information, no more room or any severe type of IDS malfunction.
TP 04	ERROR	INPUT DATE HAS INVALID M/D/Y VALUE	A date on a card in the transaction has an invalid month, day, or year value.
TP 05	ERROR	STOP PRECEDES START DATE IN TRANSACTION	For two dates expressing a range in a transaction, the stop date is before the start date.
TP 06	ERROR	ILLFORMED NUMBER IN ALL NUMERIC FIELD	In a field that must be all numbers there is a non numeric character.
TP 07	ERROR	TRANSACTION CONTINUATION DATA NOT FOUND	While processing a transaction, a logical termination point for the transaction was not reached before the next transaction was read.
TP 08	NOTE	END OF TRANSACTIONS	EOF encounter at a proper time.

Table V (Cont.)

ERROR ID	LEVEL	TEXT	PROBABLE CAUSES
TP 09	ERROR	UNABLE TO RETRIEVE PROJECT RECORD	Project has not been initialized or some serious malfunction of IDS.
TP 10	ERROR	LOOKING FOR ASSIGNMENT NAME CARD	In the IUP transaction, the AS-OF-DATE and the PROGRAMMER's name are improperly entered.
TP 11	ERROR	INVALID CHARACTER(S) IN ALPHABETIC FIELD	For fields having only certain allowable alphabetic entries, the transaction has a wrong character entry.
TP 12	ERROR	UNABLE TO RETRIEVE SUBSYSTEM	Subsystem has not been initialized, name is invalid, or IDS system malfunction.
TP 13	ERROR	DATE DOES NOT CORRESPOND TO A REPORT PERIOD	An input date used to specify a certain reporting period, i.e., the first day of a reporting period, does not match a period computed from the starting date of the project and the length of a reporting period.
TP 14	ERROR	AS OF DATE IS AFTER TRANSACTION DATE	In IUA the date of an actual expenditure of resources is after the date of the transaction on the header card. This violates the concept of a past expenditure.
TP 15	ERROR	UNABLE TO RETRIEVE FROM DATA BASE	General message used when data or record to be updated or changed or in anyway accessed has not yet been entered into the data base. It is possible that an IDS system malfunction could cause it.
TP 16	ERROR	DATA BASE INCONSISTENCY	Should not occur; indicates Simon or IDS malfunction.

Table V (Cont.)

ERROR ID	LEVEL	TEXT	PROBABLE CAUSES
TP 17	NOTE	NOT REAL FLAG, DATA BASE IS UNCHANGED	Precompiled or postcompiled program hasn't been declared "REAL" or "COMPOOL" and consequently hasn't caused a data base update.
TP 18	ERROR	PREMATURE END OF FILE	EOF was encountered before it was logically expected.
TP 19	WARNING	SEARCHING FOR VALID '*START' CARD	Transaction Processor is scanning for a header card but encountered other cards instead.
TP 20	ERROR	DATA BASE PROBLEM EXCLUDES INITIALIZATION	While trying to initialize the data base, IDS signalled an error condition which implies that either IDS space has not been allocated, or it has not been initialized by the IDS program QUTI, or there is an IDS malfunction.
TP 21	ERROR	PROJECT HAS ALREADY BEEN INITIALIZED	(Self-explanatory).
TP 22	NOTE	MODULE UPDATED IN DATA BASE	Postcompiler or precompiler results have been entered in the data base.
TP 23	ERROR	BLANK FIELD WHERE ENTRY REQUIRED	A blank field occurs where it shouldn't.
TP 24	ERROR	UNABLE TO DELETE RECORD	Either the information that is being deleted does not exist or else there is some system malfunction.
TP 25	ERROR	MODULE NOT YET COMPILED	In an attempt to update the status information, the program was not able to access

Table V (Concl.)

ERROR ID	LEVEL	TEXT	PROBABLE CAUSES
			module information in the data base implying that the module hasn't yet been compiled.
TP 26	ERROR	NO SUCH ENTRY IN DATA BASE	No such program or subsystem in the data base.
TP 27	NOTE	MODULE ADDED TO DATA BASE	The module has been either precompiled or postcompiled for the first time.
TP 28	ERROR	GIVEN DATE PRECEDES PROJECT START DATE	Date on error or discrepancy ID precedes the start of the project.
TP 29	ERROR	NUMBER FOUND IS OUT OF RANGE	For those characteristics of errors and discrepancies listed on the forms, there is a legal range; some entry is out of its range.
TP 30	ERROR	NO SUCH DISCREPANCY RECORD	When performing the UDC function, the discrepancy ID does not correspond to an outstanding discrepancy ID.
TP 31	NOTE	SKIPPING TRANSACTION	The Transaction Processor will not process this transaction.
TP 32	ERROR	UNABLE TO FIND ASSIGNMENT RECORD	While deleting a programmer's assignment (DPA), an assignment can't be found.

For the messages that imply difficulty in retrieving storing or deleting IDS records, additional IDS information is printed out on the output. For documentation, see the IDS Manual [Honeywell BR 69, Appendix B].

Table VI

Transaction Manipulation Commands

Parentheses imply that the phrase can be omitted.

The commands can be batched in a run.

I. COPY ((FROM (CHECKPOINT) date) (TO (CHECKPOINT) date))

1. COPY alone copies the entire file.

2. FROM, if present, sets up the starting position for the copy. IF not present, then the starting point is the present position of the pointer in the file.

3. TO, if present, sets up the terminating position for the copy; otherwise it is the end of the file.

II. DELETE (number)

1. Starting at the present position, DELETE skips over records.

2. If "number" is present then that number of records are skipped. If absent then one record is skipped.

III. INSERT (number)

1. INSERT inserts cards into the new file. The cards directly follow the INSERT card.

2. If "number" is present then that number of cards are inserted. If absent then one card is inserted.

SYSTEM STRUCTURE REPORT SPECIFICATIONS

Introduction

The first Simon report supplies management and programmers with a comprehensive view of the structure of the developing system. It provides cross-reference information such as calling hierarchies and data and file references in a variety of forms so that the user can easily determine the relationships of one subsystem or program with another subsystem or program for purposes of data or control flow analysis and as an aid to detecting and correcting interface errors.

The data for this report comes indirectly (via the data base) from the analyses performed by the Precompiler and Postcompiler on JOVIAL source code. The Precompiler supplies program names and the names of the subsystems to which they belong, and a list of JOVIAL DEFINE names referenced by each program. The Postcompiler also provides program names and the names of their associated subsystems, and in addition supplies the program calling hierarchy information and a list of external data items (COMPOOL items) and files referenced by each program. Some data is manually collected. This is the "Subsystem Declared Interdependency Data," which notes for each subsystem whether a driver is feasible for that subsystem, and supplies for each subsystem a list of all subsystems which are needed for testing the former subsystem, their relationship (called, calling or shares data) to the former subsystem, and whether stubs for these subsystems may be used. This information is collected via the interface information data form (see TRANSACTOR SPECIFICATIONS).

The System Structure Report consists of seven sections, which are:

1. Program-Subsystem Dictionary,
2. Subsystem Declared Interdependency Data,
3. References by Subsystem-Program,
4. References by COMPOOL Item,
5. References by File,
6. References by DEFINE Name,
7. Subsystem Interaction.

Most of the sections are in table form, and in most cases the entries within a column, whether subsystems, programs, data items, files, or DEFINE names, are arranged in alphabetical order, starting with the left-most column.

Program-Subsystem Dictionary

The first section, Program-Subsystem Dictionary, lists for every program in the data base the subsystem to which it belongs. The subsystem name is flagged if it is not currently in the data base. Figure 4 below shows a sample Program-Subsystem Dictionary.

PROGRAM-SUBSYSTEM DICTIONARY

PROGRAM	SUBSYSTEM
AA1	AA
AA2	AA
BBC	BB
BDXW	AA
CC1	CC
CC2	CC
DZQR	BB

* - NAME NOT PRESENTLY IN VALID SUBSYSTEM NAME LIST

Figure 4. Program-Subsystem Dictionary Section of System Structure Report

Subsystem Declared Interdependency Data

The second section of the System Structure Report is the Subsystem Declared Interdependency Data section described above. A list of the subsystems in the data base is printed, giving for each subsystem:

1. whether or not a driver is feasible ("Y" if yes, "N" if no, "Ø" if this field is absent in the data base);

2. a list of all those other subsystems which have been declared to be needed for testing the subject subsystem including:

   a. the relationship of the needed subsystem to the subject subsystem ("CD" if the needed subsystem is called by the subject subsystem, "CG" if the needed subsystem calls the subject subsystem, and "SD" if the needed subsystem sets any external data items used by the subject subsystem),

   b. whether or not a stub for the needed subsystem is feasible ("S" if stub is feasible, "T" if the total needed subsystem must be used).

Figure 5 gives an example of the Subsystem Declared Interdependency Data Section.

SUBSYSTEM DECLARED INTERDEPENDENCY DATA

REL - RELATIONSHIP OF NEEDED SUBSYSTEM

REQ - REQUIRED STATUS FOR TESTING

SUBSYSTEM	DRVR OK	SUBSYSTEM NEEDED	REL	REQ
ABAD	Y			
ACLO	N			
		ACCA	CD	S
		AXYA	CG	T
BADO	Y	BAMS	SD	T
CICD	Ø			

Figure 5. Subsystem Declared Interdependency Data Section of System Structure Report

References by Subsystem-Program

The third section, References by Subsystem-Program, is a comprehensive list, for every program in the data base, of all programs, COMPOOL data items, files, and DEFINE names which have any relationship to that program. The format of this section is as follows (see Figure 6). (It should be remembered that all columns of items are arranged alphabetically.)

1. The left-hand column lists one of the subsystems in the data base.

2. The second column lists one of the programs contained in the subsystem in column 1.

3. The third column, "RELATIONSHIP," states the relationship of the program in column 2 to the item in column 4, "OBJECT", which may be a program, a COMPOOL (COMMON) item, a file, or a DEFINE name (MACRO).

4. The fifth column, "WITHIN," gives:

   a. if the item in column 4 is a program, the subsystem to which it belongs (if the subsystem is not known "**" is printed);

   b. if the item in column 4 is a COMPOOL item, the COMMON to which it belongs;

   c. if the item in column 4 is a file or a DEFINE name, this column is blank.

4. The last column, "TYPE," gives the type of the item in column 4, if the item is a COMPOOL item or a file. The abbreviations used for the item types are those used in the JOCIT compiler cross-reference listings; a list of these abbreviations is given in Table VII.

Table VIII below gives a list of all possible relationships between the program in column 2 and the item in column 4. The program in column 2 is denoted as "X". Figure 6 shows an example of a References by Subsystem-Program Section.

REFERENCES BY SUBSYSTEM-PROGRAM

SUBSYSTEM	PROGM	REFERENCE	TO OR BY (OBJECT ITEM)	WITHIN	TYPE
CHARLIE	STOOP	CALL TO	STOKE	DADA	
			KREPE	**	
		CALL BY	PREAK	SUPRA	
		S/U COMMON	FORMIT	COMMN1	IS
DADA	STOKE	CALL BY	STOOP	CHARLIE	
		USE COMMN	FORMIT	COMMN1	IS
			CHAR1	COMMN2	H
		R/W FILE	FILE6		HV
		USE MACRO	DO'LOOP		
			CASE		
SUPRA	PREAK	CALL TO	STOOP	CHARLIE	
		SET COMMN	FORMIT	COMMN1	IS
			CHAR1	COMMN2	H
		R/W FILE	FILE7		BR
			FILE8		BR
		USE MACRO	DO'LOOP		
	COMPOL1	REF COMMN	FORMIT	COMMN1	IS
			CHAR1	COMMN2	H

Figure 6. References by Subsystem-Program Section of System Structure Report

Table VII

List of Abbreviations Used for File and Data Item Types in Reports

Abbreviations for File Types

Abbreviation	Expansion
BR	binary code, rigid (fixed) length records
BV	binary code, varying length records
HR	hollerith code, rigid length records
HV	hollerith code, varying length records

Abbreviations for Data Items

A	fixed, signed
AU	fixed, unsigned
B	boolean
F	floating point
H	hollerith code
IS	integer, signed
IU	integer, unsigned
S	status variable
T	transmission code

Table VIII

Explanation of References by Subsystem-Program Section of System Structure Report

REFERENCE	OBJECT ITEM	WITHIN
CALL TO	program called by X*	subsystem
CALL BY	program which calls X	subsystem
SET COMMN	COMPOOL data item whose value is set by X, where X is a procedural program	COMMON
USE COMMN	COMPOOL data item whose value is used by X, where X is a procedural program	COMMON
S/U COMMN	COMPOOL data item whose value is both set and used by X, where X is a procedural program	COMMON
REF COMMN	COMPOOL data item defined by X, where X is a COMPOOL rather than a procedural program	COMMON
R/W FILE	file referenced by X	-
USE MACRO	JOVIAL DEFINE name referenced by X	-

*X is the program listed in column 2 of the References by Subsystem-Program Section.

References by COMPOOL Item

The remaining sections of the System Structure Report are based on the same information as the References by Subsystem-Program Section, but are keyed on different items. Section 4, References by COMPOOL Item, is keyed on external (COMPOOL) data items. It lists all COMPOOL items in alphabetical order and gives for each COMPOOL item:

1. the COMMON in which it is found,

2. the type of the COMMON data item (see Table VII for list of abbreviations),

3. all subsystems which reference that COMMON item and all programs in each subsystem which reference the item, noting whether each program sets ("S"), uses ("U"), or both sets and uses ("B") the COMMON item (if the "USE" column is blank this indicates that the referencing program is the COMPOOL in which the COMMON item is declared).

Figure 7 below gives a sample of a References by COMPOOL Item Section.

REFERENCES BY COMPOOL ITEM

ITEM	COMMON	TYPE	SUBSYSTEM	PROGM	USE
ALPHA	CHRCLS	IS	AA	ABAD	U
				AXYL	S
			BOX	BYLL	B
ALLY	PERCLS	H	ALKA	ALSOM	
			BOX	BYLL	S

Figure 7. References by COMPOOL Item Section of System Structure Report

References by File

The fifth section of the System Structure Report is the References by File Section. It lists all files in the system in alphabetical order, giving for each file:

1. the file's type (see Table VII for list of abbreviations),

2. all subsystems which reference the file and all programs in each subsystem which reference the file.

Figure 8 below is an example of the References by File Section.

REFERENCES BY FILE

FILE	TYPE	SUBSYSTEM	PROGM
FILE7	HR	SUB'A	PROG'AA
			PROG'AB
		SUB'B	PROG'B
		SUB'C	PROG'CG
FILE9	BV	SUB'B	PROG'B
			PROG'BB

Figure 8. References by File Section of the System Structure Report

References by DEFINE Name

The sixth section of the System Structure Report is called References by DEFINE Name, and lists for each DEFINE name in the data base all subsystems which reference it and all programs in each subsystem which reference the DEFINE name. A sample Reference by DEFINE Name Section is given below in Figure 9.

REFERENCES BY DEFINE NAME

DEFINE-NAME	SUBSYSTEM	PROGM
DEF1	SUB'A	PROG'AA
	SUB'B	PROG'AZ
		PROG'B
	SUB'Z	PROG'ZZ
DEF2Z	SUB'Z	PROG'ZZ

Figure 9. References by DEFINE Name Section of System Structure Report

Subsystem Interactions

The last section, Subsystem Interactions, is a summary of all relationships between subsystems in the system. It consists of a matrix with the names of all subsystems listed in a column on the left-hand side labelling the rows, and also across the top of the matrix, read top-to-bottom and labelling columns. The symbol at the intersection of the row for a given subsystem (subsystem "A") and the column for some, possibly other, subsystem (subsystem "B") denotes their interactions as follows:

1. a "-" indicates no relationship;

2. an "X" denotes that (some program in) Subsystem A calls (some program in) Subsystem B;

3. a "+" denotes that Subsystem A sets a data item which Subsystem B uses;

4. a "*" indicates that both "X" and "+" apply.

This matrix may thus be read across a row to see whom a subsystem calls and who uses items that subsystem sets, and read down a column to see who calls a subsystem and who sets items which are used by that subsystem. Figure 10 below is an example of a Subsystem Interactions Section.

SUBSYSTEM INTERACTIONS

SUBSYSTEM A	SUBSYSTEM B			
	SUB'A	SUB'B	SUB'Z	SUB'Z
	A	B	Z	Z
	•	B	X	Z
	•	•	•	•
	•	•	•	•
	•	•	•	•
	•	•	•	•
	•	•	•	•
	•	•	•	•
SUB'A	+	-	X	-
SUB'BB	X	*	X	-
SUB'ZX	+	-	+	X
SUB'ZZ	-	-	*	+

Figure 10. Subsystem Interactions Section of System Structure Report

ESTIMATES VS. ACTUALS REPORT SPECIFICATIONS

Introduction

The second Simon report is a comparison between projected resource expenditures (estimates) and actual resource expenditures (actuals) for each subsystem and for the entire system. The data is taken from the estimates and actuals forms used in the Transactor subsystem.

The following resources are used in the comparison: person hours, terminal hours, main memory, computer dollars, other (non-computer) dollars and file space. These resources are identical to those listed on the estimates and actuals forms. Expenditures are divided into those incurred during the design phase and those incurred during testing, again as in the estimates and actuals forms.

The estimates used in the comparison are a programmer's current estimates for a subsystem. The actuals are current actual expenditures. A series of Estimates Vs. Actuals Reports will thus show the progression of current estimates and actuals over time. Since the estimates for each subsystem are continually revised and refined throughout the subsystem's development, the series of estimates-actuals comparisons should keep converging until: (1) the current design estimates equal the current design actuals when the design phase has been completed, (or, more accurately, is estimated to have been completed), and (2) the test estimates and actuals are equal, when the subsystem has been tested.

Subsystem Estimates Vs. Actuals

First a comparison between current estimates and actuals is printed for each subsystem in the system. A comparison for a subsystem is printed even if there have been no estimates or actuals entered into the data base for that subsystem; in this case all counts will be zero. Figure 11 below gives an example of the estimates-actuals comparison for the subsystem named "SUBSYSTEM-A." As can be seen, this subsystem has had its design completed but is still being tested.

SIMON REPORT #2 -- ESIMATES VS. ACTUALS AS OF 04/22/75 PAGE 1

SUBSYSTEM-A SUBSYSTEM ESTIMATES VS. ACTUALS

RESOURCE	ESTIMATES		ACTUALS	
	DESIGN	TEST	DESIGN	TEST
PERSON-HOURS	120	65	120	30
TERMINAL-HOURS	45	30	45	11
MAIN MEMORY		26		26
COMPUTER $	$105	$75	$105	$35
OTHER $		$60		$51
FILE SPACE - TEMP	22	6	22	8
- PERM	14	18	14	14

Figure 11. Subsystem Estimates vs. Actuals

Project Estimates vs. Actuals

Finally, the current estimates-actuals comparison is printed for the entire system. This is a sum of the counts for all subsystems for the resources person hours, terminal hours, computer dollars, other dollars and file space. The main memory usage for the entire system is the maximum of the main memory usages for all subsystems. A third column, giving the total project budget for person hours, main memory, computer dollars, other dollars, and file space, is included, so that a comparison may be made between the estimates and actuals for the entire system and the project budget. In this way it can easily be seen if the budget has been overrun (actuals compared with budget), or if it is predicted that the budget will be overrun (estimates compared with budget).

Figure 12 shows a sample system comparison for the project "PROJECT-A." In the example, the budget for computer dollars has already been overrun; the budgets for main memory and file space have not yet been overrun, but are predicted to be overrun.

Terminal-hours are not shown as having a "budget" because this type of resource is more usefully regarded as limited on a per-unit-time basis (e.g. maximum of 16 hrs in any one day) rather than overall. See the paragraph entitled "Projected Resource Overruns," below.

PROJECT-A PROJECT ESTIMATES VS. ACTUALS, COMPARED WITH BUDGET.

RESOURCE	ESTIMATES		ACTUALS		BUDGET
	DESIGN	TEST	DESIGN	TEST	
PERSON-HOURS	600	400	375	300	1000
TERMINAL-HOURS	270	250	200	121	
MAIN MEMORY		36		22	30
COMPUTER $	850	650	800	645	1300
OTHER $		700		300	1000
FILE SPACE - TEMP	80	40	43	32	
- PERM	83	67	43	40	
- TOTAL	163	107	86	72	200

Figure 12. Estimates vs. Actuals for an Entire Project

PROJECT SCHEDULES REPORT SPECIFICATIONS

Introduction

The third Simon report, Project Schedules, deals with schedules of resource usage, including programmer hours, computer dollars, other (non-computer) dollars, terminal hours and file space. The report shows both past (actual) and projected (estimated) resource usage, predicts budget overruns, and checks the programmers' schedules for any inconsistencies and conflicts.

All data used in this report is manually collected and includes the following:

1. programmer schedules,

2. subsystem estimates and actuals,

3. project budgets,

4. subsystem interdependency (interface) data.

The Project Schedules Report consists of five sections: Person Hours Schedules, Other Resource Schedules, Projected Resource Overruns, Projected Scheduling Inconsistencies, and Projected Scheduling Conflicts.

Person Hours Schedules

The first section, Person Hours Schedules, uses the programmer schedules put in the data base by a project manager to print programmer-subsystem schedules for the entire system. A sample Person Hours Schedule Section is given below in Figure 13. This example shows programmer schedules for the period from 3/11/75 to 5/20/75. The line of dates gives the start dates of the periods of time for which assignments have been made. The programmers are listed at the left; under each date in each programmer's rows are given the names of the subsystems on which that programmer will be working, noting the number of hours to be worked and whether design or test will be performed ("D" or "T").

SIMON REPORT III PAGE 2

PROJECT SCHEDULES AS OF 042175

PROJECTED RESOURCE USE

LENGTH OF REPORTING PERIOD - 14 DAYS

PROJECT STOP DATE - 063175

PERSON HOURS

PROGRAMMER	031175	032575	040875	042275	050675
CAPLAN	ABAD	ABAD	ABAD	ABAD	ABAD
	40 HOURS-D	40 HOURS-D	40 HOURS-D	40 HOURS-T	40 HOURS-T
	ZENO	ZENO	ZENO	ZENO	ZENO
	20 HOURS-D	20 HOURS-D	20 HOURS-T	20 HOURS-T	20 HOURS-T
MARDRID	CONO	CONO	CONO	CONO	CONO
	30 HOURS-D	30 HOURS-D	30 HOURS-D	30 HOURS-T	30 HOURS-T
NOSER			KAPLA	KAPLA	KAPLA
			40 HOURS-D	40 HOURS-D	40 HOURS-D

Figure 13. Person Hours Section of Project Schedules Report

Other Resource Schedules

The second part of the Project Schedules Report shows past and projected schedules for resources other than person hours, using the subsystem estimates and actuals input by the programmers and the subsystem schedules produced in the first section of this report. The resources which are covered in this section are computer dollars, file space (number of file blocks), other dollars, and terminal hours. Figure 14 below shows a sample of this section.

SIMON REPORT III PAGE 2

PROJECT SCHEDULES AS OF 020275

OTHER RESOURCES

RESOURCE	123174	011475	012875	021175	022575
COMPUTER DOLLARS	62	165	145	98	148
FILE SPACE	77	344	385	308	423
OTHER DOLLARS	71	63	73	75	82
TERMINAL HOURS	40	30	74	95	51

Figure 14. Other Resource Schedules Section of Project Schedules Report

Projected Resource Overruns

The resource usage schedules set up in the second section of the Project Schedules Report are used in the third section, Projected Resource Overruns. This section notes any overruns which may be predicted for computer dollars, file space, other dollars, or terminal hours. Computer dollars and other dollars are each summed up over time; the first period of time (if any) in which the budget is exceeded is noted for each of these resource types. For file space and terminal hours, the budget may be exceeded in any period of time; for these resources, all periods of time in which overruns occur are noted. For terminal hours, the "budget" for a period of time is calculated to be the number of terminals available times the number of hours in a period of time.

For example, if the budget for file space for a project is 400 blocks, there is one terminal available, and the budget for other dollars is $300, and if Figure 14 shows the resource schedules for this project, then Figure 15 below would be the Projected Resource Overruns Section.

SIMON REPORT III PAGE 3

PROJECT SCHEDULES AS OF 020275

PROJECTED PROBLEM AREAS

OVERRUNS OF RESOURCES -

FILE SPACE EXCEEDS BUDGET IN PERIODS STARTING 022575

OTHER DOLLARS EXCEED BUDGET IN PERIOD STARTING 022575

TERMINAL HOURS EXCEED BUDGET IN PERIODS STARTING 021175

Figure 15. Projected Resource Overruns Section of Project Schedules Report

Projected Scheduling Inconsistencies

The programmer schedules in Section 1 of this report were entered into the Simon data base by project managers. Programmers put in their estimates of the number of person hours which will be needed for each subsystem they are working on. Therefore, there exists for each subsystem in the data base two estimates of the number of person hours which will be needed for that subsystem, one estimate implied by the schedule and one entered directly by the programmers. These two estimates may not coincide; the fourth section of the Project Schedules Report notes all such scheduling inconsistencies.

For example, consider the project represented in Figure 13. If the total schedules for subsystems ABAD and CONO are given here, and if programmer CAPLAN estimated 250 hours for ABAD, and programmer MARDRID estimated 130 hours for subsystem CONO then the Scheduling Inconsistencies Section would be as in Figure 16.

PROJECT SCHEDULES AS OF 042175

SCHEDULING INCONSISTENCIES

SUBSYSTEM - ABAD

SCHEDULED PERSON-HOURS NEEDED - 200

ESTIMATED PERSON-HOURS NEEDED - 250

SUBSYSTEM - CONO

SCHEDULED PERSON-HOURS NEEDED - 150

ESTIMATED PERSON-HOURS NEEDED - 130

Figure 16. Projected Scheduling Inconsistencies Section of Project Schedules Report

Projected Scheduling Conflicts

Programmer and subsystem schedules depend not only on programmer availability and estimates for the number of person hours needed to complete a subsystem, they also depend on the calling and data sharing relationships between subsystems, that is, on the "Subsystem Interface Data." For example, if Subsystem A needs Subsystem B for testing, and a stub for B cannot be used, then Subsystem B must be completed before testing may begin on Subsystem A. If Subsystem A has been scheduled to have testing begun before Subsystem B is complete, a scheduling conflict results.

Part 5 of the Project Schedules Report is called Projected Scheduling Conflicts. It uses the schedules set up in Section 1 and the programmer-input Subsystem Interface (Interdependency) Data (see TRANSACTOR SPECIFICATIONS) to check for any conflicts such as the one described above.

For example, assume again that in Figure 13 the total schedules for subsystems ABAD and CONO are presented. If ABAD is declared to be needed for testing CONO, and a stub cannot be used, there is a scheduling conflict. Figure 17 below shows the Scheduling Conflicts Section noting this conflict (and others).

SIMON REPORT III PAGE 7

PROJECT SCHEDULES AS OF 042175

SCHEDULING CONFLICTS

SUBSYSTEM - CONO

SUBSYSTEM NEEDED FOR TESTING - ABAD

WHEN NEEDED - 040875

WHEN SCHEDULED TO BE COMPLETE - 052075

SUBSYSTEM - KAPLA

SUBSYSTEM NEEDED FOR TESTING - DENOL

WHEN NEEDED - 052075

WHEN SCHEDULED TO BE COMPLETE - 061175

SUBSYSTEM NEEDED FOR TESTING - CARRA

WHEN NEEDED - 052075

WHEN SCHEDULED TO BE COMPLETE - 060375

Figure 17. Projected Scheduling Conflicts Section of Project Schedules Report

SUBSYSTEM AND PROGRAM STATUS REPORT SPECIFICATIONS

Introduction

The fourth Simon report shows the current status of all subsystems in the system and of every program in each subsystem. It uses several types and sources of information to provide programmers and managers with a rounded view of a subsystem's or program's status. The report contains two sections; the first deals with subsystem status and the second with program status.

Subsystem Status

The Subsystem Status Section lists for each subsystem in the system a number of data entries which provide status information about the subsystem. The information is of several types and comes from various sources, as follows:

1. status information which illustrates how far along the subsystem is in its development; this data is either

   a. manually input by programmers or managers, or

   b. automatically collected by the Simon functions;

2. error information, manually collected, which provides the user with information concerning the subsystem's reliability and the nature of its errors;

3. complexity and length measures calculated by the Precompiler, measures which are believed to be correlated with subsystem reliability (incidence of errors) and comprehensibility.

Table IX shows for each data entry in the Subsystem Status Section which of the above sources of information supplies its data. Figure 18 shows an example of this section of the Subsystem and Program Status Report for the project "PROJECT-A" and the subsystem "SUBSYSTEM-A" which has had design completed but is still being tested.

Program Status

The Program Status Section uses information collected during precompilations and postcompilations to provide a view of the status of all programs in the system. As only information collected by the Pre- and Postcompiler is used, only information on a program's

Table IX

Data Entries in the Subsystem Status Section of the Subsystem and Program Status Report

Entry	Source of Data
Programmers	manual input
Date of Definition	manual input
Design - 1st assignment	manual input
Design - last assignment	manual input
Estimated Number of Programs	manual input
Current Number of Programs	automatically collected when programs are entered into data base after a precompilation or postcompilation
Current % cleanly compiled	automatically collected after post-compilation
Testing - 1st assignment	manual input
Testing - last assignment	manual input
Current % tested	manual input
Estimates specified	manual input
Actuals specified	manual input
Design/Test Interface Specified	manual input
Test Plan done	manual input
Documentation done	manual input
Errors charged	manual input

Table IX (Concl.)

Entry	Source of Data
Lines of Code	automatically collected upon precompilation
Number of Statements	
Halstead Length	
Complexity*	
Number of Compilations	automatically collected upon post-compilation

*The complexity measure has not yet been implemented.

```
SIMON REPORT #4 -- PROJECT-A SUBSYSTEM STATUS ON 04/25/75     PAGE 1A


SUBSYSTEM - SUBSYSTEM-A

PROGRAMMERS -                                          DESIGN/TEST
     MARLOWE                                             D     T

DATE OF DEFINITION - 03/16/75

DESIGN PHASE -
     FIRST ASSIGNMENT PERIOD - 03/20/75
     LAST ASSIGNMENT PERIOD - 04/08/75
     ESTIMATED NUMBER OF PROGRAMS -3
     CURRENT NUMBER OF PROGRAMS - 3
     CURRENT % CLEANLY COMPILED - 100%

TEST PHASE -
     FIRST ASSIGNMENT PERIOD - 04/22/75
     LAST ASSIGNMENT PERIOD - 06/03/75
     CURRENT % TESTED - 0%

ESTIMATES - SPECIFIED
ACTUALS - SPECIFIED
DESIGN/TEST INTERFACE - NOT SPECIFIED
TEST PLAN - NOT DONE
DOCUMENTATION - DONE

ERRORS CHARGED TO THIS SUBSYSTEM -
     COMPILER-DETECTED ERRORS - 17
     NON-COMPILER-DETECTED ERRORS -
          MEMORY -  2          LOGIC -  4
          MEMORY +  6          LOGIC +  0

TOTAL LINES OF CODE - 63
TOTAL NUMBER OF STATEMENTS - 55
TOTAL HALSTEAD LENGTH - 342
TOTAL COMPLEXITY - 0
TOTAL NUMBER OF COMPILATIONS - 14
```

Figure 18. Subsystem Status Section of Subsystem and Program Status Report

design status (as opposed to test status) is given. This information includes:

1. dates of first and last precompilations,
2. dates of first and last compilations.
3. number of compilations,
4. data measures calculated by the Precompiler, to wit:
    a. number of statements,
    b. lines of code,
    c. Halstead length,
    d. complexity.

For each subsystem in the system, the above information is presented for each of its programs which have been entered into the data base. A summary for that subsystem is then printed, giving the first and last precompilation and compilation dates for any program in the subsystem, the total number of compilations for the subsystem, and the total number of statements, lines of code, Halstead length and complexity for the subsystem. If no programs for this subsystem have been entered into the data base, i.e., no programs have been precompiled or compiled, a message is printed to that effect. This format is repeated for every subsystem in the data base.

Figure 19 below shows the Program Status for two subsystems. "SUBSYSTEM-A" has had programs entered into the data base, while "SUBSYSTEM-B" has had no programs entered.

SIMON REPORT #4 -- PROJECT-A PROGRAM STATUS ON 04/06/75 PAGE 1B

PROGRAM NAME	PRECOMPILATION FIRST	PRECOMPILATION LAST	COMPILATION FIRST	COMPILATION LAST	COUNT	STMT COUNT	LINE COUNT	HAL-STEAD
SUBSYSTEM - SUBSYSTEM-A								
AA	032175	032675	032175	032675	6	16	23	180
BB	032275	032175	032575	032875	10	48	69	215
ZZ	040175		040475		1	27	38	101
SUMMARY FOR SUBSYSTEM-A								
	032175	040175	032175	040475	17	91	130	496
SUBSYSTEM - SUBSYSTEM-B								
*******NO PROGRAMS HAVE BEEN COMPILED FOR THIS SUBSYSTEM*****								

Figure 19. Program Status Section of Subsystem and Program Status Report

ERRORS AND DISCREPANCIES REPORT SPECIFICATIONS

Introduction

The fifth Simon report deals with errors and discrepancies. An error is any single mistake (however often repeated) which causes a program to behave in a manner contradictory to its specifications. A discrepancy is any inconsistency found in the source code, operating system or compiler diagnostics, or test results which is not immediately diagnosed. It may or may not be found to be due to an error.

There are three parts to the Error and Discrepancy Report: Error and Discrepancy Summary, Discrepancy Summary, and Errors Reported Over Time.

Error and Discrepancy Summary

The first part of the report, Error and Discrepancy Summary, is based directly on the error and discrepancy forms used in the Transactor subsystem and is a summary of all such forms entered into the data base. First, a summary of the error forms is given, which gives for each entry on the error form a summation of its counts over all error forms entered into the data base. A similar summary of discrepancy information follows, based on all discrepancy forms entered into the data base. Finally, the first part of this report contains a printout of the descriptions of all outstanding discrepancies, i.e. all discrepancies which have not yet been diagnosed. These descriptions are also taken from the discrepancy forms that have been entered in the data base. A sample of the first part of the Error and Discrepancy Report is found in Figure 20.

Summary of Discrepancy Information

The second part of this report, Summary of Discrepancy Information, gives a projection of the project completion date, based on the error and discrepancy forms entered into the data base. The algorithm used is a least squares regression producing two lines, one plotting all discrepancies found against time and one plotting resolved discrepancies against time. The intersection of these two lines is then projected as the date on which there will be no outstanding discrepancies; this is an approximation of the project completion date. Two approximations are made, one using only discrepancies and one using both discrepancies and errors, considering errors as resolved discrepancies. Figure 21 shows an example of this section of the Error and Discrepancy Report.

Errors Reported Over Time

The final part of the Error and Discrepancy Report is called Errors Reported Over Time. This section gives a breakout of errors by subsystem over time. A row entitled "INTERFACE" is also included and shows interface errors over time. A sample of this section is given below in Figure 22.

SIMON REPORT V 042275

ERROR AND DISCREPANCY SUMMARY

ERRORS

SUMMARY OF ERROR INFORMATION -

HOW MANIFESTED -

DISCREPANCY FORMS	13
DESK CHECKING	9
CODE READING	13
COMPILER DIAGNOSTICS	58
OTHER SYSTEM DIAGNOSTICS	0
TEST RESULTS	16
OTHER	7

HOW DIAGNOSED -

OBVIOUS	69
LOGIC ANALYSIS	32
INSTRUMENTATION	6
OTHER	9

MENTAL LEVEL -

NOT PROGRAMMER	11
MOTOR	14
MEMORY -	50

Figure 20. Error and Discrepancy Summary Section of Errors and Discrepancies Report

MEMORY +	17
LOGIC -	19
LOGIC +	5
NUMBER OF OCCURRENCES -	
1 -	74
2 -	10
3 -	8
4 -	5
5 -	2
6 - 10	11
11 - 15	2
16 - 20	2
OVER 20 -	2
WHEN OCCURRED -	
ORIGINAL CODE	95
MAKING CHANGE	7
ADDING INSTRUMENTATION	4
CORRECTING ERROR	7
OTHER	3

Figure 20. Error and Discrepancy Summary Section of Errors and Discrepancies Report (Cont.)

SIMON REPORT V 042275

ERROR AND DISCREPANCY SUMMARY

DISCREPANCIES

SUMMARY OF DISCREPANCY INFORMATION -

HOW FOUND -

CODE READING	2
SYSTEM DIAGNOSTIC	6
TEST RESULTS	14
OTHER	2

DISPOSITION -

ERRORS	9
LAPSE IN COMMUNICATION	8
NOT DISCREPANCY	3
OTHER	1

SIMON REPORT V 042275

ERROR AND DISCREPANCY SUMMARY

DESCRIPTIONS OF OUTSTANDING DISCREPANCIES

021975	TRUNCATED ZEROS COMING OUT OF REPORT-2
030375	PRINTS RESOURCES HEADING WHEN SHOULDN'T
041175	DISAGREEMENT OVER NAMES OF TOKEN TYPES

Figure 20. Error and Discrepancy Summary Section of Errors and Discrepancies Report (Concl.)

SIMON REPORT V 042375

DISCREPANCY SUMMARY

NUMBER OF OUTSTANDING DISCREPANCIES	13
NUMBER OF RESOLVED DISCREPANCIES	21
TOTAL NUMBER OF DISCREPANCIES	34
PROJECTED CATCH-UP DATE	
1. USING DISCREPANCIES ONLY	03-21-77
2. USING DISCREPANCIES AND ERRORS (CONSIDERING ERRORS AS RESOLVED DISCREPANCIES)	03-10-77

Figure 21. Discrepancy Summary Section of Errors and Discrepancies Report

SIMON REPORT V 042275

ERRORS REPORTED OVER TIME

ATTRIBUTED TO	011475	012875	021175
AA	3	31	23
BB	2	5	23
CC			15
INTERFACE	1	11	2
ZZ		6	10

Figure 22. Errors Reported Over Time Section of Errors and Discrepancies Report

SECTION III

EXAMPLES OF USE*

SIMON'S GENERAL ROLE

An important aspect of any system is the human interface, i.e. how people interact with and use a system to accomplish goals. In order to illustrate this aspect, this section presents a few scenarios of possible uses of Simon by managers and programmers engaged in the everyday business of producing software. It must be borne in mind that Simon in no sense automates management or programming as such, but rather is a tool that collects information and presents it to the user in reports. The exact implications of the reported information are still very much left up to the users themselves, who must bring to the reports their own subjective knowledge of how a project is running. Thus, even though the cases presented below are typical of the smaller problems faced by programmers and managers in a project, their solutions will undoubtedly vary greatly from person to person and circumstance to circumstance.

GLOBAL ITEM CHANGES

The first example of a use of Simon deals with the relatively frequent task of changing a shared variable in a common. Suppose there is such a variable XYZ referenced by several programs that needs to be changed from a half word to a full word declaration. It is likely that the programmer responsible for the XYZ change will not know of all instances of its use and could therefore overlook the recompilation of some programs. However, by simply looking up the variable in the common items section of the System Structure Report, he would have an exhaustive listing of all affected programs and so be able to carry out the task more quickly and thoroughly, and moreover be able to gauge in advance the required amount of effort for a proposed change. The cross-reference listings for macros and files can obviously be used in a similar way.

SYSTEM STRUCTURE CONSIDERATIONS

Another feature of the System Structure report is the Subsystem Interaction matrix, of which Figure 10 is an example. This matrix affords an overview of the structure of a developing system.

*A more detailed paper on this topic by two of the authors (Fleischer and Spitler) is being prepared for separate publication.

Since it is produced automatically from the compiler output, it can be used to check the structure of the built system with that of its design. It can also guide in making up overlays or in planning changes to the system or major components. It can also be used to enforce system design disciplines; for example, if a subsystem is defined to correspond to a "level of abstraction," this matrix can be used to see where data sharing or unwarranted calls take place.

TROUBLE SPOTTING

To facilitate the understanding of this and the next case, we first present a scenario of an operating environment of a project using Simon: To initialize the project, the manager sets up the data base and puts into it the initial budget and data describing the programmers' schedules and subsystems. The programmers enter their initial estimates for their assigned subsystems. As the project progresses, actual expenditures (actuals) are put into the data base by the programmers on a regular basis, presumably at the end of a reporting period. Whenever necessary, each programmer also updates his estimates of resource expenditures required to complete his assigned subsystem. Also, error reports, interface summaries, discrepancy reports etc. are entered as they occur. The manager receives back the five reports at least once every reporting period. As needed, he makes updates to programmer assignments throughout the project.

Simon's reports can give several indications of potential problems that might be developing in a software project. As stated above, a programmer that realizes a subsystem will require more work than previously anticipated should make changes as needed to his previous estimates in the data base. Assuming that a manager will schedule as many programmer hours on a subsystem as was most recently estimated by the programmer, the updating of an estimate by a programmer will manifest itself as an exception in the Project Schedules report for the manager the next time the report is run. Figure 16 gives such an exception report. Of course, human failings, e.g. not to bother updating, and perennial optimism combire to make this source of potential warning data much less than adequate all by itself.

Other indications of problems can be found by comparing a programmer's schedule with the Program and Subsystem Status report. For example, if a subsystem should shortly be completed according to a programmer's schedule and yet only 20% of the modules have even had clean compilations as indicated on the Subsystem Status report, then the manager is alerted to do some investigating on the exact

status of the subsystem. In looking for other problems a manager could check:

the actual present expenditures on a subsystem versus the programmer's estimated expenditures;

the total project expenditures versus the project's budgeted expenditures;

the outstanding discrepancies (problems or unexpected results for which the cause hasn't yet been determined) that still exist;

projected completion dates that are computed automatically from discrepancy data;

complexity or length vs. that of comparable programs or subsystems or the manager's estimate of a reasonable value;

frequency of compilation;

errors and their distribution over time;

and many other similar indicators that, in conjunction with a certain amount of management expertise, should give a fairly accurate picture of the present status of a subsystem or project.

RESCHEDULING

If from this information, the manager decides to reschedule a subsystem, Simon performs consistency and completeness checks on the new schedule. The process of producing a consistent schedule is iterative. The manager prepares and enters updates to the schedule after which he runs the Project Schedules report. From the report, he receives exception reports on problems in the new schedule. Figures 15-17 give examples of the three exception reports. Accordingly he makes whatever necessary corrections to the schedule and again runs the report, etc. In a reasonable situation, this process should converge toward a complete, consistent, revised schedule for the project. In the process of working out this schedule, the manager has been able to see first hand and cope with the problems of coordinating a highly interdependent group effort. Simon provides an aid by handling much of the complex detail.

CHANGES IN SPECS

Another problem that SIMON helps solve is that of changing part of the specification of a system. A major part of the difficulty is often in evaluating the impact of the suggested change, as may be needed to judge the relative merits of the change vs. the status quo as well as to understand any other trade offs that might be made.

The evaluation often requires intimate knowledge of the system. In some cases, the information cuts across several people's work and wouldn't ordinarily be available from a single source. Under those circumstances, the global view of a project that Simon affords the manager can obviously be helpful.

If the change requires modifying existing subsystems and structures, the manager may profitably refer to the Project Structure information, such as the Subsystem Interaction Matrix in the System Structure Report to see what subsystem interactions will be affected, or the individual program interactions including calling and called programs, and global data and files references.

Depending upon whether the manager needs to add assignments or just change some old ones, Simon can help determine, as previously discussed, the scheduling ramifications of the proposed changes. Then, depending on how he evaluates the impact such a schedule will have on the project, he can decide whether or not the proposed change is acceptable.

SECTION IV

EXTENSIONS TO SIMON

COMPLEXITY MEASURE

One motivation for the design of Simon is the provision to the manager of early warnings of potential trouble in the system design and implementation process. Although Simon cannot check program code for correctness, it can report to the manager several kinds of statistics generated from the source code itself. The degree to which these statistics deviate from their expected values can give an indication of unusually complex or inefficient coding or of inadequate progress in coding.

Several statistics are generated from program source code by the prototype Simon: number of program lines, number of program statements, and a modified token count (the "Halstead" count [Halstead, 1972]). These measures are fairly simple and do not completely agree with intuitive notions of what attributes of a program affect its complexity or comprehensibility. Thus a new measure of psychological complexity was developed at MITRE [Sullivan, 1973, Bell and Sullivan, 1974]. This measure is derived from path counts after a partitioning of the program graph (logical flowchart) into minimal 1-in, 1-out subgraphs; it tends to be at a minimum for programs that are "structured" in the classical sense of having been generated by composition of the three "Dijkstra" control forms. Such a measure should give both the programmer and the manager a quick and more relevant indication of how complex, and thus how difficult to test and how prone to errors, a module is likely to be.

An experimental program to implement the complexity-measuring algorithm has already been written and tested in PL/1. The prototype Simon has been coded to facilitate the quick incorporation of a JOVIAL coded version of the algorithm. The JOVIAL preprocessor calls a stub (dummy) module that needs only to be replaced by the actual measurement algorithm. The data base maintenance program, the data base itself, and the Subsystem and Program Status report already provide and maintain a field for the complexity value. Thus the implementation of this complexity measure for Simon is essentially the re-coding of the measurement algorithm in JOVIAL.

TESTING-TOOL INTERFACE

The prototype Simon system is designed to monitor the entire program production process, but the prototype only provides truly automatic monitoring of the coding phase. This monitoring is done through the JOVIAL pre-processor and post-processor tools. A worthwhile extension to Simon would be the inclusion of a tool for automatic monitoring of the testing phase.

Such a testing tool has been developed for JOVIAL programs. The JOVIAL Automatic Verification System, or JAVS, provides code analysis, code instrumentation, testcase coverage analysis, and testcase generation guidance.* JAVS could be integrated with Simon in the sense that Simon could automatically record some of the outputs of JAVS and incorporate those outputs or derived measures into Simon's reports.

Simon's "Subsystem and Program Status" report could be augmented with fields that express the degree to which JAVS has been applied and the results of its applications. Simon could indicate whether a module had been entered into the JAVS data base, and whether the analysis step had been run for that module. An aggregate percentage of the modules so processed could be reported for the subsystem. Similarly, status information on other major milestones of JAVS use, such as whether the instrumented code has been generated, executed, or analyzed, could also be provided in that report.

Some of the most important statistics generated by JAVS could also be reported as part of the "Subsystem and Program Status." These might include: the percentage of decision-to-decision (D-D) paths covered, the number of instrumented tests, and the number of executions. The "complexity" computed by JAVS could also be reported, to supplement and compare with the abovementioned other measures of that same elusive quality.

Simon could also do other things with the information produced by JAVS. If historical values of such variables as "percentage of D-D path coverage" are kept, projections of time to completion could be made. It might also be possible to use JAVS-derived "reach sequence" information in conjunction with Simon's global data

*Due to a lack of documentation on JAVS at present, planning for integration with Simon is based on the available documentation for RXVP, the similar Fortran-based automatic verification system. JAVS and RXVP were developed by General Research Corporation.

reference information to produce information on data item interdependencies. Simon's error and discrepancy records could be automatically updated to reflect errors and warnings detected by JAVS' module static analysis step.

Simon could be interfaced with JAVS using the same method in which it interfaces to the JOVIAL (JOCIT) compiler. A JAVS "post-processor" program could scan the JAVS report output to extract the desired data. This program could be required to be run after certain JAVS activities. An alternate method of interfacing would be for Simon to directly access the JAVS data base, if all the required information is available there. Such an approach, however, would be very sensitive to internal changes in JAVS.

SCHEDULING AUTOMATION

The prototype Simon maintains comprehensive schedule information in its data base. The primary use made of this information is its display in the "Project Schedules" report and the projections of conflicts or overruns. The schedules themselves, however, could be derived automatically by an extension to Simon. The program estimates, testing dependencies, budget, and other resource-limiting data (e.g., number of terminals, specific personnel strengths or responsibilities) could be mechanically combined, using a PERT-charting algorithm, into a consistent development schedule. This schedule could then be displayed in a variety of ways, such as in tabular form (for printed output) or graphical form (using a plotter or CRT).

An automatic scheduling system could give further assistance to a manager in a number of ways. Several alternate schedules could be produced, for example, and the user could choose the "optimal" (by any criteria) one. The system could identify the "critical path" and describe the scheduling factors to which the schedule is most sensitive. Also, the automatic scheduler could assist the manager in an ongoing project in the task of revising schedules due to unexpected difficulties, budget changes, or personnel changes.

An automatic scheduling extension to Simon might be fairly easy to provide. Standard PERT packages are already available for use in major computer languages and systems. Simon could interface to such a package with the addition of a module to transfer information kept in the Simon data base to the format required by the PERT package.

AUTOMATIC COLLECTION OF ACCOUNTING DATA

The prototype Simon requires that a considerable volume of data on actual expenditures of resources (computer and personnel) be entered manually. This data is used in the "Estimates vs. Actuals" and "Project Schedules" reports, and forms part of the essential core of data needed for clear and current project visibility. Some of this data on "actuals" is directly derivable from information produced or maintained on the development computer facility, and thus it could be collected automatically in a basically straightforward manner. The data available in this fashion are program execution resources (and thus computer charges) and current file space requirements. Such automatic collection would not only reduce the level of clerical effort needed, but also ensure more accurate and timely data.

The Honeywell GCOS accounting statistics file [Honeywell 71] is the source of program execution data. The data stored on this file do not include the actual dollar costs, but rather the system resource usage from which the costs can be computed. One or more records is produced for each GCOS activity execution.

One difficulty in using this file is that GCOS activities are identified only by job number (SNUMB) and activity number. Thus a method is needed to establish the correspondence between subsystems and project phase (i.e., design/test) and the SNUMB and activity. One solution is to provide an extra activity to be executed immediately prior to all activities to be recorded by Simon. This extra activity could record both the subsystem and phase (provided by the user as an input) and the SNUMB and activity (obtained from GCOS) in a file to be used later by Simon. When Simon processes the accounting file, it will then be able to identify the relevant data records. Probably the only statistics Simon would produce from a scan of the accounting data would be the computer dollar charges (computed using the facility's charging algorithm) broken down by subsystem and phase.

Current file storage requirements can be obtained from the File System's CLIST function. This produces, among other data, the storage space occupied by each file. If an appropriate naming convention is devised and used, or separate subcatalogs are established, the correspondence between file names and subsystem, phase (i.e., design/test), and duration (i.e., permanent/temporary) could be determined by Simon.

INTERACTIVE QUERY CAPABILITY

The addition of an interactive query capability to the prototype Simon would allow the user to produce custom-designed reports derived from the most recent contents of the data base. For example, although Simon already produces two reports that list programmers within subsystem assigned to them, there is no report giving programmers followed by their subsystem assignments. Such a report could be added by the user of Simon if a suitable query system were available. An interactive query system could also be used to obtain current information from the data base between reporting periods. For example, a programmer might want to check the list of external references from a module to determine the effects of a recent change. Or a manager might want a most up-to-date list of outstanding errors (discrepancies).

An interactive query system for Simon could take one of two forms. One form would be a completely general query capability, able to produce a wide variety of reports and access any data in the data base in any fashion. Such a capability would allow the user great flexibility, but would probably be harder to learn and more cumbersome to use than a more specialized query system. The more specialized form of query capability would be one in which the user can only select from a limited set of preprogrammed reports or questions. Such a system could be customized highly to the Simon environment and would be essentially a fast-response extension to the current line of reports.

A generalized query system would be costly to build, but the IDS data management system used to implement the prototype system already provides an interactive query program. This program is quite general; it allows pre-programmed queries to be recalled for execution, and can even do simple accounting for reports. Thus an interactive query system for Simon could be developed quite readily using the IDS query system.

ON-LINE DATA INPUT

A User's reliance on Simon's reports requires the timely and correct input of changes and additions to manually-collected data. Assignments and estimates in the data base should reflect all recent changes, or Simon's listings and projections will not be dependable. Actual resource usage and error incidents should be recorded when they occur. Thus Simon's manual input functions should be as easy and foolproof as possible to use.

Currently, these functions are performed by the filling of forms that are later keypunched. One way to facilitate manual input is to provide an on-line data input capability. A programmer should be able to walk up to a terminal and be guided through the process of entering actuals and estimates. Likewise, a manager should be able to make changes to schedules or assignments interactively. In both cases, Simon should prompt the user for all required input. The user should not be forced to remember fixed input formats, and should be able to omit specifying items that are not required.

An interactive input feature can be added to Simon rather easily. For each transaction type, a time-sharing program can be written to prompt the user, accept information from the terminal, and format the data into a batch transaction. That transaction could then be either saved for subsequent background processing or immediately processed by Simon's transactor. A pre-prototype version of such an input program was written and used during the development of Simon to enter test data into the data base.

On-line, interactive data input would be most convenient if integrated with the query system discussed above. This would allow the user to examine the data base before updates are made, and thus could further contribute to convenience and accuracy.

GRAPHIC OUTPUT

Reports can often be made more effective through the graphical presentation of data. Much of the information presented in tabular form by the prototype Simon, such as schedules, testing interdependencies, and assignments, might be better presented through the use of bar graphs or network charts. The results of analyses done by Simon such as the extrapolation of error histories or the projections of overruns, could be augmented by line graphs which would give a better feel for the trends and the magnitudes of the projections.

Graphical output can be added to the prototype Simon by inserting calls to plotter (or CRT driver) subroutines into the reporting programs. In most cases the data required is already processed in producing the current reports.

HUMAN ENGINEERING

Under the term "human engineering," as used herein, is included the adjusting of system interfaces and procedures to fit better the actual environment of the users. This might involve, as

an example, the substitution of "weeks of full-time effort" for "person-hours" as the input form for estimates. Another example would be the reporting of original estimates alongside of current estimates in the "Estimates vs. Actuals" report.

It is really not possible at this point to detail the small changes that might be needed to make Simon into a completely comfortable tool for a given programming and project environment. Differences in terminology, conflicts of procedure, and difficulties in interpreting or correlating reports can only be determined through an examination of the actual use of Simon in a realistic environment. It should be assumed that, as with most other man-machine computer systems, some such problems will surely appear.

SENSITIVITY ANALYSIS

The prototype Simon is capable of making projections and recognizing certain conflicts based on the content of its data base. When the manager (or programmer) is contemplating changes to the data base, for example in assignments, schedules, or estimates, the Simon system should, on request, present the effects of those changes on system projections before those changes are made permanent. At present, if a subsequent report shows an entered change to be undesirable, the change must be undone by entering an inverse change. A procedure for making temporary or tentative data base updates would help ease the process of "experimenting" with Simon.

DESIGN LANGUAGE/ANALYZER INTERFACE

Simon is designed to monitor both the designing and testing of computer programs, and yet Simon's automatic involvement with design now begins at the fairly late stage of code compilation. Traditionally, program design has been an unstructured, ad hoc process. In recent years, however, specification and design languages have been developed to formalize the process of detailed software design. Language analyzers have also been designed to perform various kinds of verification, checking, and statistics gathering on the design expressed in the language.

An example of one such design language and analyzer is the User Requirements Language/User Requirements Analyzer (URL/URA) developed at the University of Michigan [Hershey, Teichroew et al, 1974]. This system allows a high-level system design to be expressed

in terms of inputs, outputs, and processing steps. The language analyzer checks the specification for consistency and completeness, and produces various reports concerning the system design. This design statement can then be used to guide the actual coding of the programs for the system.

Simon could be integrated with a system such as URL/URA. Such a coupling would allow earlier tracking of the design process, and would force an increase in the discipline of design over current ad hoc methods. Simon could keep track of the number of modules, files, inputs, and outputs that are defined. Simon could also collect and report on errors and inconsistencies observed by URL/URA. Later on in the project, Simon could compare the system-as-designed with the system-as-built, and report any discrepancies.

It is perhaps premature to specify just how such a system would be physically interfaced to Simon, but the discussion under TESTING-TOOL INTERFACE, above, suggests some likely approaches.

INTEGRATION OF PRECOMPILER, COMPILER, AND POSTCOMPILER

The prototype Simon uses both a pre-compilation pass and a post-compilation pass over the source coding to gather the information needed for its reports. All this data could be more efficiently gathered by the compiler (JOCIT) itself if suitable modifications were made to the compiler. This would eliminate the extra syntax scan required by the precompiler, and allow direct access to the symbol table created by the compiler. Not only would this reduce the cost of using Simon, but it would ease problems of coordinating changes among the tools, whether due to language specification changes or relatively minor improvement such as in output formats.

APPENDIX I

DATA BASE DESIGN

INTRODUCTION

The logical design of the current data base is presented here as a supplement to the specifications and as an aid to understanding the mechanization of the specified functions.

The Simon data base is a hierarchical data base. The major divisions are called records; there are five records:

1. the Project record, which contains data relevant to the project as a whole;

2. the Person record, which consists of programmer assignments; there is one Person record for every programmer (managers are not explicitly represented in the Simon data base);

3. the Subsystem record, one for each subsystem in the system;

4. the Module record, one for each module (i.e., single program, or unit of compilation) in the system;

5. the Error-disc-inf record, which contains all information relating to errors and discepancies.

Each of these records contains several data fields, each of which may or may not be divided further into fields, and so on. Any field which contains several other fields is called a "group item"; all other fields are "elementary items" or simply "items." A group item and all its sub-fields may be repeated several times within a record; such a group item is called a "repeating group." For example, each programmer will in general have several assignments; therefore, under the Person record the "assignments" field is a repeating group.

The logical data base design follows. Group items are easily noted as having several (indented) items listed beneath them. Repeating groups are noted as such. The data type of each elementary item is listed. Finally, such explanatory notes as are deemed necessary or helpful are included at the end of each record description.

PROJECT RECORD

A. project_name (character string)

B. start_date_gregorian (character string)

C. start_date_julian (number)

D. stop_date (character string)

E. period_of_time_length (number)

F. resource_allocation

1. number_of_terminals (number)

2. filespace (number)

G. funding_allocation

1. total_allocated

a. dollars

i. computer_dollars (number)

ii. other_dollars (number)

b. person_hours (number)

2. used_to_date

a. date (character string)

b. dollars

i. computer_dollars (number)

ii. other_dollars (number)

c. person_hours (number)

Notes:

i. stop_date = time the project must stop or, if no stop date, then Dec. 31, 1999.

ii. units of items:

a. *dollars for:* computer_dollars

other_dollars

b. days for period_of_time_length

c. blocks for filespace

PERSON RECORD

A. as_of_date (number)

B. person_name (character string)

C. assignments (repeating group)

1. subsystem_name (character string)

2. test_design_indicator (one character code)

3. period_of_time_designator (number)

4. total_hours_worked (number)

Notes:

i. period assignments cover past and projected; past are actuals, projected are estimates; whether a period is past or projected is determined using the as_of_date.

ii. period_of_time_designator contains the start date for the period of time of the assignment.

AD-A031 806 MITRE CORP BEDFORD MASS F/G 9/2
SPECIFICATIONS FOR SIMON, A SOFTWARE IMPLEMENTATION MONITOR.(U)
SEP 76 A E CORRIGAN, R J FLEISCHER F19628-76-C-0001
UNCLASSIFIED RADC-TR-76-288 NL

2 OF 2
AD
A031806

END

DATE
FILMED
12-76

SUBSYSTEM RECORD

A. subsystem_name (character string)

B. date_of_subsystem_definition (character string)

C. estimates

1. number_of_programs_included (number)
2. core_space (number)
3. person_hours_design (number)
4. person_hours_test (number)
5. terminal_hours_design (number)
6. terminal_hours_test (number)
7. computer_dollars_design (number)
8. computer_dollars_test (number)
9. other_dollars (number)
10. file_space_design
    a. temporary (number)
    b. permanent (number)
11. file space test
    a. temporary (number)
    b. permanent (number)

D. actuals (repeating group)

1. period_of_time_start_date (character string)
2. same as C.2
3. same as C.3

4. same as C.4

. .
. .
. .

12. same as C.12

E. subsystems_needed_to_complete_testing (repeating group)

1. subsystem_name (character string)

2. relationship (one character code)

3. required_status (one character code)

F. driver_feasible_flag (boolean)

G. design_done_flag (one character code)

H. test_done_flag (one character code)

I. test_plan_done_flag (one character code)

J. documentation_done_flag (one character code)

K. number_of_programs_included (number)

L. number_of_programs_compiled_with_no_errors (number)

M. number_of_programs_finished_with_testing (number)

N. programs_included (repeating group)

1. program_id (pointer)

O. total_errors_charged

1. compiler_detected_errors (number)

2. non_compiler_detected_errors (number)

1. memory_+ (number)

2. memory_- (number)

3. logic_+ (number)

4. logic_- (number)

Notes:

i. core_space: measured in units of 1024 words.

ii. relationship:
1 - called flag is 'CD'
2 - calling flag is 'CG'
3 - shares data flag is 'SD'

iii. required_status:
1 - totally complete flag is 'T'
2 - all stubs can be used flag is 'S'

iv. test_plan_done_flag:
1 - not present flag is 'N'
2 - complete flag is 'Y'

v. documentation_done_flag:
1 - not done flag is 'N'
2 - complete flag is 'Y'

MODULE RECORD

A. module_name (character string)

B. real_or_compool_flag (character)

C. subsystem_name (character string)

D. first_precompilation_date (number)

E. last_precompilation_date (number)

F. first_compilation_date (number)

G. last_compilation_date (number)

H. number_of_compilations (number)

I. Halstead_length (number)

J. C2_length (number)

K. lines_of_code (number)

L. number_of_statements (number)

M. modules_called (repeating group)

    1. program_name (character string)

N. commons_variables (repeating group)

    1. common_name (character string)

    2. variable_name (character string)

    3. type (one character code)

    4. set_used (one character code)

O. files (repeating group)

    1. file_name (character string)

    2. type (one character code)

P. defines_referenced (repeating group)

1. define_string_name (character string)

Q. first_clean_compile_date (number)

R. error_count_in_test_compile (number)

S. status_of_testing (one character code)

T. testing_complete_date (number)

U. errors_charged (number)

Notes:

i. status_of_testing: 1 - not_started flag is 'N'

2 - partially_complete flag is 'P'

3 - complete flag is 'T'

ERROR-DISC-INF RECORD

A. errors_system (number)

B. errors_operating_system (number)

C. how_manifested

1. classes 1 through 7 (number)

D. how_diagnosed

1. classes 1 through 4 (number)

E. mental_level

1. classes 1 through 6 (number)

F. number_of_occurrences

1. classes 1 through 9 (number)

G. when_occurred

1. classes 1 through 5 (number)

H. discrepancies_how_found

1. classes 1 through 4 (number)

I. disposition_of_discrepancies

1. classes 1 through 4 (number)

J. outstanding_discrepancies (repeating group)

1. discrepancy_ID (character string)

2. description (character string)

K. errors_and_discrepancies_by_period_of_time (repeating group)

1. pot_id (number)

2. errors_not_related_to_discrepancies (number)

3. discrepancies_reported (number)

4. discrepancies_resolved (number)

5. errors_by_subsystem_by_period_of_time (repeating group)

   a. subsystem_name_in_which_error_occurred (number)

   b. number_of_errors_found (number)

APPENDIX II

TRANSACTION DESCRIPTIONS

GENERAL NOTES

Any number of similar or dissimilar transactions may be included in the single input file to the Transaction Processor. In all cases, the listed inputs are those that are required for a single transaction. The output file from any run contains a job summary and any error messages relating to the input data.

DELETE MODULE

Input

1. the name of the module to be deleted

Function

1. deletes a module from the data base
2. updates module information in the containing subsystem

DELETE PROGRAMMER ASSIGNMENT

Input

1. name of the programmer whose assignment is being deleted
2. the assigned subsystem
3. whether or not the assignment is a design or test assignment
4. the start and stop dates that bound the range of the assignments

Function

1. deletes a range of assignments, all the same, from the data base

DELETE SUBSYSTEM

Input

1. the name of the subsystem to be deleted

Function

1. deletes a subsystem and all subordinate information

ENTER DISCREPANCY REPORT

Inputs

1. the date, time and submitter of this report
2. how this discrepancy was found (a number code)
3. brief description (text)

Function

1. enters the input data into the data base error summary records

ENTER ERROR REPORT

Inputs

1. the date, time and submitter of this report
2. how the error was manifested (a number code)
3. how the error was diagnosed (number code)
4. the mental level of the error (number code)
5. number of occurrences of the error (number code)
6. when the error occurred (number code)
7. when the error was manifested (number code)
8. the subsystem(s) and program(s) this error was charged to

Functions

1. enters the input data into the data base error summaries
2. increments the number and type of errors charged to the named subsystem(s) and the number of errors charged to the named program(s), and updates the data base accordingly

ENTER PREPROCESSOR RESULTS

Inputs

1. program and subsystem name for identification
2. real or compool flag for the program
3. number of severe errors, errors, and warnings from the precompiler
4. number of statements in the program
5. number of lines of code
6. Halstead length of the code
7. names of all DEFINE's

Functions

1. creates a module record in the data base if one does not already exist
2. updates all the quantities of the precompilation including first and last precompilation dates
3. increments the number of precompilations

ENTER POSTPROCESSOR RESULTS

Inputs

1. the program and subsystem names for identification
2. real or compool flag for the program

3. number of errors in the compilation

4. called programs

5. referenced files

6. referenced compool (common) variables

Functions

1. creates a module record in the data base if one doesn't exist

2. updates all quantities of a postcompilation including first and last postcompilation dates

3. increments the number of postcompilations

INITIALIZE PROJECT

Inputs

1. project name

2. start date of the project

3. stop date of the project

4. total allocated computer dollars

5. total allocated non-computer dollars

6. total allocated person hours

7. number of available terminals

8. total allocated file space

9. length of a period of time (POT),
where period of time is the length of time
which a Simon report will cover, e.g. 2 weeks

Functions

1. perform all necessary functions for initializing the data base in terms of file and record allocation and initialization

2. enter the data submitted by the user into the data base

3. zeros the following items and enters them into the data base:

   a. person hours used to date

   b. computer dollars used to date

   c. other dollars used to date

INITIALIZE SUBSYSTEM

Inputs

1. subsystem name

2. date of definition of the subsystem, i.e. when the specifications for that subsystem were first defined

Functions

1. sets up the data area for this subsystem

2. enters the input date into the data base

3. zeroes all items in the data base relating to this subsystem, other than the two input items

INITIALIZE/UPDATE ACTUALS

Inputs

1. subsystem name

2. start date for the period of time (POT) to which the data applies

3. actual resource usage for this subsystem for the stated POT, including the following:

   a. core space

   b. person hours for design

   c. person hours for testing

   d. terminal hours for design

   e. terminal hours for testing

   f. computer dollars for design

   g. computer dollars for testing

   h. any other dollars used

   i. file space used for design separated into temporary and permanent file space

   j. file space used for testing, separated into temporary and permanent file space

Functions

1. enters the input data into the data base

INITIALIZE/UPDATE ESTIMATES

Inputs

1. subsystem name

2. the following estimates for this subsystem:

   a. number of programs included

   b. core space

   c. person hours needed for design

   d. person hours needed for testing

   e. terminal hours needed for design

f. terminal hours needed for testing

g. computer dollars needed for design

h. computer dollars needed for testing

i. any other dollars needed

j. file space needed for the design phase, separated into temporary and permanent file space

k. file space needed for testing, separated into temporary and permanent file space

Function

1. updates the data base with the input data

INITIALIZE/UPDATE INTERFACE INFORMATION

Inputs

1. name of the subsystem which needs to be tested

2. driver feasibility

3. list of the subsystems which are needed for testing the above subsystem, giving:

   1. subsystem name

   2. whether this subsystem is to be added to the current list of needed subsystems, whether it is to be deleted from that list, or whether information is to be changed for that subsystem

   3. relationships to above subsystem (called, calling, or shares data)

   4. required status for testing (fully complete, partially complete, or stub can be used)

Function

1. updates the data base with the input data.

Note

If a needed subsystem in the input list is to be deleted from the current data base list, only the name of that subsystem and the "delete" indication need be specified.

INITIALIZE/UPDATE PROGRAMMER ASSIGNMENTS

Inputs

1. date as of which this information is current
2. programmer's name
3. any number of lists containing the following information:
   a. the start date and stop date for a consecutive group of periods of time (POTs).
   b. any number of lists containing the following information:
      1. name of a subsystem which the given programmer will work on or did work on during the given POT.
      2. whether the programmer will be doing or did do design or test on this subsystem during this POT.
      3. the total number of person hours worked or to be worked by this programmer on this subsystem on this phase (design or test) per POT represented by the start/stop dates.

Function

1. enters the input data into the data base.

Notes

1. This procedure allows a user to enter programmer work schedules. This includes both past and future schedules. If the user-supplied POT start date is a

past date, the information is assumed to be actual past schedules. Otherwise, the information is considered projected work schedules.

2. If the assignments for one programmer are the same over several POTs, then the user may enter one set of start/stop dates covering all these POTs and need only enter the assignments once. If only a start date is given for item 3a, it is assumed that only one POT is covered.

3. If a programmer is to work on both design and testing of one subsystem during one POT, these are considered 2 separate assignments and will be listed as such.

UPDATE DISCREPANCY REPORT

Inputs

1. disposition of the discrepancy, including
   a. date of disposition
   b. nature of disposition

Function

1. enters the data into the data base

UPDATE PROJECT INFORMATION

Inputs

1. project name
2. stop date of the project
3. total allocated computer dollars
4. total allocated non-computer dollars

5. total allocated person hours

6. number of available terminals

7. total allocated file space

Function

1. updates the data base with the input data

UPDATE SUBSYSTEM STATUS

Input

1. subsystem name and date as of which this information is current

2. status of the test plans for this subsystem (non-existent, or complete)

3. status of the documentation for this subsystem (non-existent, or complete)

4. list of the programs in this subsystem whose testing status has been changed, including:

   a. program name

   b. status of testing (not started, or complete)

Functions

1. updates the data base with the input data

2. calculates the following items and updates the data base accordingly:

   a. number of programs in this subsystem which have been completely tested.

   b. the "testing_complete_date" for all programs whose testing status has been declared changed.

   c. whether or not this subsystem is complete, i.e., all programs are tested.

APPENDIX III

TRANSACTION FORMATS

GENERAL RULES

The order and general format of the cards (or card images) for a transaction is as follows: The first card is the Transaction Header card. Following the header card is one or more data cards supplying the information necessary to complete the transaction. The data for each transaction has its own separate format, as will be described below. The format on the header card, however, is standard. Every such card has a "*START," a function field, and three optional fields. The first optional field is the date field which defaults to today's date if left blank; the second is the name field for identifying the author of the transaction; and the third is the "TRACE" "NOTRACE" field which determines whether the following data cards will be printed on the output report. If anything but "NOTRACE" is specified, this last field defaults to "TRACE" implying that the data cards are to be printed.

SPECIAL FORMATS

(*Means that those fields can be omitted, in which case the fields default to zero, blank, or "no update" depending on context.)

Abrev.	Card Type	Column Number	Field Length	Value
	Header Card			
		1	6	"*START"
		8	3	transaction function (listed in Table IV)
		12	6	date of transaction (MMDDYY); can be blank
		19*	35	Name of person responsible for transaction
		55*	7	"TRACE" or "NOTRACE"; default is "TRACE"

Abrev.	Card Type	Column Number	Field Length	Value
DMD	Delete Module Record	2	6	Name of module to be deleted (card can be repeated)
DPA	Delete Programmer Assignments			
		2	6	Reporting period start date of the assignment
		9*	6	Reporting period stop date of the assignment (blank implies only one period)
		16	12	Assigned subsystem
		29	1	Whether assignment is a test (T) or design (D) assignment
		31	35	Person's name
DSB	Delete Subsystem Record	2	12	Name of subsystem to be deleted (can be repeated)
EDC	Enter Discrepancy Record-1			
		2	13	Discrepancy identification
		16	1	How discrepancy found
	Enter Discrepancy Record-2			
		1	80	80 character description of the discrepancy
EER	Enter Error Report Record-1			
		2	13	Error identification
		16	1	How manifested
		17	1	How diagnosed
		18	1	Mental Level

Abrev.	Card Type	Column Number	Field Length	Value
		19	1	Number of occurrences
		20	1	When manifested
	Enter Error Report Record - 2			
		8	1	P or S according to whether the next field is a program or subsystem
		10	6 or 12	Program or Subsystem name
EPR	Enter Pre-processor Results Record-1			
		2	6	"MODULE"
		10	6	Program name
		20	12	Subsystem to which it belongs
		35	1	Real or compool flag ("R" or "C").
	Enter Preprocessor Results Record-2			
		2	4	"DATA"
		10	3	Number of severe errors from the Precompiler
		14	3	Number of errors
		18	3	Number of warnings
		24	6	Number of statements in the program
		33	6	Number of lines of code
		42	6	Halstead length of the code.

Abrev.	Card Type	Column Number	Field Length	Value
	Enter Preprocessor Results Record-3			
		2	6	"DEFINE"
		10	30	Name of a define
	(This card is used as many times as there are defines in the precompiled program.)			
EPS	Enter Postprocessor Results Record-1			
		2	6	"MODULE"
		10	6	Program name
		20	12	Subsystem name
		35	3	Number of errors in the compilation
		41	1	Real or Compool Flag ("R" or "C")
	(The following three cards can occur as often and in whatever order is needed.)			
	Enter Postprocessor Results Record-2			
		2	5	"CALLS"
		10	6	Called program's name
	Enter Postprocessor Results Record-3			
		2	6	"COMMON"
		10	6	Common variable's name
		45	6	Name of common

Abrev.	Card Type	Column Number	Field Length	Value
		52	2	Type of variable
		56	1	Set, used or both flag ("S," "U," or "B")
	Enter Postprocessor Results Record-4			
		2	4	"FILE"
		10	6	Name of file
		45	2	Type of file
IPR	Initialize Project Record			
		2	12	project name
		15	6	project start date (MMDDYY)
		22*	6	project stop date; default is last assignment
		29*	2	POT-length in days
		31*	2	Number of terminals
		33*	5	File space in llinks
		38*	10	Computer dollars
		48*	10	Other dollars
		58*	8	person-hours
ISB	Initialize Subsystem Record			
		2	12	Subsystem name
		15	6	Date of subsystem definition

Abrev.	Card Type	Column Number	Field Length	Value
IUA	Initialize or Update Actuals Record 1			
		2	13	Subsystem name
		15	6	POT for which this is actual
		21*	8	Core space
		29*	6	Person hours design
		35*	6	Person hours test
		41*	6	Terminal hours design
		47*	6	Terminal hours test
		53*	10	Computer dollars design
		63*	10	Computer dollars test
	(If the POT is past then this is an update and functions as in updating estimates. Otherwise, it results in creation of a new actuals record.)			
	Initialize or Update Actuals Record 2			
		8*	10	Other dollars
		18*	6	Temporary design file space
		24*	6	Permanent design file space
		30*	6	Temporary test file space
		36*	6	Permanent test file space
	(As in Initialize or Update Estimates, this card can be omitted or any of the fields can be blanks.)			

Abrev.	Card Type	Column Number	Field Length	Value
IUE	Initialize or Update Estimates			
		2	12	Subsystem name
		15*	6	Number of modules
		21*	8	Core space
		29*	6	Design hours
		35*	6	Test hours
		41*	6	Terminal hours design
		47*	6	Terminal hours test
		53*	10	Computer dollars design
		63*	10	Computer dollars test
	(Blank field means that the field remains unchanged.)			
	Initialize or Update Estimates - Record 2			
		8	10	Other dollars
		18*	6	Temporary design file space
		24*	6	Permanent design file space
		30*	6	Temporary test file space
		36*	66	Permanent file space
	(This record can be omitted when no file space updating is necessary. Again, blank fields mean no updating is to be performed.)			

Abrev.	Card Type	Column Number	Field Length	Value
IUI	Initialize or Update Interface Information Record 1			
		2	12	Subsystem name
		15*	1	Driver Feasibility Flag
	Initialize or Update Interface Information Record 2			
		8	12	Subsystem related to record 1 subsystem
		21	1	Action to be performed for this relationship: "A" = add this relationship "D" = delete this relationship "C" = change this relationship
	(if action is "C" or "A")			
		23*	2	Relationship of this subsystem to the above one: "CD" = called by above subsystem, "CG" = calls above subsystem, "SD" = shares data with above subsystem; default is no change
	(if action is "C" or "A")			
		26*	1	Required status of this subsystem for testing of the above subsystem: "T" = must be totally finished for testing,

Abrev.	Card Type	Column Number	Field Length	Value
				"S" = can use total or partial stub for testing; default is no change.
	(Record 2 can be repeated as often as needed for each occurrence of Record 1.)			
IUP	Initialize or Update Assignments Record 1			
		2	6	As-of date for this set of assignments
		9	35	Person being given assignment
	(There can be as many assignments for the above programmer as is needed, i.e. as many Record 2's as needed. If an assignment duplicates any in the data base - in name, subsystem, and designated flag - it is assumed that the number of hours in the assignment is being updated. The only update allowed is a change in the number of hours in an assignment. Deletion of an assignment is accomplished by the DPA function.)			
	Initialize or Update Assignments Record 2			
		8	6	POT-date for this assignment
		15*	6	Stop POT-date for a set of assignments. If blank then there is only one assignment
		22	12	Subsystem name
		34	1	Test-design flag ("T" or "D")
		36	4	Number of hours on this subsystem for this POT

Abrev.	Card Type	Column Number	Field Length	Value
UDC	Update Discrepancy Record			
		2	13	Discrepancy to be updated identification
		16	1	Disposition of discrepancy
UPR	Update Project Record (blanks imply no update)			
		2*	12	New project name
		15*	6	Project Stop date
		22*	10	Allocated computer dollars
		33*	10	Allocated other dollars
		44*	8	Allocated person hours
		53*	2	Allocated number of terminals
		56*	5	Allocated file space
USS	Update Subsystem Status Record 1			
		2	12	Subsystem name to be updated
		15	6	Date for which this info is current
		22*	1	Test-plan-done flag ("Y," "N," " "); default is no update.
		24*	1	Documentation done flag ("Y," "N," " "); default is no update
	Update Subsystem Status Record 2			
		8	6	Program within the above subsystem

Abrev.	Card Type	Column Number	Field Length	Value
		15	1	Status of Testing for this program: "N" = none performed, "P" = partially complete, "T" = totally complete

(There can be from 0 to as many Record 2's as are needed to specify the status of program testing within a subsystem.)

REFERENCES

David E. Bell and Joseph E. Sullivan, "Further Investigations into the Complexity of Software," The MITRE Corporation Technical Report MTR-2874, Vol. 2, June 1974.

Judith A. Clapp, "Monitoring Software Development for Reliability Indicators," Proc. EASCON Conference, 1974.

Judith A. Clapp and Joseph E. Sullivan, "SIMON: Finding the Answers to Software Development Problems," The MITRE Corporation, Technical Paper MTP-152, Bedford, Massachusetts, May 1974.

Maurice H. Halstead, "Natural Laws Controlling Algorithm Structure?" ACM SIGPLAN Notices 7, 2 (February 1972), 19-26.

E. A. Hershey, D. Teichroew et al., URL Language Reference Manual, U. of Michigan, Ann Arbor, July 1974.

Honeywell Corp., FORTRAN, Document No. BJ67, June 1971.

Honeywell Corp., Integrated Data Store, Document No. BR69, Rev. 1, December 1971.

Joseph E. Sullivan, "Measuring the Complexity of Computer Software," The MITRE Corporation Technical Report MTR-2648 Vol. 5, June 1973 (Reissued as RADC-TR-74-325, Vol. V, Jan. 1975 and as DDC document AD/A007770).