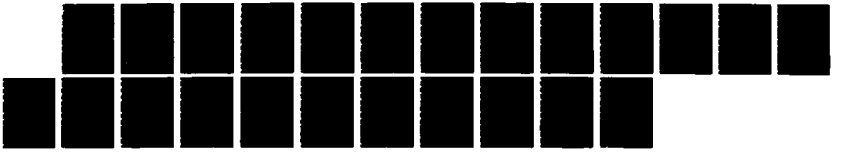
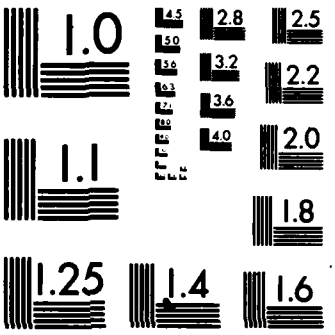
AD-A171 859 SOFTWARE DEVELOPMENT PRODUCTIVITY WITH MODEL - SHIP 1/1
STORE PILOT PROJECT(U) COMPUTER COMMAND AND CONTROL CO
PHILADELPHIA PA 1985 N00014-84-C-0282

UNCLASSIFIED F/G 13/3 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

COMPUTER COMMAND AND CONTROL COMPANY

2401 WALNUT STREET, SUITE 402 • PHILADELPHIA, PA 19103 • (215) 854-0555

AD-A171 059

SOFTWARE DEVELOPMENT PRODUCTIVITY WITH MODEL -SHIP STORE PILOT PROJECT

Prepared for Workshop
November 6, 1985
Held at George Washington University
Washington, D.C.
With Sponsorship of
The Office of Naval Research

DTIC FILE COPY

Contract N00014-84-C-0282

This document has been approved for public release and sale; its distribution is unlimited.

DTIC ELECTE AUG 1 4 1986

86 8 5 049

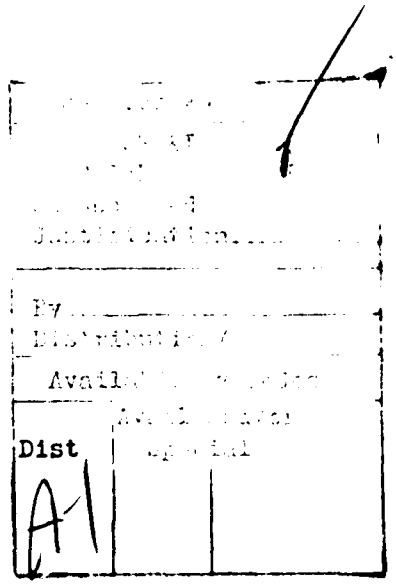
SOFTWARE DEVELOPMENT PRODUCTIVITY WITH MODEL --SHIP STORE PILOT PROJECT

Prepared for Workshop
November 6, 1985
Held at George Washington University
Washington, D.C.
With Sponsorship of
The Office of Naval Research

Table of Contents

1. OBJECTIVES AND SUMMARY 1
   1.1. Objectives 1
   1.2. Summary 1
2. THE SHIP STORE SYSTEM 2
   2.1. Functions Of The Ship Store System 4
   2.2. Background Of The Ship Store System 4
   2.3. Enhancements Of The System 5
3. MODEL SOFTWARE DEVELOPMENT METHODOLOGY 5
   3.1. Multi-Level, Top-Down Or Bottom Up Approach To Development 5
   3.2. Multi-Level Structure Of The Ship Store System 6
   3.3. Program Module Development 7
4. TRAINING OF FAADCPAC STAFF 7
5. SOFTWARE DEVELOPMENT PRODUCTIVITY 10
6. ERROR DETECTION AND CORRECTION AND CHANGES TO IMPROVE PROGRAMS 13
   6.1. Static Checking 17
   6.2. Dynamic Checking 17
   6.3. Highlights Of Error Analysis 18
7. CONCLUSIONS 18

QUALITY INSPECTED 1



1. OBJECTIVES AND SUMMARY

1.1. Objectives

The objective of the project reported here has been to provide a bridge across a wide gap that typically exists between research and the application of its results. In this case, the research concerns the MODEL system and the area of software development. Of special interest are the software development of applications characterized as complex, time-critical, medium to large scale, that utilize modern computer technology. The research on the MODEL system has been conducted at the University of Pennsylvania for almost a decade under partial sponsorship of the Office of Naval Research. The pilot project objectives are to verify, measure and evaluate the impact of the innovations in MODEL in real life environment.

The innovative features of MODEL will be discussed in detail in this report. They can be summarized by the following three main points.

1. *Automatic support for the entire software development process.* The MODEL system components perform documentation, checking and code generation, spanning the development of requirements, modularization, module specification, coding, static and dynamic debugging and performance evaluation.

2. *Use of Equational Language.* The user of MODEL defines the inherent concepts and rules of the problem through equations. The automatic system (a) aids in debugging through in-depth completeness and consistency checks and (b) transforms the statement of the problem into highly efficient programs.

3. *Application of on-line, concurrent/parallel processing technology and evaluation of performance to guide further tuning of efficiency.* The application is enhanced through use of operator/terminal interactions, on-line databases and interactions between concurrent processes. Performance evaluation is conducted automatically, reporting to the user program computation times.

The pilot project is the result of cooperation among three Navy agencies: The Office of Naval Research (ONR) - sponsor of the research, the Navy Comptroller (NAVCOMPT) - which has responsibility for Navy financial systems, and the Fleet Accounting and Disbursing Center, Pacific (FAADCPAC) - which hosted the project.

The Ship Store system has been selected as a highly suitable real-life environment to investigate the effectiveness of the MODEL system. The Ship Store system is used for financial and inventory management of large number of stores aboard Navy ships. It derives its complexity and size from the large number of diverse inputs, wide reporting responsibilities, and particularly the need for a man-machine interface to balance, reconcile and adjust many isolated but connected actions. The latter require complex man-machine interfaces, on-line databases and real-time communications among programs. There are 14 different inputs and 36 reports, in addition to large on-line databases. The project development produced automatically 38 programs with a total of almost 50,000 lines of PL/1 code.

1.2. Summary

Through implementation of the Ship Store System, the project verified the feasibility of using MODEL for large scale software development. More important are the measurements made and the evaluation of the effectiveness of the MODEL methodology and automatic systems. Three types of claims are made for MODEL.

1) *Tripling of Software Development Productivity:* Verifying improvements in productivity required measurement of manpower and costs of the development, from the initial conception of an application, including development of requirements, coding and testing, to the point where the developed system passed a strict acceptance test. This can be compared with similar measurements of conventional software development productivity reported

in technical publications. Widely reported statistics and models of conventional software development have been collected by Barry W. Boehm in "Software Engineering Economics", Prentice Hall, 1981. There are many factors that effect productivity. The evaluation focused on the number of statements and lines of code in a high level language (PL/1) produced per hour. Boehm reports in similar applications a range of 1-3 lines per man hour. This depends greatly on the complexity of the application and proficiency of the programmers. The project reported here is considered complex, particularly due to the interactive on-line aspects and the man-machine interfaces that aid the clerks in balancing, reconciling and adjusting transactions. The staff employed in the development had no prior data processing experience. Still an average of more than 10 lines of PI/1 code were produced per man hour.

2) *Use of MODEL by non-programmers:* A specification of a program in MODEL does not consist of a computer solution to a problem, but rather of a mathematical definition of the problem itself. It is the task of the MODEL compiler to automatically translate the specification of a problem into highly efficient computer programs that solve the specified problem. Therefore the user of MODEL is not directly concerned with the computer solution and need not have computer proficiency. In this project, FAADCPAC assigned to the Ship Store system development three persons who have not had prior exposure to, or experience in, developing computerized business data processing systems. Only one of these had learned previously a high level conventional computer programming language used on a personal computer (BASIC).

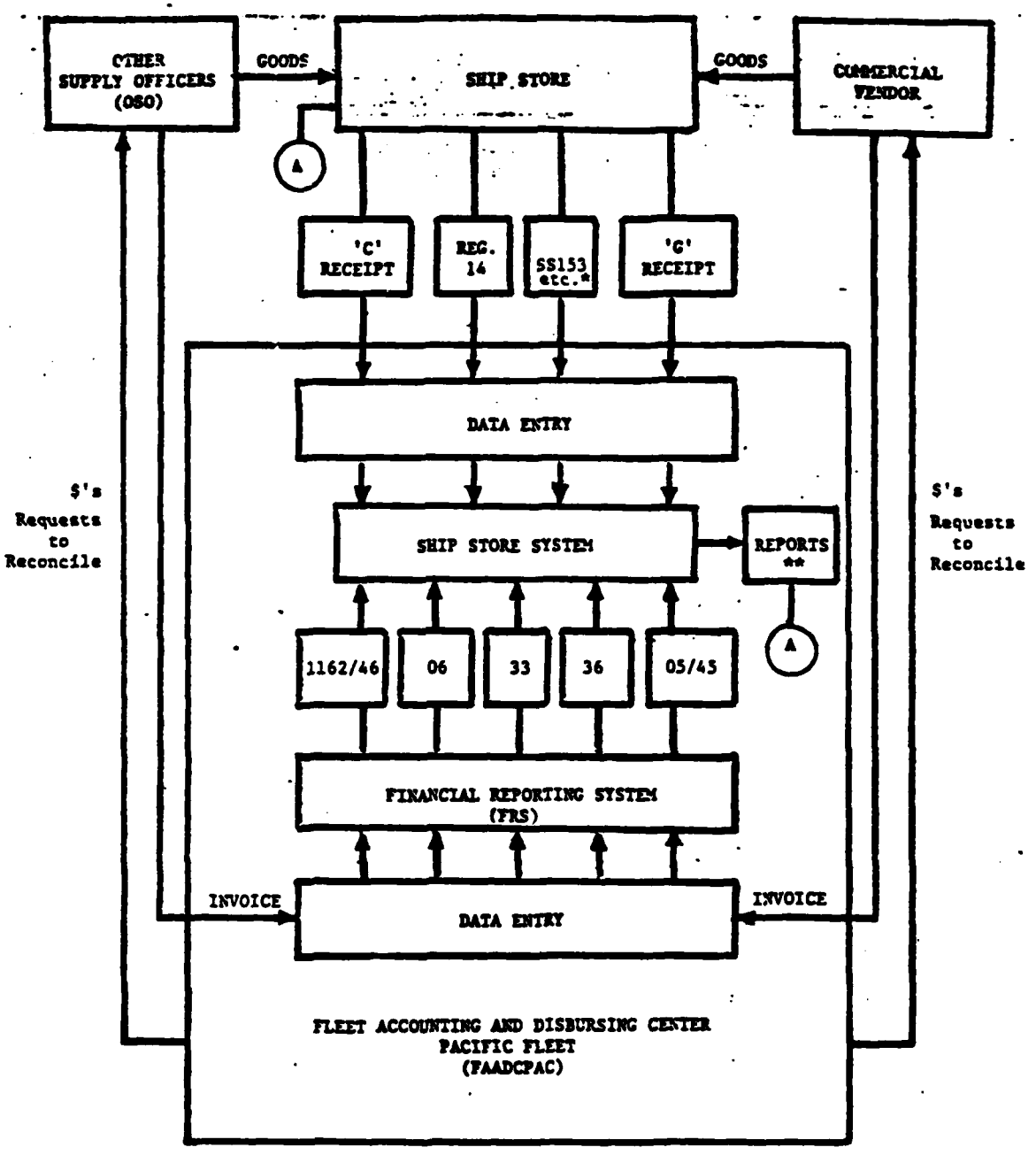
3) *Detecting high percentage of errors in static debugging:* The MODEL compiler has a much greater capability of detecting errors than conventional compilers. Detecting errors through the compiler and their correction is called *static* debugging. This is contrasted with *dynamic* debugging which consists of detecting errors or necessary changes by executing the program with real-life data and making appropriate changes. The latter is several fold more costly per error than the former. Conventional compilers typically detect less than 40% of the logical errors in a program. The MODEL compiler performs many additional checks and the collected statistics show that approximately 80% of the logical errors are detected by the compiler and corrected in static debugging.

The report is organized as follows. Section 2 gives a functional overview of the Ship Store system. The overall MODEL methodology, used in the implementation, is described in Section 3. Sections 4-6 deal with measurements and presentation of implementation statistics. The areas of special interest are: training of users - Section 4, software development productivity - Section 5, and error and change detection and correction - Section 6. Section 7 concludes the report.

2. THE SHIP STORE SYSTEM

The Ship Store system performs typical general ledger and inventory accounting functions. It performs these functions centrally, at FAADCPAC, for over 200 stores aboard Navy ships in the Pacific. Its complexity is derived from the great multiplicity of sources and destination of data and the need to balance and reconcile numerous actions by the community which it serves. Especially complex is the analysis and man-machine interface to help clerks find and resolve conflicts. As stated, the system is therefore also fairly large in scope.

The Ship Store system is described in this report on multiple levels of details. This section focuses on the top level, where the system is regarded as one box and it is defined functionally, through its inputs and outputs. Figure 2-1 illustrates also the overall environment surrounding the Ship Store system. The Ship Store system is represented as a central box in the figure, showing its inputs and outputs, their sources and destinations.



* 2074/2051, 27, 47, Profits, Contributions
** Reports are used by FAADCPAC Clerk, Ship Store and other agencies

Figure 2-1: Functional Description Of The Ship Store System

2.1. Functions Of The Ship Store System

The Ship Store System supports FAADCPAC financial managers and clerks in financial management and in reporting to the Naval Research Systems Office (NAVRESSO). FAADCPAC performs the following functions in conjunction with ship stores.

a. Audit
b. Maintain files, effect reconciliation of documents for invoices and receipts and balance accounts for financial statements
c. Effect reconciliation of cash reported in returns
d. Effect payment of dealers' bills
e. Consolidate financial statements

The Ship Store system performs automatically items b,c, and e and supports the other items.

As shown in Figure 2-1, input to the Ship Store System comes from two classes of sources:

Financial Reporting System (FRS) - including purchase invoices (Registers 05 and 45), transfer invoices (Register 1162 and 46), and adjustment, (Registers 33, 36 and 06).

Ship Stores - including purchase receipts ('G'), transfer receipts ('C'), deposit ticket (Register 14), and SS153 financial summaries.

These input data are available to the system on tapes in the form of card images. The input programs check the validity of the data before storing them in the database. Clerks use the system to resolve the more difficult errors in the input data.

Monthly reconciliations of receipts of merchandise and invoices are performed. Reports based on reconciliation of transactions show the matching status of every invoice or receipt in the system and the summaries of different categories. These reports provide also easy-to-reference information for the Ship Store clerks in researching and resolving the more complex discrepencies.

Tri-annually, financial statements are consolidated to produce balance sheet, profit and loss statements, and consolidated NAVCOMPT Forms 145 and 146.

2.2. Background Of The Ship Store System

FAADCPAC has had responsibility for financial management of approximately 200 ship stores since the 1960s. The original Ship Store System has been batch oriented for the IBM 360 computer, with a heavy emphasis on tape and card files. During 1980, the system was converted to the UNIVAC 1100 computer system for use in the NARDAC network.

Recently, the system, has been audited and referred to as "ineffective" and "incomplete". Unsubstantiated or duplicate payments to vendors have been a significant problem. This imposed the requirement on the new system to reconcile automatically invoices with receipts, and provide an effective interface to clerks conducting the necessary investigations. The input data had many inconsistencies which required incorporating in the new system automatic balancing of accounts and correcting of erroneous data.

2.3. Enhancements Of The System

Enhancements in the new system implemented by using MODEL address the problems outlined in the previous section. They include the following:

a. Interactive operation by clerks engaged in correcting inputs and balancing or reconciling conflicting data.

b. Terminal display - update - retrieve capability

c. Creation of database files that can be updated and perpetuated on a current basis (rather than on periodical basis).

d. Concurrent interactions with Ship Store clerks, processing massive input and updating of data

The benefits resulting from these enhancements are:

a. Capability to edit and correct inputs and balance and reconcile conflicts, resulting in timely handling of unsubstantiated or duplicate invoices.

b. Reduced clerical labor

c. Timely updating of databases (independently on reporting cycle)

d. Timely reporting

e. Ease of maintenance.

3. MODEL SOFTWARE DEVELOPMENT METHODOLOGY

3.1. Multi-Level, Top-Down Or Bottom Up Approach To Development

The MODEL system can support both the top-down and bottom-up approaches to application development. It also supports additions, changes and deletions. This capability is possible because of the nonprocedural and concurrent nature of MODEL. All changes are made in the MODEL language specification and their translation into program design and computer operations is performed automatically. The translation is invisible to the users and the user need not understand, or even know of, the complex computer technology that is involved.

Because of this flexibility it is not necessary to provide complete detailed requirements for the application prior to beginning individual program development. It is possible for instance to concentrate first on the more complex program modules, rapidly obtaining prototypes, and use the results to define the rest of the application. Thus, only a broad statement of objectives was available prior to initiating development of the Ship Store System, and detailed requirements evolved later.

The MODEL approach distinguishes between global and local aspects of the implemented system. There are separate components of MODEL for these two purposes. A *Configurator* component accepts as input a data-flow like diagram of the system, performs consistency checking and generates command language code for setting up communications between concurrently operating program modules and initiating respective programs. The Configurator can be used on each level of the top-down breakdown of the application. The *Compiler* accepts

equational specifications, performs the checking and generates a highly efficient program in PL/1. This is illustrated in the following for a top-down approach. A third component of MODEL, the Timer, which evaluates the processing times between inputs or output is oriented to real-time applications and has not been used in this project.

3.2. Multi-Level Structure Of The Ship Store System

The top-level view of the system was shown in Figure 2-1. The Ship Store system is shown as a single box at the center of the diagram, indicating its inputs and outputs.

The software development for the Ship Store System was then divided on the second level into two subsystems, for Requisition processing and Financial processing. The subsystems are similar to one another. Both of them have to validate input, perform reconciliation, generate reports, and allow the operator to modify the contents of their databases.

Based on the type of processing required, each of the two subsystems is further divided on a third level into three areas:

1. Front Office: provide interfaces for Ship Store clerks to perform editing and modification of transactions stored in the databases. Clerks requests are sent to the back office to be carried out via intermodule messages
2. Back Office: Respond to requests from the front office, access databases, and return the results to the front office.
3. Batch Process: All the non-interactive processing including input handling, balancing, reconciliation and report generation.

The implementation of the Ship Store System is thus decomposed in a top- down fashion as shown in Figure 3-1.

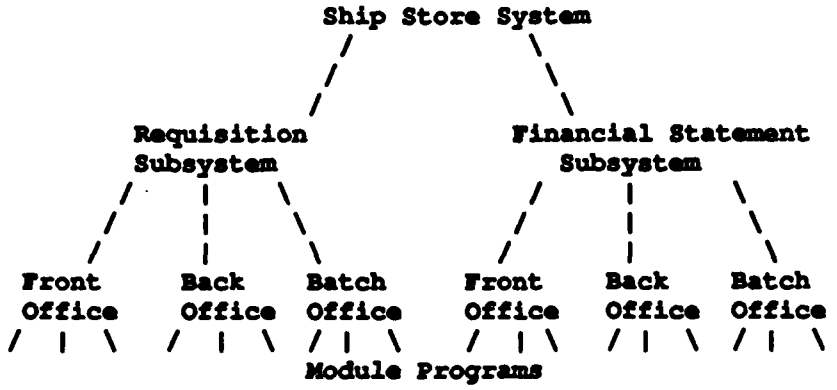


Figure 3-1 Multi-Level Breakdown Of The Shipstore System

The whole Ship Store System is decomposed on the fourth level to consist of 38 MODEL modules. The complicated module-to-module and module-to-file interfaces, especially for the front and the back offices, were established by using the MODEL Configurator.

3.3. Program Module Development

Most of the development time involved the use of the MODEL compiler for checking the detailed specification of each module and for generating the respective programs. This activity was carried out by the FAADCPAC staff, working independently, but consulting with the on-site CCCC staff, as they progressed.

Essentially, one person developed the front and back offices modules for both the Requisition and Financial subsystem. This part, considered the most complex and difficult, was developed in two steps. First providing a prototype and later finalizing the modules. The other two persons developed the Batch subsystems for the Requisition and Financial subsystems respectively. Each person typically worked on one module at a time, composing a detailed equational specification, debugging it statically, and then testing it with live data dynamically. Program modules were synthesized and tested as they were completed, to create progressively larger subsystems. This was done by use of the Configurator.

The Configuration of the batch and front and the back offices is shown for the Requisition and Financial Subsystems in Figure 3-2 and 3-3 respectively. It shows the communications between modules via files. It was specified in the MODEL Configuration Specification Language (CSL). The Configurator translated the statements in CSL into a series of command procedures and a set of communication mailboxes, which provided the participating modules communication facilities and ensured the synchronization of the intermodule message transfer. The Configurator took away communication interface concerns from the individual module developers, allowing them to treat communication messages as regular files.

4. TRAINING OF FAADCPAC STAFF

The three persons assigned by FAADCPAC to the Ship Store project received training in use of MODEL over the three months of September through November 1984. Previously to that they were trained by FAADCPAC in use of a terminal and in use of the EMACS editor to create and maintain files. They did not have prior experience in either programming or business data processing design. One person learned previously the use of the Basic programming language. The lack of previous training or experience by the assigned FAADCPAC staff proved beneficial to the objective of the pilot project as it required focusing on and analysis of the needs of training in MODEL.

Formal classes in the syntax and semantics of MODEL, including a refresher on arrays and equations, were conducted for one month (September 1984). This was followed by two months of more informal training on business data processing concepts, how to express these concepts in MODEL, how to debug specifications and how to test them with live data. The two months period also included learning the operation and design of the original Ship Store system. During this period the staff also identified the input data from the original Ship Store system that will be needed later for testing the new system.

Several of the more difficult training topics were identified in the training as follows:

a) Subscripting: variables in MODEL are typically envisaged as vectors, matrices, or generally multi-dimensional arrays. The elements of arrays must be referenced by providing with each variable a subscript expression.

b) Debugging: Inconsistencies are discovered by the MODEL compiler in dimensionality, ranges and data types of referenced arrays and by circular definitions.

c) Control variables: The parameters of data structures, especially the ranges of dimensions of variables can be defined dynamically, to depend as the input data.

d) Use of on-line data: This requires the user to reference data structures in index-sequential files.

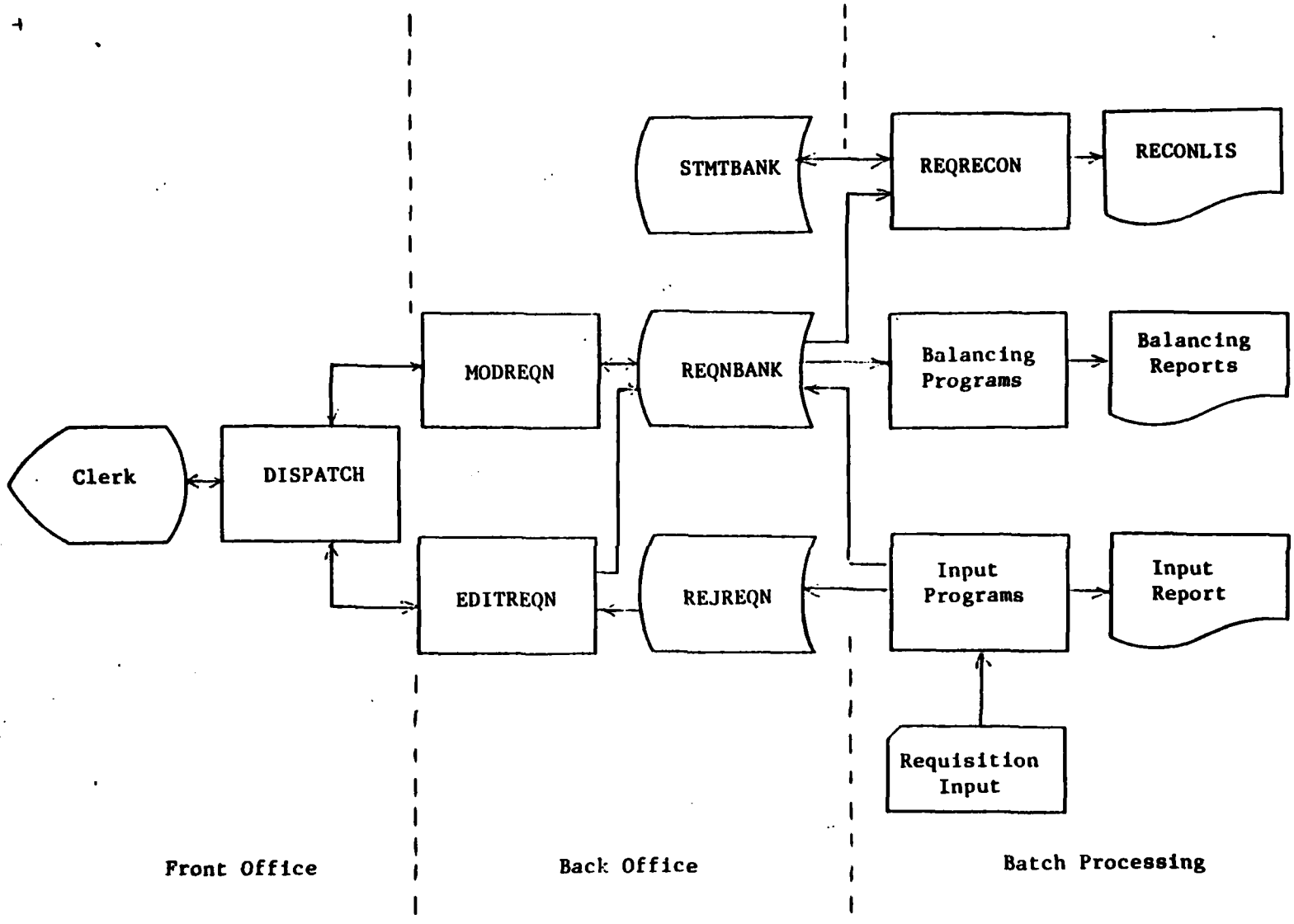


Figure 3-2: Configuration of Requisition Subsystem

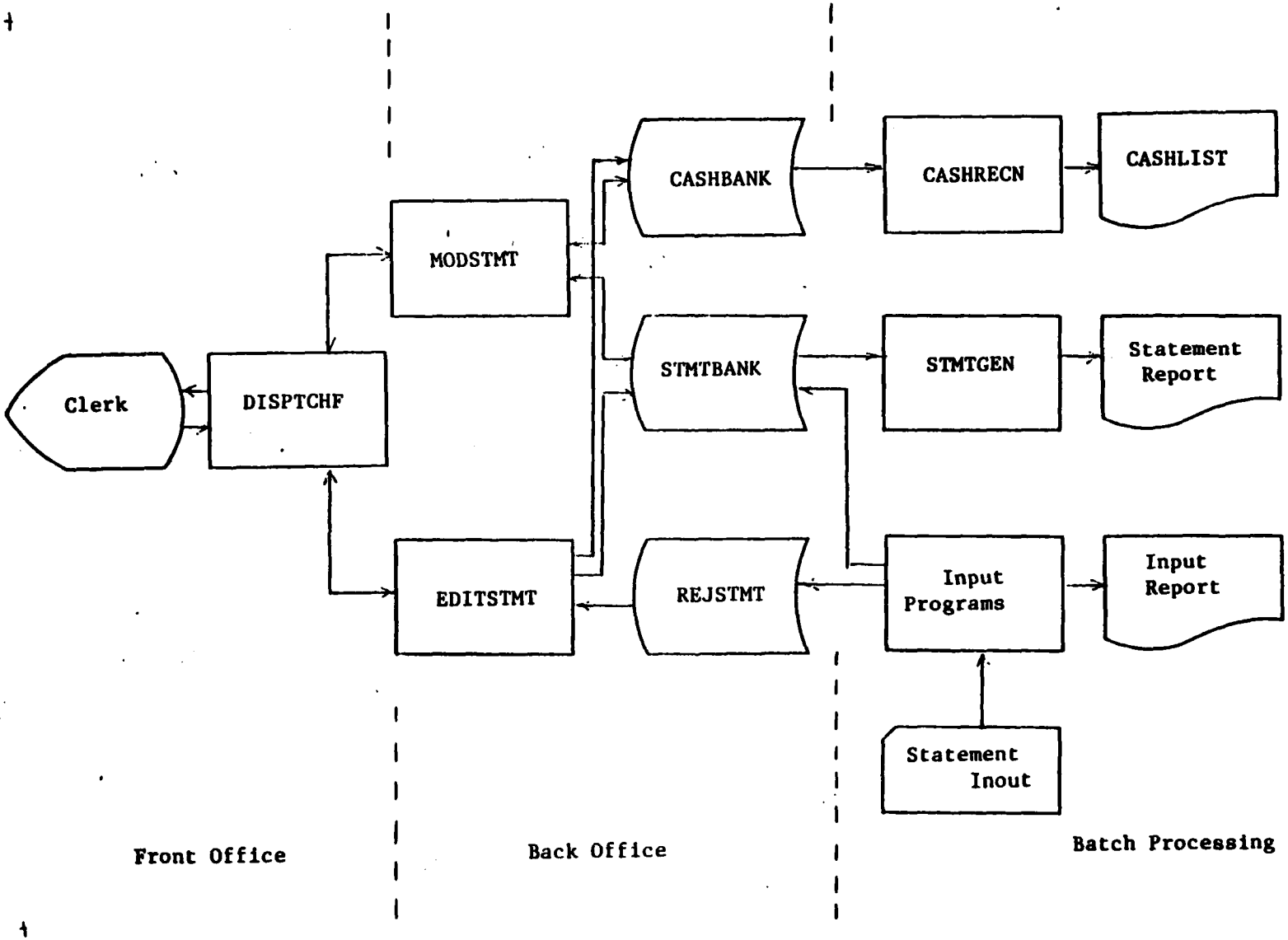


Figure 3-3: Configuration of Financial Statements Subsystem

The above concepts must be taught very carefully,

5. SOFTWARE DEVELOPMENT PRODUCTIVITY

As described in the previous section, the software development of the overall system was partitioned into Requisition and Financial Subsystems, each further divided into three areas: (a) batch, (b) front office for clerk interactions and (c) back offices for maintaining and updating on-line data. These areas were assigned to the three FAADCPAC members of the project for independent development.

The development started early in December 1984 after the above high level breakdown was established. Development area (a) progressed substantially from the beginning, while areas (b) and (c) required approximately three months for building a prototype interactive system before the final one was started. The input programs in area (a) were emphasized in the beginning as they processed input of data that was necessary to test the rest of the system. By the end of February 1985, all the modules for input of data were completed and a database containing actual Ship Store data was established.

A number of problems were discovered in the MODEL system as the development progressed. The problems were corrected on a continuous basis by the CCCC personnel in Philadelphia. The facilities of the MODEL system were enhanced throughout the project based on comments from the FAADCPAC staff.

The need for some changes to the initially conceived database structures became apparent during the development, requiring modifications to the previously completed modules. The nonprocedural aspects of the MODEL language had made this otherwise complicated job easier.

By the end of July 1985 all modules were coded and mostly tested with actual data. During the months of August and September, the February through May tri-annual data received by the original Ship Store system was loaded to form the databases for a thorough test of the new Ship Store system. Some problems due to the huge volume of data were discovered and corrected. Changes were also made for more flexibility and readability of the reports.

The development was concluded as of September 30, 1985. A total of 38 programs comprise the new Ship Store System. Figure 5-1 shows the numbers of statements and lines of each of the individual programs and their totals. Figure 5-2 shows the monthly production of programs over the ten month period. (Dec. 1984 through Sept. 1985)

From December 1984 through September 1985, a total of 4874 hours have been spent on the development. Consisting of 3816 hours by the three members of the staff of FAADCPAC assigned to the project and 1058 hours by CCCC staff on-site at FAADCPAC who consulted on the development. (Additional 526 hours were spent by the CCCC staff on site at FAADCPAC on maintenance of the MODEL system). Overall productivity in terms of MODEL and PL/1 statements or lines per man-hour are as follows:

```
MODEL statements per hour = 4635/4874 = 0.95
MODEL lines per hour = 21217/4874 = 4.35
PL/1 statements per hour = 30237/4874 = 6.20
PL/1 lines per hour = 48907/4874 = 10.03
```

As discussed in Section 1, this has been compared with productivity models and statistics for conventional methodology (as reported by Boehm). This supports the claim of tripling the productivity by use of MODEL over conventional high level programming languages.

PROGRAM NAME	MODEL Stmt	PL/1 Stmt	MODEL Lines	PL/1 Lines % of total	Hours % of Total	Productivity PL/1 lines/ per hour
A. Requisition Input						
REG05	174	1110	888	1825		
REG06	147	1079	666	1705		
REG1162	149	1074	687	1710		
REG33	171	1090	864	1796		
REG36	167	1086	840	1781		
REG45	138	1027	711	1655		
REG47	134	1019	696	1640		
REGGNC	136	870	719	1418		
TOTAL	1216	8355	6071	13530		
B. Requisition Balancing						
GNCBAL	67	416	341	607		
REGSORT	25	242	214	389		
REGBALP	90	694	481	976		
REGBALT	83	656	427	917		
TOTAL	265	2408	1463	2889		
C. Requisition Reconciliation						
EXCTMTC4	93	556	584	1142		
PREP30F4	52	339	229	549		
JDAT30F4	28	570	215	932		
RNBR30F4	28	570	214	932		
PREPRCN	24	183	73	251		
REQRECON	513	3289	1622	4618		
TOTAL	738	5507	2937	8424		
TOTAL Batch/Requisition Subsystem	2219	16270	10471	24843 50.6%	1272 26%	19.5

Figure 5-1(a) Program Development Statistics For Batch/ Requisition Subsystem.

Program Name	MODEL Stmts	PL/1 Stmts	MODEL Lines	PL/1 Lines % of Total	Hours % of Total	Productivity PL/1 lines per hour
D. Financial Statement Input						
NSS153	167	922	645	1718		
ANSUM447	116	817	409	1187		
CPROJPROF	155	988	686	1537		
APROJB10	120	813	401	1167		
CASHSALE	89	572	286	821		
ANCSALES	110	630	345	914		
TOTAL	757	4742	2772	7344		
E. Financial Statement Generation and Cash Reconciliation						
STMTGEN	249	949	1032	1749		
CASHRECN	64	506	263	725		
TOTAL	313	1455	1295	2474		
F. Auxiliary Modules						
BANKRCVR	48	460	113	654		
CLSEREQN	44	391	106	585		
CLSESTMT	101	850	320	1128		
CLSECASH	48	365	143	502		
TOTAL	241	2066	682	2869		
Total for Batch/Financial Subsystem and Auxiliary Program						
	1311	8263	4749	12687 26%	1742 35%	7.3

Figure 5-1(b) Program Development Statistics For Batch/ Financial and Auxiliary Programs

Program Name	MODEL Stmts	PL/1 Stmts	MODEL Lines	PL/1 Lines % of	Hours % of Total	Productivity Pl/1 lines per hour
G. Requisition Editing and Modification						
DISPATCH	218	776	1158	1368		
EDITREQN	181	1085	870	1875		
MODREQN	160	874	1411	1946		
GOODNITE	12	119	30	140		
	---	----	----	----		
TOTAL	571	2854	3469	5329		
H. Financial Statement Editing and Modification						
DISPTCHF	135	716	633	1199		
EDITSTMT	195	1314	822	2393		
MODSTMT	192	1101	1043	2316		
GDNITEF	12	119	30	140		
	---	----	----	----		
	534	3250	2528	6048		
Total for Front and Back Office subsystems						
	1105	6104	5997	11377	1860	6.1
			23%		37%	
GRAND TOTAL	4635	30237	21217	48907		

Figure 5-1 (c) Program Statistics For Front and Back Offices and Produced Pl/1 Programs

The area of Front and Back Offices for both Requisition and Financial Subsystems was the most difficult one, involving on-line data and interactive usage screens. The productivity in this area was therefore the lowest 6.1 PL/1 lines per hour.

The area of Financial/batch subsystem was next in difficulty, as it included financial statement generations and various calculations. The productivity in this area was 7.3 Pl/1 lines per hour.

Finally the Requisition subsystem batch area, had a number of similar inputs, which accelerated progress. It also however included the complex reconciliation of receipts and invoices. The productivity in this area was 19.5 PL/1 lines per hour.

6. ERROR DETECTION AND CORRECTION AND CHANGES TO IMPROVE PROGRAMS

Error detection and correction is probably the most intellectually demanding and costly activity in software development. Typically, it accounts for close to half of the total cost of development. Errors that are not detected during the development can affect reliability and confidence in the application. As noted, the MODEL compiler checks at considerable depths the global consistency of all the statements and locates conflicts as precisely as possible. One of the main advantages of the MODEL approach are the facilities for debugging. They make a major

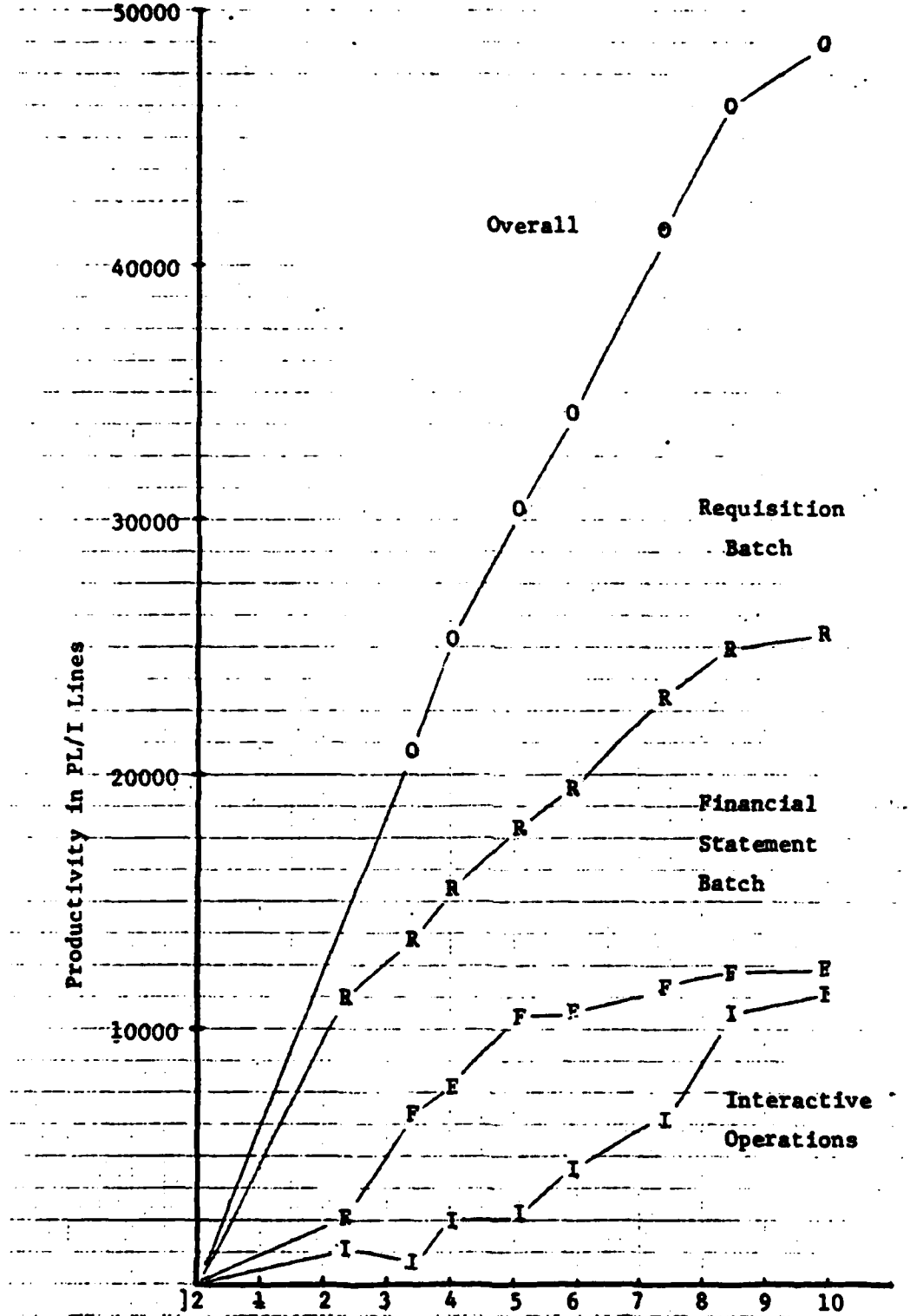


Figure 5-2. Monthly Progress In Program Development

contribution to the high productivity obtainable with MODEL.

One of the objectives of the project was to categorize the errors into a number of classes and analyze the respective difficulty and time to correct each class of errors.

Errors were monitored in several categories. The overall development of a module can be viewed as consisting of 3 parts - initial specification, static debugging and dynamic debugging. Note that subparts of each part can be carried out incrementally, and the user can alternate from one part to another. Error breakdown is first by static checking and dynamic checking. Static checking is for those syntax and semantic errors detectable by the MODEL Compiler. Dynamic checking consists of detecting incorrect outcomes when the compiled programs are executed. Static checking is by far less costly per error than dynamic checking. This section summarizes the occurrences of errors in these major categories and their subcategories.

Figure 6-1 summarizes the categories of errors found and changes made during the development, their frequencies in the respective categories, the average time required to make a correction in each category and the percentage of overall development manpower invested in correcting errors in each category. This information is important to locating areas in MODEL where additional enhancements would contribute the most. Figure 6-1 also gives an insight into the anatomy of software development with MODEL.

Category		Occurrences		Percentage of errors		Total Hours	Percentage of Hours	Hours to Correct
Initial Composition of a Specification		NA		NA		1103	22.6	NA
Static-syntax errors	(a) syntax	1668		43.5		350	7.2	0.21
Static logic errors	(b) Control variables	505		36.3				
	(c) Data type	450		32.4				
	(d) Subscripting	215	1390	15.5	36.3	1833	37.6	1.32
	(e) Incompleteness	136		9.8				
	(f) others	84		6.0				
Dynamic errors changes	(a) Report Layout	350		45.1				
	(b) Data conversion	112		14.4				
	(c) Calculations	98	776	12.6	20.2	1588	32.5	2.05
	(d) Input format	78		10.1				
	(e) File attribute	73		9.4				
	(f) others	65		8.4				
TOTAL		3834		100		4874	100%	

Static logic errors per MODEL stmt. = 0.30, per Pl/1 line = 0.028
Dynamic errors per MODEL stmt. = 0.17, per Pl/1 line = 0.016
Logic errors per MODEL stmt = 0.47, per PL/1 line = 0.044

Figure 6-1 Error and Changes Statistics

6.1. Static Checking

The error and warning messages produced by the MODEL processor for every compilation were saved on a disk file and served as a basis for the statistics in Figure 6-1. Following are the explanations of the subcategories of static checking shown in Figure 6-1.

Static Checking - Syntax Errors

a) Syntax errors occur most frequently and they account for more than half of the total number of errors. Fortunately this type of error is the most easy to correct. Usually they are simple omissions of semicolons or improperly nested IF-THEN-ELSE phrases. This, type of error will not be further considered in the evaluation of MODEL's error detection capability. The other types of errors involve logical inconsistencies or changes required for satisfactory operation.

Static Checking - Logical Errors and Changes

b) The next type of error involves the use of control variables. Control variables are used in MODEL to specify structural and organizational attributes of data. Improper use of control variables occurred more often at the earlier stages of the project, before their usage has become familiar to the development staff.

c) The next category are errors due to wrong specification of data types. Typically, this happened because arithmetic operators were used on character strings. During the period of development, the MODEL compiler was enhanced to better explain in error messages the causes of errors in this category. Therefore, in the future, the time to correct these errors will decrease.

d) The percentage of errors caused by missing subscripts or inconsistent uses of subscripts was about 15%, but these errors took longer to correct.

e) Errors caused by ommission of variable definitions are trivial to correct because an error message pinpoints exactly the undefined variable.

6.2. Dynamic Checking

Dynamic checking detected not only logical errors but also deficiencies in the operation of the program to meet the objectives of the application. Dynamic errors were recorded daily manually and served as a basis for the statistics in Figure 6-1. Every time a program (or a set of programs in debugging interactive operations) was executed and improperly terminated, the error was analyzed and its type recorded. More than 50% of the errors detected by dynamic checking occurred while debugging the interactive operations. Following is an explanation of the subcategories of dynamic checking shown in Figure 6-1.

a) The most frequent and easy to correct were the errors or inadequacies in specifying the data formats and report layouts. About half of the dynamic errors (45.1%) were of this type.

b) Data conversion errors (14.4%) usually were the results of omitting conditional expressions in equations. This resulted in undefined variables in some instances. These errors were relatively hard to trace.

c) Errors from inaccurately specified calculations resulted mostly from wrong initial or end conditions for calculating totals or subtotals.

d) Errors in input formats were the easiest to detect and correct.

e) Errors in file attributes consisted of erroneous organization of files, or ranges of dimensions, causing references to elements outside the range.

f) Dynamic errors other than the ones mentioned above were very much system-dependent. Since the VAX VMS file system is quite flexible, logical file name assignments are sometimes necessary if files other than the defaulted ones are to be used in testing. File-not-found or inconsistent attribute errors occurred when the logical name assignments before program execution were omitted.

6.3. Highlights Of Error Analysis

The statistics on errors and changes during the Ship Store application software development are shown in Figure 6-1, together with their summaries. The overall software development is divided into three parts-initial composition of specification (22.6% of overall manpower), static checking (44.8% of overall manpower, divided into 7.2% for syntax errors and 37.6% for logical errors) and dynamic checking (32.5%) that include not only individual module testing but also acceptance testing of the entire system.

Figure 6-1 shows also a further breakdown of the static/logical and dynamic checking into error categories. Subscripting errors in static checking are the most difficult to correct. The MODEL system has been enhanced by an additional report and more precise pointing to the offending statement to reduce the difficulty in correcting this type of error.

In dynamic checking, errors in calculation were the most severe to correct. A number of functions have been added to the MODEL language to be used for summing, averaging and searching, over vector variables, from where many of the calculation errors originate.

The errors in the subcategories denoted as "others" are few in number but severe. In static checking they include detection of circular logic. In dynamic checking they include errors in the configuration.

64% of the logical error and changes were detected by static checking and 36% by dynamic checking. The latter includes as a large portion changes made to enhance the Ship Store application based on rapidly obtaining prototypes of programs and testing them with data. These are therefore not errors in logic but improvements.

Finally the statistics show a very high error rate per MODEL statement (.45, namely one logical error or change per 2 MODEL statements. This amounts to one error or change per 23 PL/1 lines of produced code). This is because of the many changes made as a user progressively increases the size of a module and simultaneously determines the requirements of the module while composing a module specification. It also reflects the effect of a learning curve-with the staff initially making more errors, which receded subsequently to a constant level after the third months of the project.

7. CONCLUSIONS

The objectives of the pilot project were achieved as follows:

a) *Feasibility*: Use of MODEL in a fairly large project involving both diverse data and complex computation was demonstrated.

b) *Training*: We demonstrated that personnel without prior knowledge of data processing or computer programming can be effectively trained to use MODEL in software development of large applications.

c) *Productivity*: Tripling of software development productivity was demonstrated. This agrees with much of the

other experience in use of MODEL, prior to this project, and concurrently with it, by users of MODEL elsewhere.

d) *Error and Change Analysis*: Statistics were collected on categories of errors and changes, giving an insight into the role of automatic checking of consistency and completeness by the MODEL compiler.

Much has been learned on where and how to enhance the MODEL system. Of greatest importance were the lessons learned about needed thoroughness and completeness of training and additional automatic aids in debugging. Enhancements were made to the MODEL system incorporating what was learned from this project.

Another important lesson learned is the power of the concurrent/ parallel processing techniques built in MODEL. The project demonstrated how business data processing can be greatly simplified. Use of concurrent programs that communicate one with the others resulted in much smaller program modules than if stand alone programs that communicate only through data files had to be developed. Also the concurrent processing, together with the nonprocedurality of MODEL enabled much simplified modularity scheme and ease in making changes.

The development of the Ship Store system utilized the VAX family of Digital's computers under the VMS operating system. Another version of MODEL operates on IBM main frames under VM/CMS or MVS operating systems. The MODEL compiler operates on relatively small computers as well, on Digital's Microvax and IBM's PC-AT 370. These computers can also operate in a distributed processing network. This makes it feasible also to operate the Ship Store system in such a network. For example a small computer can be placed in each Ship Store to make corrections to the data at the source.

END

DTIC

9-86