System quality through structured programming

by F. T. BAKER

IBM Corporation
Gaithersburg, Maryland

INTRODUCTION

Experience in development and maintenance of large computer-based systems for government and industry has led the IBM Federal Systems Division to the formulation of a new approach to production programming. This approach, which couples a new kind of programming organization (a Chief Programmer Team) with formal tools for using structured programming in system development,[1] was recently applied on a contract with The New York Times for an online information system. Compared to experience on similar contracts in the past, the approach resulted in increased programmer productivity coupled with improved quality. An earlier paper[2] describes the approach in detail and gives productivity measures in a form which should allow comparability to other systems. Following a brief description of the system and a review of the approach, this paper discusses the quality of the system as observed during a thorough acceptance test and in the initial period of operation following its delivery.

THE INFORMATION BANK SYSTEM AND ITS DEVELOPMENT

The New York Times Information Bank is an on-line system which will eventually replace the clipping file (morgue) now used by the Times to provide background information for articles being written. An inquirer may interact with the on-line system to select index terms, specify document parameters (e.g., date of publication, section of the paper), and view article abstracts until he has identified those articles relevant to his immediate needs. Reporters and editors at the Times do this by means of an IBM 4506 Digital TV display unit which can display either text transmitted from the IBM System/360 Model 40 Central Processing Unit or images from standard TV cameras, and transmit text to the System/360 from a standard keyboard. They may also view the full original articles, which are stored in a microfiche retrieval device containing TV cameras capable of being switched to the IBM 4506's under System/360 control.

While editorial support is the main purpose of the system, a number of other features are provided. In addition to the 40 terminals mentioned above, another 24 IBM 4506 units without article viewing capability are interfaced to the on-line system for use by indexers keying index terms, abstracts and document parameters for eventual entry into the system files. The Times is marketing the retrieval service, and up to 500 remote terminals may be added to the system. (While remote users cannot view the articles on their terminals, they can view the abstracts, which provide information sufficient to permit retrieval of the articles from back issue files or from microfilm.) The on-line system (the "Conversational Subsystem") is supported by other subsystems which provide the security data and interactive message texts used by it, edit the keyed indexing data, maintain the system files, print abstracts and clipping references so that users may receive hard copy, log all major interactions with the system and maintain and print statistics on its use. All programs operate on the 360/40 under control of the Disk Operating System.

The system was developed by a Chief Programmer Team, a functional programming organization similar in concept to a surgical team. Members of the team are specialists who assist the Chief Programmer in developing a program system, much as nurses, anesthesiologists and laboratory personnel assist a surgeon in performing an operation. A team is organized around a nucleus of a Chief Programmer, a Backup Programmer and a Programming Librarian. The Chief Programmer is both the prime architect and the key coder of the system. The Backup Programmer works closely with the Chief to design and produce the system's key elements, as well as providing essential insurance that development can continue should the Chief leave the project. The

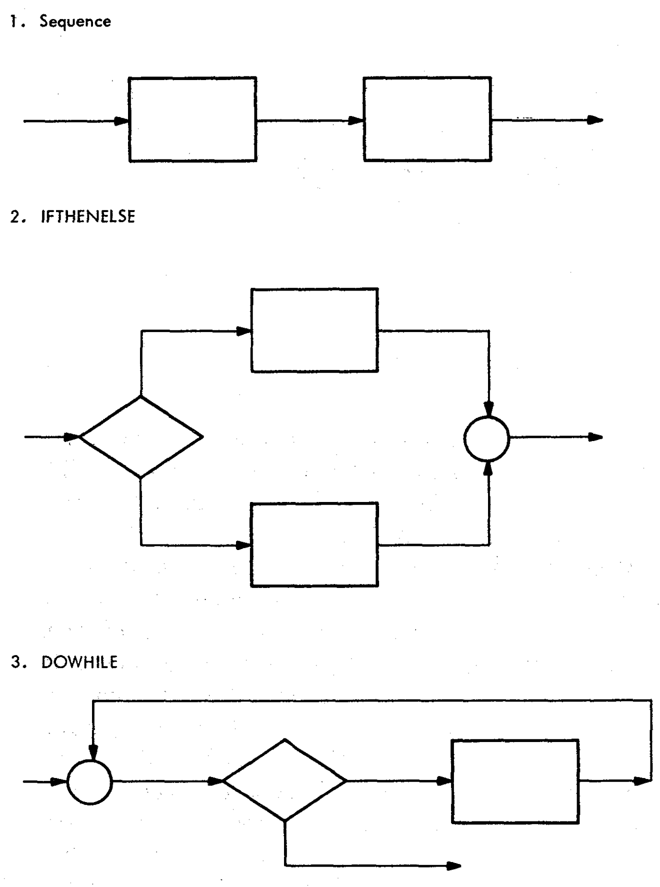

Figure 1—Progressions allowed in structured programming

Programming Librarian is responsible for maintenance and operation of a program library system used to keep all system programs and data both internally in machineable form and externally in well-organized, highly readable form. This Team nucleus, usually assisted by a systems analyst, designs and begins development of the system. The Team is then augmented by additional programmers who produce the remainder of the code under the close supervision of the Chief and Backup Programmers. "Egoless programming,"[3] featuring careful code review by team members other than the original programmer, is practiced throughout.

In addition to the functional organization and the enhanced cooperation fostered by the program library system, the Team operates in a highly disciplined fashion using principles of structured programming described by Dijkstra[4] and formalized by Mills.[5,6] These couple a top-down, evolutionary approach to systems development with the application of formal rules governing control flow within modules. In the top-down approach a nucleus of control code is written and debugged first. Function code is then written incrementally and added to the already operational system. This approach eliminates the need for throwaway drivers and reduces integration problems typically encountered at the end of a project. It also improves reliability because code is debugged within the actual system and because major portions of the system, including critical control code, are operational during almost the entire development period. The rules governing control flow are a consequence of a program structure theorem proved by Böhm and Jacopini.[7] This states that any proper program—a program with one entry and one exit—can be written using only the programming progressions illustrated in Figure 1.

Application of these rules permits a program to be read from beginning to end with no control jumps. It therefore simplifies testing and greatly enhances the visibility and understandability of programs. Finally, it supports the writing of program modules in top-down fashion by enhancing the ability to write and debug control code before adding function code.

DEBUGGING EXPERIENCE

Throughout the development of the system, progress was noticeably enhanced due to the use of structured programming and the library Although no statistics on number of errors or number of runs per module were kept, it was apparent from a qualitative standpoint that both were significantly reduced when compared to similar systems on which team members had previously worked. In a number of cases, program nuclei consisting of two to four hundred source statements ran correctly the first time. In all cases, debugging was clearly faster. Identification of paths to be tested was greatly facilitated by the use of only those formalized control structures permitted by our structured programming conventions.

ACCEPTANCE TEST EXPERIENCE

The system was developed in two major steps. To allow the Times to prepare data for the system files and for debugging and testing of the on-line system, the File Maintenance Subsystem was developed first. Following delivery of that subsystem, the Conversational Subsystem and the rest of the supporting subsystems were developed.

Rigorous and extensive formal testing was performed prior to acceptance by the Times of each of these major phases of the system. For each phase, a test plan was developed jointly by IBM and the Times. Each plan was designed specifically to test all functions included in that phase and was derived principally from the detailed functional specifications agreed upon by the two parties. Data to test these functions were then prepared exclusively by the Times, and these acceptance tests were conducted by the Times with IBM personnel in attendance. All the functional tests were rerun after all problems identified had been corrected, so that corrections could not have undetected effects on parts of the system already tested.

The File Maintenance Subsystem contained 12,029 lines of source code (about 14 percent of the overall system). The test plan for it contained tests for 171 separate functions required in creating and maintaining system files. Acceptance testing lasted a week and also covered all operational aspects of the subsystem, including the elaborate backup and recovery procedures incorporated to ensure preservation of the valuable data. Listings and hexadecimal dumps of all files were made and checked to ensure compliance with predicted file content and specified formats. No errors at all were detected during any of the testing of the File Maintenance Subsystem.

Acceptance testing of the Conversational Subsystem, which contained 38,990 lines of source code (about 47 percent of the overall system), was carried out during a five-week period. The first two weeks were devoted to single-thread (one user signed on from an IBM 4506) testing of the 286 separate functions itemized in the Test Plan. Seventeen errors were detected during this testing, all in the interaction processing modules and none in the time-sharing control program.

Following the single-thread testing, multiple-thread testing was conducted. All the previous tests were repeated with multiple users executing them asynchronously from IBM 4506's. No additional errors were discovered during this testing. Finally, the tests were repeated a third time from IBM 2740 and IBM 2265 terminals, serving to test the remote terminal handling features of the system. While all function was verified to be identical to that observed using the IBM 4506's, three errors were detected in the control program. These all had to do with handling of unusual types of transmission errors on remote lines.

Finally, "Free-form" testing allowed for several periods of retrievals by typical users under conditions when any errors or anomalies detected could be carefully recorded and analyzed. No errors were detected during this type of testing. In addition to the formal acceptance testing, system performance was measured to compare normal and peak load performance to a set of performance goals specified by the Times. Even though the system was operating on an IBM 360/40 with three disk drives, instead of the IBM 360/50 with seven drives which had been proposed and accepted on the basis of the performance goals, the goals were still met.

In all, twenty errors were discovered in the Conversational Subsystem during the five weeks of testing. Only two of those caused abnormal termination of the system; in other words, most of the coding errors were of such a nature that the system continued to function even though output was incorrect. Also, only nine represented bugs in the usual sense; the remaining eleven errors represented functions which had not been incorporated into the coding, or coding which performed as we expected but not as the Times desired. It is also of interest to note that twelve of the errors were in code written during the last two months of the nine-month coding period, and all were in code written during the last four months.

Acceptance testing of the Data Entry Edit Subsystem, which contained 13,421 lines of source code (about 16 percent of the overall system) was carried out during the third week. Pre-defined entries were keyed to test all identified features of this subsystem. However, due to pressure of other duties on the part of the indexers, little free-form testing was conducted. One error (misinterpreted function) was detected in this subsystem as a result of the formal tests.

The five other supporting subsystems, containing 18,884 lines of source code (about 23 percent of the overall system), primarily prepare files and tables for use by the Conversational Subsystem and produce listings, logs and statistical reports on the basis of outputs from it. Because of the variety of conditions required for and ensuing from these tests, it was agreed that the smaller subsystems would be sufficiently tested without the need for additional data. These subsystems were run on a regular basis during the five weeks of acceptance testing, and no errors were detected in any of them.

The complete system contained 83,324 lines of source code. Table I summarizes the total of twenty-one errors found during formal and free-form acceptance testing of the system. In the tables, "incorrect function" refers to code which operated improperly; "omitted function" refers to specifications not implemented; and "misinterpreted function" refers to code which did not perform precisely the functions specified.

OPERATIONAL EXPERIENCE

The File Maintenance Subsystem was delivered in June, 1970. It was used during 1970 and early 1971 to build files for the acceptance testing described above. Beginning in November, 1971, it has been in use on a

TABLE I—Errors Identified During Acceptance Testing

		Error Type			
Subsystem	Source Lines	Incorrect Function	Omitted Function	Misinterpreted Function	Total
File Maintenance	12,029	0	0	0	0
Conversational	38,990	9	8	3	20
Data Entry Edit	13,421	0	0	1	1
Other	18,884	0	0	0	0
Total	83,324	9	8	4	21

daily basis to add to the files new data keyed by the indexers and several years of past data converted from tapes used to publish The New York Times Index. Only two errors have been discovered in this subsystem, neither of which affected the data base. One of these involved incorrect function and the other misinterpreted function.

The Conversational Subsystem was delivered in June, 1971. It was used for experimental and demonstration purposes until November, 1971. Since that time it has been operational eight hours a day for online indexing and for inquiries designed to ensure the consistency of the operational files now being constructed. A total of seven errors have been discovered since delivery. Only one of these resulted in abnormal termination of the system, and this was due to lack of any capability in the System/360 Disk Operating System to handle the particular file error condition which caused it. (Additional application coding was added to circumvent the possibility of this error occurring again.)

The Data Entry Edit Subsystem was also delivered in June, 1971, and became operational on a daily basis in November, 1971. It had the least formal testing of any of the subsystems and has had a number of extensions made to it since delivery. Sixteen errors have been identified in this subsystem.

TABLE II—Errors Identified During Operation

		Error Type			
Subsystem	Source Lines	Incorrect Function	Omitted Function	Misinterpreted Function	Total
File Maintenance	12,029	1	0	1	2
Conversational	38,990	4	3	0	7
Data Entry Edit	13,421	8	5	3	16
Other	18,884	0	0	0	0
Total	83,324	13	8	4	25

The five other supporting subsystems have been used on an intermittent basis since their delivery in June, 1971. No errors have been detected in any of these during that period.

Table II summarizes the operating experience to date which has resulted in a total of 25 errors being identified, only thirteen of which involved incorrect function. This represents about three errors per 10,000 lines of code, a result which informal comparisons suggest is substantially better than average. From another standpoint, there was about one error for each five man-months of effort on the project. In fact, the programs written by the Chief and Backup Programmers had about one error per year of effort on their parts.

Consequently, initial operation has been very smooth. The important Conversational Subsystem has only suffered one abnormal termination due to an error in thirteen months of experimentation and operation; the other five errors prevented a single user from completing an inquiry or entering indexing data but permitted continued operation. To the best of our knowledge, no errors have been created in the files during two years of operation of the File Maintenance Subsystem. The experience with the Data Entry Edit Subsystem has not been as good, and it has suggested some changes in procedure discussed below.

CONCLUSIONS

Structured programming, and the organization and tools used to achieve it, were key factors in developing this kind of system. The fact that most of the errors encountered during acceptance testing were in code written during the last two months tended to confirm our expectations that the longer period of operation permitted by the top-down approach would lead to a more reliable system. The application of the program structure rules made it thoroughly practical for programmers to read, check and criticize each other's code and nearly eliminated the need for flowcharts as a means of communication. The Chief Programmer and Backup Programmer together reviewed much of the code on the project, particularly that of the more junior members of the team. This ensured that specifications and standards were being adhered to and that code would function as intended. Numerous problems were identified by code reviews which would otherwise have led to problems later.

The program library system used was also a major factor in improving quality. Ensuring that up-to-date versions of programs and data were always available reduced problems frequently encountered due to use of obsolete versions. For instance, when programmers were ready to use an interface, they could directly include the appropriate declarations into their code instead of writing their own version. When the interface changed, it was only necessary to recompile to incorporate a new version into all affected programs. In addition to reducing interface problems, the library system facilitated study of code to allow one programmer to adapt an approach used by another instead of re-creating it. Most importantly, it permitted the ready review and criticism of code by others as described above. As a side benefit, the availability of all this information in usable form reduced the need to get it verbally and thus further reduced errors due to distraction or interruption.

While it was not essential to structured programming, the use of the functional Chief Programmer Team organization had three major benefits in the area of program quality. First, the use of senior people directly in the design and programming process led to a cleaner, more rapidly implemented design. Second, use of a programming librarian to do many of the clerical tasks associated with creating, updating and maintaining programs reduced interruptions and diversions which tend to cause programming errors. Finally, the higher degree of specialization and smaller number of programmers led to a reduction in the number of misunderstandings and inconsistencies.

This project has suggested two areas in which further work needs to be done. First, it may not always be possible to follow a strictly top-down approach in development of a large programming system. If a system organization, viewed as a tree structure, is narrow and tall, then a pure top-down approach may take too much elapsed time to be practical. Second, a more rigorous approach to code review needs to be developed. In retrospect, a number of the problems encountered in the Data Entry Edit Subsystem after delivery were of such a nature that they would probably have been caught earlier if all the code had been read. The Chief and Backup Programmers did much functional coding themselves on the project, but it would probably have been more effective for them to have reviewed more code and written less. This would have reduced productivity slightly but would have eliminated a number of the remaining problems.

While the initial objective of the approach was improvement in production programming productivity, it became apparent that the same methods also resulted in increased quality. Experience gained on this project is leading IBM to more experimentation with structured programming and Chief Programmer Teams, and limited results to date confirm the conclusions reached here.

REFERENCES

1 H D MILLS
Chief programmer teams: Principles and procedures
Report No. FSC 71-5108—May be obtained from International Business Machines Corporation Federal Systems Division Gaithersburg Maryland 20760

2 F T BAKER
Chief programmer team management of production programming
IBM Systems Journal 11 No 1 pp 56–73 1972

3 G M WEINBERG
The psychology of computer programming
Van Nostrand Reinhold New York 1971 p 72

4 E W DIJKSTRA
Notes on structured programming
Report No EWD249 Technische Hogeschool Eindhoven Eindhoven Netherlands August 1969

5 H D MILLS
Mathematical foundations for structured programming
Report No FSC 72-6012—May be obtained from International Business Machines Corporation Federal Systems Division Gaithersburg Maryland 20760

6 H D MILLS
Top down programming in large systems
Debugging Techniques in Large Systems
Prentice Hall Englewood Cliffs New Jersey 1971 pp 41-55

7 C BOHM G JACOPINI
Flow diagrams, turing machines and languages with only two formation rules
Communications of the ACM 9 No 3 pp 366-371 May 1966