integration of that code. Now, as at all times during development, they are available to provide expert help or advice whenever troublesome problems are encountered.

Since the chief programmer is usually the key coder of the group, he frequently handles the detailed design and coding of the critical programs and program segments. This undertaking gives him the technical challenge he desires and ensures that the most difficult programs will be written by an expert programmer.

In his other major role as functional manager, the chief programmer deals with those management problems that directly influence his ability to get his job done. These have to do mainly with project planning and the allocation and control of material and people resources. He does not have to worry about administrative or personnel matters, however, because these are handled by a project manager.

His primary concern in carrying out his management role is to interface with the System Project Manager and to translate his directives into program designs, implementation plans, and work schedules that stay within the budget, time, and capability constraints of the installation. To do this, he often works with the backup programmer in sizing the jobs that they have before them

and assessing the tools and resources that are available to them. Together, they plan their workload to maximize their resource utilization and obtain the greatest return for their effort.

B. Backup Programmer

The second ranking member of the Chief Programmer Team is the backup programmer. Like the chief programmer, the backup programmer is also a senior programmer or analyst. He is a technically oriented individual with roughly the same qualifications as the chief programmer. In many cases, he is a potential candidate for chief programmer and sees this as the next step in his career path. Often, he is as capable as the chief programmer, but lacks the broad base of technical expertise that only time and experience can bring.

The backup programmer's main function is to assist and support the chief programmer in any of his work. He is as familiar as the chief programmer with the design and implementation processes and is in a position to assume the role of the chief programmer at any time, either temporarily or permanently. This is essential to ensure that development can continue in the absence of the chief programmer.

The backup programmer serves as the chief program-

mer's personal assistant. He is in a position to help the chief programmer in designing, coding, testing, and integrating any of the programs or routines that make up the new system. His store of knowledge can compensate for deficiencies in the chief programmer's technical repertoire, and he can serve as a valuable sounding board for testing any new ideas or techniques that the chief programmer may have. He can explore alternative design approaches, independent test planning, or any other development related task at the chief programmer's request. In addition, he can serve as a research assistant in the areas of programming strategy and tactics, and free the chief programmer to concentrate on the main problems of system development.

In any case, the backup programmer plays an extremely valuable role in the Chief Programmer Team. He is valuable to the chief programmer as a knowledgeable and capable assistant, and he is valuable to the project because of his backup role. His presence is reassuring to management because he represents the second person who is totally familiar with the developing project. He is a viable substitute for the chief programmer, and his standby position guarantees that the chief programmer's absence or departure will not put the whole project in

jeopardy.

C. Programming Librarian

The job of programming librarian may be handled by a programmer technician or a secretary with special training. The librarian's role is to relieve the chief programmer and the other team members of their normal operational and clerical duties. He gives this kind of assistance by submitting their programs for test runs and documenting the results. This may include anything from performing compilations, assemblies, and link edits to the actual execution of their code against test data. In addition, he is responsible for maintaining the Program Production Library. This library contains a chronological history of the development work done over the course of the project. Every computer output, regardless of whether it is good or bad, is filed into this library and an entry of it is made in the records. These records are helpful for maintaining a chronicle of development progress and providing a tally of the errors and problems discovered.

Another section of the Programming Production Library contains only usable, operational programs. This portion is a repository for all programs and data needed for development as well as for all programs and routines developed for the new system. The copies that are kept

here are for reference purposes and are maintained in both source code and machine readable form.

D. Supporting Members

The chief programmer, backup programmer, and programming librarian form the nucleus of the Chief Programmer Team. The rest of the team consists of programmers, system analysts, programmer technicians, and technical writers. Their job is to implement the system designed by the chief programmer and the backup programmer. These people design the non-critical parts of the system and do most of the coding, testing, and documentation associated with the project. They work under the close supervision of the chief programmer and backup programmer. The number of supporting members that make up the team depends on the size of the project. Members are added as the workload increases or may be dropped as it decreases.

Almost all members of the supporting team are experienced people. There is no room on the Chief Programmer Team for inexperienced people or trainees. Installations that use the Chief Programmer Team concept ordinarily assign these people to work on program maintenance rather than program development. It is not until they have acquired a reasonable amount of experience before they are allowed to join the team and work on developing and

writing new code.

3.3 CHARACTERISTICS

The Chief Programmer Team concept is characterized by a number of novel features. These features affect the procedures for implementing, retaining, and documenting new programs. They also impose limits on the size of the programming groups that work within the installation. Each of these distinguishing characteristics is described in the sections that follow.

A. Code Reading

In a conventional programming environment, whenever a program or sub-program is assigned to a person, he generally has the sole responsibility for that program. He designs it, codes it, debugs it, and documents it himself. He feels a sense of ownership toward it.

In a team operation, the feeling that a program can become a programmer's own personal product is played down. In this environment, emphasis is placed on making programs part of the public domain. Programs are considered to be public assets or team property. The team practices what is described by Gerry Weinberg as "egoless programming". (Reference 6).

The Chief Programmer Team produces this kind of

environment through the use of a number of program techniques. One of these is the walk-through in which a programmer presents his program design and his implementation plans for review and comment by his fellow programmers on the team. They go over his logic and read through his code. By doing this, good communication is maintained within the group. Each programmer knows what the other is doing and how he is doing it. The review sessions create a learning environment where programmers can pick out each other's program errors and pick up each other's good programming techniques.

The effect of the walk-throughs is reinforced by the critical review of all programmers' code by the chief programmer and backup programmer. It is their job to carefully read all the code produced to see that the standards of the installation are being met and to understand and validate all the programs developed so that no major flaws will go undetected.

Another factor that promotes "egoless programming" is the existence of the Programming Production Library. This library contains the output from all runs, both good and bad. It is a common repository for all developed programs, support programs, utilities, and test data. The librarian works diligently to keep all this information

current in order to enhance its value as reference material. The availability of this material gives programmers easy access to one another's programs and, as a consequence, facilitates communication among them. It is factors like these that allow ideas and programming techniques to pass freely among the whole group. Such factors tend to promote the idea that each piece of code is the product of the team as a whole rather than the sole product of an individual member. This increases the identification of the programs as public assets and is an important characteristic of the Chief Programmer Team.

B. The Programming Production Library

All programs under development and all programs and data required for support of the project are maintained in the Programming Production Library. The programs that are being developed are kept in two separate libraries called the internal library and the external library, while the programs and data that are being used to support the effort are designated as computer procedures or office procedures depending on their purpose. All of these programs are handled in one way or another by the programming librarian.

Of the two libraries, the internal library holds the code and the external library holds the listings.

The internal library contains copies of all the current project programs and data in machine readable form. This means that each of these programs or collections of data is represented by a copy in source code form, a copy in object (relocatable machine language) module form, and another in load module form. These copies may be stored on punched cards, or as data sets on tape or disk. The external library maintains copies of the programs and data in human readable form. It consists of all the program listings and includes the listings generated by the compiler, assembler, and linkage editor. These listings are kept in binders or folders for safekeeping and for easy filing.

Because each and every program listing produced is stored into the external library, its contents are always up to date, and, when these contents are taken as a whole, they reflect the history of the entire project. Consequently, this library is an extremely useful reference tool for the members of the Chief Programmer Team. Good intragroup communications can be maintained because the listings of each programmer are readily available to the others on the team. They can determine the status and progress of each other's work and can also determine the interface requirements between their own programs and

someone else's. Also, any intermodule testing that is required can be done simply by referencing the internal library for the other team member's program modules and linking these together with their own. The purpose of the Programming Production Library is to facilitate this kind of interchange among the team members.

The Programming Production Library provides additional functions in the form of computer procedures and office procedures. Computer procedures consist of the support programs that are necessary for performing the normal operations of a development group. These include procedures for compiling, assembling, link editing, and executing the development programs, as well as procedures for executing test case data sets, library updating routines, utility programs, and so forth. Since all these programs are available in the Programming Production Library, each team member does not have to put together his own set. This reduces the expenditure of redundant effort and allows time to be spent more productively. It also dictates a standard set of procedures for the entire team.

Office procedures are those support programs and operational rules that permit programmers to unload their clerical work onto the programming librarian.

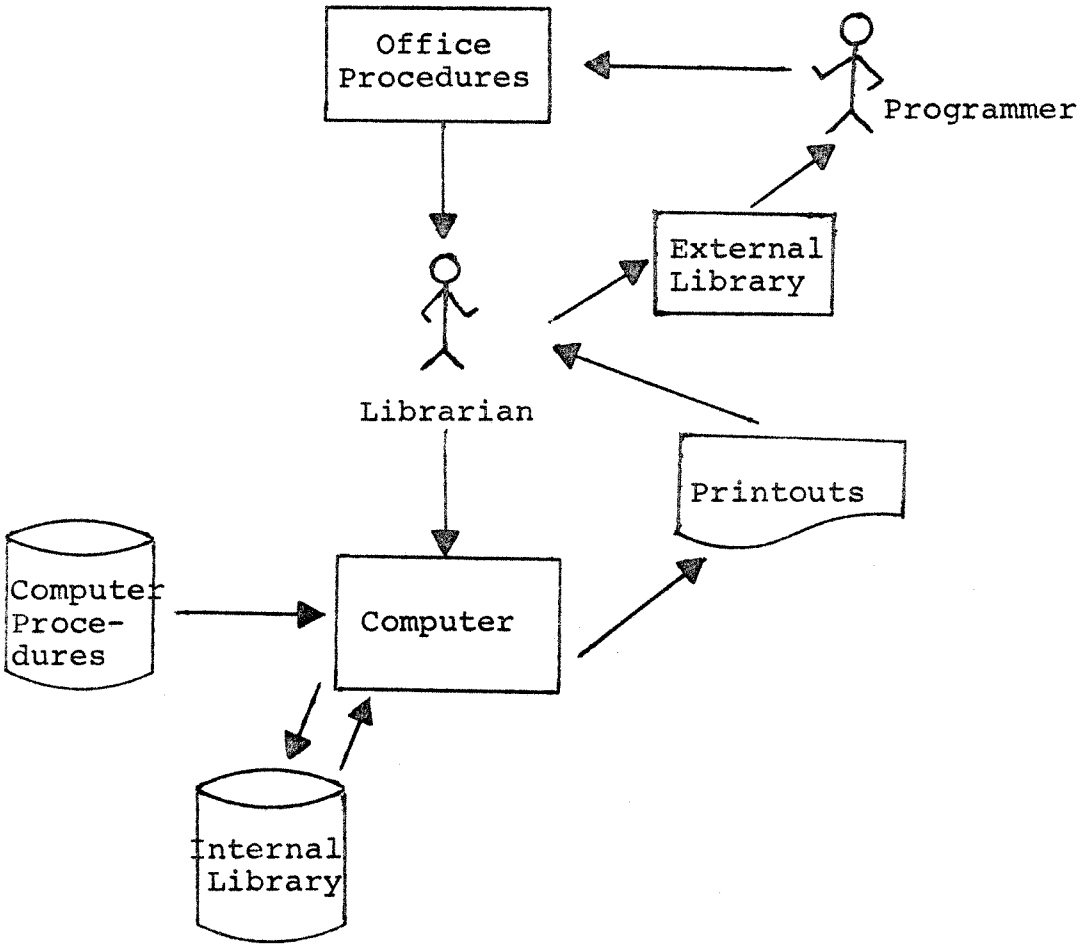
Through the use of these procedures, programmers can delegate some of their less critical, but time consuming jobs to the librarian. These include such things as making program modifications and updates, setting up programs for test runs, job submission, and the maintenance of data sets. Office procedures provide the programmer with standard methods for telling the librarian what needs to be done in each of these cases.

The librarian accepts the responsibility for these jobs and performs a vital service to the team. He relieves the chief programmer, backup programmer, and support programmers of most clerical work and allows them to use their time more effectively. By having one person in charge of the clerical area, there is less confusion, errors are reduced, job turnaround is more consistent, and programmer productivity is increased. (See Figure 2).

C. Documentation

In installations using the Chief Programmer Team approach, documentation is usually carried on in parallel with program development. Unlike the policy followed in most programming groups, documentation is not the responsibility of the programmers.

In the smaller Chief Programmer Teams, this task is the responsibility of the programming librarian. His



PROGRAMMING PRODUCTION LIBRARY

Figure 2

written descriptions of the programs augment the large amount of information already in the Programming Production Library. He has available to him the resources of the

external library which contains the complete set of all program listings. Since these include the results of the good as well as the bad runs, they represent the complete history of the development work done for the project. The retention of the bad runs for reference purposes helps programmers to learn from the mistakes of others and hopefully will help to prevent the re-occurrence of the same mistake.

In larger Chief Programmer Teams, technical writers may be added to assist in performing the documentation work. It is up to the chief programmer to decide if he needs them and to pick the number that he wants. The technical writers are required to work along and keep pace with the chief programmer, backup programmer, and support programmers in producing their program documentation. In this way, the documentation remains current and, more importantly, gets done. Good and thorough documentation will pay dividends later on when the programs will have to be maintained.

D. Installation Size

The Chief Programmer Team approach is applicable to installations with a total staff of 10 to 15 people. These would include a chief programmer, a backup programmer, a programming librarian, and, depending on the size

of the job, four to seven programmers, a secretary, and three to five other specialists. A project office might also be part of the team to help the chief programmer with financial, legal, and administrative matters, thus allowing him to concentrate his attention more fully on technical topics and management decisions.

In terms of program size, about 50,000 to 60,000 lines of source code seems to be the practical maximum that a team can produce. Work on programs or systems larger than this will necessitate subdividing the installation into several teams. By restricting the team to be relatively small in size, it retains its cohesiveness, a factor which aids productivity and program quality.

3.4 BENEFITS

The organization of the Chief Programmer Team is geared to efficiency. This efficiency derives from the team's highly centralized structure, the high degree of job specialization that is provided for its members, and the high level of technical expertise that they bring to their work. These factors lead to a group that is strong in discipline and unity. In addition, these factors also produce very beneficial results in the areas of program productivity, reliability and maintainability, project

control, and career orientation. The following sections will discuss the tangible benefits in each of these areas.

A. Program Productivity

The chief programmer and backup programmer are both experts in the data processing environment. Their close guidance at the program development and implementation stages helps the project maintain its proper course and prevents costly mistakes from occurring. They exert a strong initial effort to produce a good mapping of the design to the code and work hard at dividing the project into the segments that make the most efficient use of the skills in the team. Because of this, the Chief Programmer Team can tackle larger projects than other types of programming groups of the same size. The team can take on more work with fewer people and do it successfully. With fewer people (and with the services provided by the programming librarian), there are fewer interface and communication problems. This means fewer integration nightmares later on especially in light of the careful program design work done initially by the chief programmer and the backup programmer.

Productivity is further enhanced by the composition of the team. Normally, its membership consists of experienced people so minimal time is lost to education

and training. This means that from the project's very start, most of the work done is useful from the productivity standpoint. Also, during the course of the project, certain checks and balances are instituted to keep things under control. The design and code walk-throughs and the additional code reviews by the chief programmer and backup programmer verify the design and coding solutions that have been worked out. Such checking techniques act to reduce debugging and rework on the final product and consequently speed development. All these many factors work in concert from the start of design to the release of the finished product to yield programs of high quality in the minimum time.

B. Reliability and Maintainability

One of the characteristics of the Chief Programmer Team is the use of senior programmers in the positions of chief programmer and backup programmer. These people are directly involved in the design and programming processes. They expend a great deal of effort during the design phase to insure that the design will work and is well integrated and easily implementable. During the implementation phase, they code the critical programs and closely supervise and review the work of the other team members. Their technical knowledge and experience allows

them to spot and eliiminate many potential problems which would be more difficult, more time consuming, and more expensive to fix in the future. This kind of effort is made to ensure the production of a clean piece of work from the highest level to the lowest level.

The programming librarian contributes to improve program quality in several ways. First, he frees the technical staff to concentrate on technical matters. He does this by taking over most of their clerical work. Second, he makes certain that the internal and external libraries are synchronized and up to date. This eliminates problems caused by the use of obsolete versions of data or programs. Third, he and the technical writers work on the documentation and keep it current. Good documentation helps in promoting communication among the team members and in making future maintenance work easier. When these factors are coupled with the technical skill and high degree of specialization of the supporting team, the end result should be a product that is reliable and free of errors. Moreover, because good and abundant documentation is available, future rework and upkeep of the system should be simple and inexpensive. The depth of detail in the documentation and its historical perspective should permit trainees to perform the maintenance required and permit

them to acquire the knowledge and experience necessary to become future programmers or analysts on the Chief Programmer Team.

C. Project Control

In project control and measurement, the team operation offers certain built-in advantages over the conventional mode of operation. The functional responsibilities for the chief programmer, backup programmer, programming librarian, and the supporting team are separate and well defined. These sharp distinctions in functional responsibilities offer a ready-made structure on which to build a progress tracking and reporting system. Lines of communication between the various parts of the team already exist and are well defined so that it is easy to pinpoint the responsibility and the status of any member of the team. Consequently, it is normal for the chief programmer and the backup programmer to be constantly aware of the current status of the project and where the various responsibilities lie because of the nature of their jobs. The organization of the team automatically makes them the focal point for this kind of information. As a consequence, the problem of project control is limited to assimilating the information that is provided by each of the separate functional entities and making corrections or

adjustments where necessary.

The Chief Programmer Team's separation of functional responsibilities also has performance implications. Each function, because of its specialization, is expected to operate at a high level of performance. This provides increases in productivity and simplifies the chief programmer's job of assigning tasks. Because of the way the team is structured, he can assign tasks to each part of the team merely on the basis of whether that area is optimized for handling that type of functional responsibility or not.

D. Career Orientation

In the hierarchical structure of the Chief Programmer Team, the chief programmer has reached the pinnacle of his career for this kind of work. The backup programmer is a potential candidate for chief programmer, and the other programmers and system analysts are working to bring themselves closer to these two positions.

Each team member has a role to serve and can develop his talents and perfect himself in that position before moving on. He is exposed to strong positive influences from other team members including the chief programmer and the backup programmer. They can help him identify and correct his poor or inefficient coding habits and techniques. In effect, he is given the unique

opportunity to practice precision programming under expert guidance without becoming encumbered by clerical duties since these are handled by the programming librarian. This allows him to dedicate more of his time to develop and improve his technical skills. In these ways, the team operation promotes professional growth and provides a good environment for building the team leaders of the future.

3.5 SUMMARY

The Chief Programmer Team is an organizational approach to the programming development process with the expressed purpose of improving the quality of programs developed as well as increasing the productivity of the development team. In the conventional environment, a programming group consists of a large number of people with widely varying amounts of experience in design, coding, and testing. Unfortunately, the results of their work very often exhibit as much variation in quality as their variation in experience. The Chief Programmer Team overcomes this problem by replacing the conventional programming group with a team of small, functionally specialized groups composed of experienced and highly skilled programmers. These people form a highly qualified and capable organization, but the key factors in their

success as a group are the chief programmer and his assistant, the backup programmer. These two positions make up the core of the team and must be filled by experienced and extremely competent individuals. They are the technical leaders of the group and carry the responsibility for designing the system, managing the implementation and testing efforts, and personally programming the system's most critical areas.

The organization of the Chief Programmer Team defines the roles of the all specialists on the team and their relationships to each other. It provides a new documentation and communications aid in the Programming Production Library. It provides for the efficient production of programs which are both well designed and error free because each stage of development was done under the close scrutiny and supervision of the two team leaders. Its relatively compact size makes it a more unified group and makes it more immune to the difficulties encountered at integration time because of poor communications. The end results produced show the influence of all these factors. The finished product is a well documented system of programs that have not only undergone review by the original author, but also by all other team members including the chief programmer and the backup programmer.

The product is released with a high degree of confidence in its correctness as well as its overall quality.

3.6 REFERENCES

1. Baker, F. Chief Programmer Team Management of Production Programming. IBM Systems Journal, 1972, Vol. 11, No. 1.

   This article describes how the Chief Programmer Team concept was put in place to develop the New York Times Information Storage and Retrieval System.

   It is a success story portraying the development of an 83,000 line program with a phenomenally low error rate of one per every 4,000 delivered instructions.

2. Baker, F. and H. Mills. Chief Programmer Teams. Datamation, December 1973, Vol. 19, No. 12.

   This article gives a good introduction and description of the concepts behind the Chief Programmer Team.

3. Canning, R. (ed) Issues in Programming Management. EDP Analyzer, April 1974.

   This article provides a comparison between the Chief Programmer Team and other programming organizations. It discusses the benefits and drawbacks of

each type of organization as well as the role of the individual programmer and his performance in each of these environments. This article is valuable as a general reference.

4. I.B.M. A Disciplined Approach to Application Development. IBM Corporation, Productivity Techniques, Bethesda, Md. 20034.

This report defines the organization of the Chief Programmer Team and describes the roles of each of the team members.

5. I.B.M. Improved Programming Technologies. IBM Corporation, Data Processing Division, Installation Productivity Programs Department, White Plains, N.Y. 10604. 1973.

This report contains essentially the same material that is in the previous document except it includes the advantages that a team operation offers when it is properly formed.

6. Weinberg, G. The Psychology of Computer Programming. Van Nostrand Reinhold Company, New York, N.Y. 1972.

Weinberg addresses the human dimensions of programming performance and introduces the concept of "egoless programming".

CHAPTER 4

TOP DOWN STRUCTURED PROGRAMMING

4.1 DEFINITION

Structured programming is a programming technique that goes a long way toward bringing discipline and order to the whole program development and implementation process. It advocates the use of a limited language subset and a restrictive set of rules as a means of improving the clarity, readability, and maintainability of programs. The theorems that structured programming is based on were introduced by C. Bohm and G. Jacopini in the mid 1960's. In the early 1970's, they were mathematically proven to be applicable to all programming situations and implemented as a set of technical programming standards by Harlan Mills. Since then, many of these standards have been adopted as normal programming practices in many software groups.

The benefits provided by the use of structured programming techniques is augmented by the top down programming process. In essence, top down programming is the downward expansion of functional specifications from very

high level functions to simpler and simpler functions. The expansion process produces a tree-like structure where the nodes at each level represent functions of a uniform complexity. The lower down the levels are, the simpler the functions are, and the lowest level contains the most elementary functions to be implemented. Experience has shown that the top down technique is an excellent design tool when it is used in conjunction with structured programming.

4.2 STRUCTURE

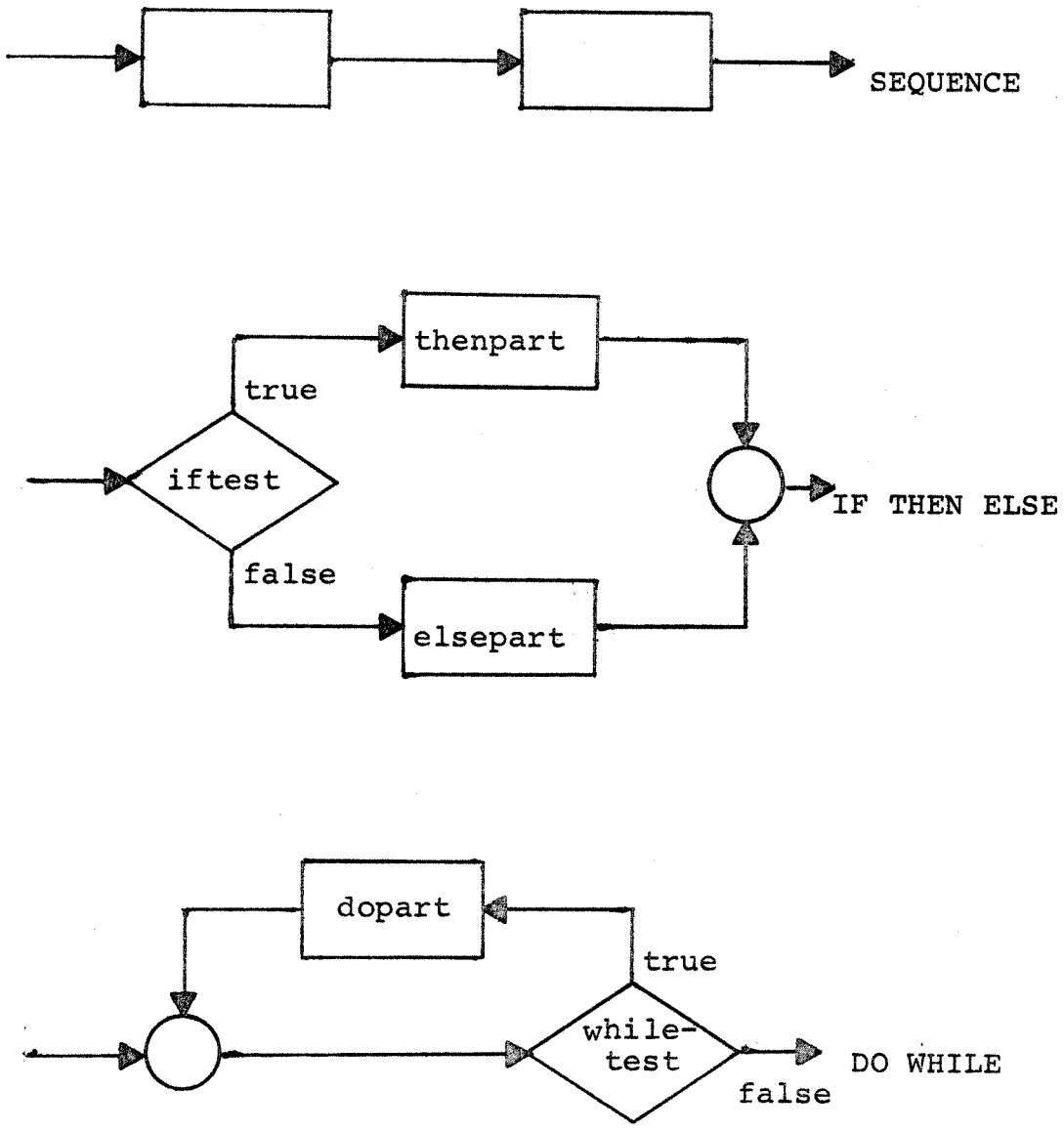
The basic principle behind structured programming was introduced by C. Bohm and G. Jacopini in their paper entitled "Flow Diagrams, Turing Machines, and Languages With Only Two Formation Rules" (Reference 2). This principle states that any proper program can be written by using three basic program figures or structures, (See Figure 3), namely:

1. a sequence
2. IF a THEN b ELSE c
3. DO b WHILE a

These terms are defined as follows:

Sequence - A sequence is a collection of code whose statements are executed in the order written. It does not

contain any statements that alter the sequence of execution.



STRUCTURED PROGRAMMING BASIC FIGURES

Figure 3

IF a THEN b ELSE c - In this expression, a represents a logical relation and b and c represent one of the three basic structures listed above. The operation of this expression is such that if condition a is true, then the structure b is executed and an exit is performed, while, on the other hand, if condition a is false, then the structure c is executed and an exit is performed.

DO b WHILE a - This expression is a loop control mechanism. The term a represents a logical relation and the term b represents one of the three basic structures (b's execution may have an effect on a). The operation of this expression proceeds as follows. Control enters at the single entry point where test a is performed. If this condition is true, b is executed and control loops back to the testing point where condition a is tested again. If this condition remains true, control proceeds again to structure b, while if it has become false, control bypasses b and leaves by the only exit provided.

Proper Program - A proper program is a program which exhibits the following two attributes:

1. It contains one and only one entry point and one and only one exit point.
2. For any point within the program, there is a logical path that goes from the entry point to that point and

another logical path that goes from there to the exit point.

C. Bohm and G. Jacopini proved that these three basic logic structures were necessary and sufficient for implementing all proper programs. They went further to demonstrate that all general programs could be expanded into or realized by a set of proper programs. These results showed for the first time that all computer programs, no matter how simple or complex, could be implemented through the use of the three basic structures. Their impact was extremely far reaching for they established the foundation for the structured programming standards and techniques that exist today.

The discovery of the principles of structured programming made it possible to make full use of the top down design concept. This is because a program written according to structured programming concepts can have its control flow move from top to bottom without passing through any unconditional branches. Control flow of this type is characteristic of the orderly tree-like structure of top down programming designs. Code written in this manner is easier to follow and easier to understand because control does not jump from place to place in the program.

The readability of such code can be further enhanced by indenting all the statements that belong to the same level of the tree structure by the same amount. This makes the hierarchical structure of the program highly visible and allows it to be discernible at a glance. Good readability of this sort greatly simplifies the scanning of the program whether it is being checked for correctness or being analyzed for maintenance purposes.

Another positive factor for maintainability and for testing as well, is the one entry, one exit organization of the program segments that make up the program. A program segment usually consists of one page of code or less. Control enters at the top of the code sequence and exits from the bottom. There are no other means of entry or exit in the segment. This organization reduces the effort required to follow the flow of program control. In maintenance work, this is important because the programmer making a modification to a program segment can be reasonably certain that his change will not produce unwanted side effects. From the testing standpoint, this straightforward sequential organization simplifies the task of finding the errors in the program. Top down design lays out the program structure in such a way that changes in one segment will not affect other segments, and

also such that errors are isolated and do not propagate throughout the whole program.

Code in a structured programming environment that employs top down design is generated in the same order that it will be executed. The job control statements to compile, assemble, link edit, and execute the source code are generated first. Then, the source code is created in a top down order. The highest level of the program hierarchy is coded first, followed by the next lower level, and so on until the lowest level is coded. This sequence of code generation has strong implications on the implementation and testing phases of program development.

4.3 CHARACTERISTICS

The characteristics of a top down structured programming environment are found in every phase of program development. Their influence extends to all phases from design to coding to testing to maintenance. This was not always the case, however. Years ago, the influence of top down structuring concepts was not nearly so far reaching. Even though top down design had been practiced for some time, implementation was almost always done from bottom up. Segments of the program were first coded and tested as separate units and then brought together at integration

time to be tested as a whole. Top down programming practices have reversed this routine by restricting the coding process to follow the same sequence as that followed in top down design. The code is created in the order that it will be executed. This top down implementation arrangement permits testing and integration to occur simultaneously and results in continuous verification of the program at each step of its development.

A. Design

An important factor that program developers must keep in mind is that programs have long lives and must be maintained for as long as they are used. One of their foremost considerations should be to simplify the task of program upkeep. Both management and development people must realize that this is a significant and achievable goal. They must be willing to invest the time and effort required to produce a careful, well thought-out, well integrated design. This will result in a more maintainable product and one that is free of costly errors.

In top down design, the system is laid out in a top down fashion, starting from a broad overview at the top level and progressing into more detail at each lower level. The result is a tree-like structure where the nodes at each level represent program functions at approximately

the same level of detail. This structure forms the basis for the top down programming approach. It is created early in the design phase. At this time, the skeleton or general structure of the program is laid out. Next, the basic functions of the program are determined and are mapped onto the highest levels of the structure. Then each basic function is broken into more detailed subfunctions which define and form the nodes at lower levels of the tree structure. Each subfunction in turn is broken into more elementary subfunctions and so on until all subfunctions have been resolved to a sufficient and consistent level of detail. The most elementary subfunctions form the nodes at the lowest level of the structure.

The branches that run between the nodes of the structure point out the relationships between subfunctions. They present a system of pathways that trace out the dependencies of high level functions. If each node represents a function or subfunction that can be realized by a program segment, then the branches define the interfaces between program segments. This is a very clean and graphical way of specifying interface relationships. Its use should prevent inconsistent interface definitions and the integration problems they cause.

The branches of the tree structure also perform

another important role. They help to define the dataflow for the entire program. The branches form paths which show the accessability of data to all the program segments (nodes). They show the segments that have access to a particular kind of data and those which do not. The ability to determine the range or scope of the data processed by a program is important especially during program testing since it can help to pinpoint the location of problems.

Program testing also receives a substantial amount of attention during the top down program design phase. A design is considered incomplete unless it has a thorough set of testing specifications. These specifications include interface, timing, priority, and performance requirements, sets of initial conditions, and data specifications to prevent the creation of unnecessary or redundant test cases.

The design phase in a top down structured programming environment is usually more lengthy than in the normal programming environment. This is natural in view of all that must be done. The design must be carefully worked out if the result is to be a well integrated system. It must be plotted out from the highest level functions to the most detailed low level subfunctions. Interfaces and

data flow between each of the program segments must be defined. Testing specifications must also be defined and written. If these stringent requirements are met, the end result will be a design that minimizes the interaction and data sharing among the program segments.

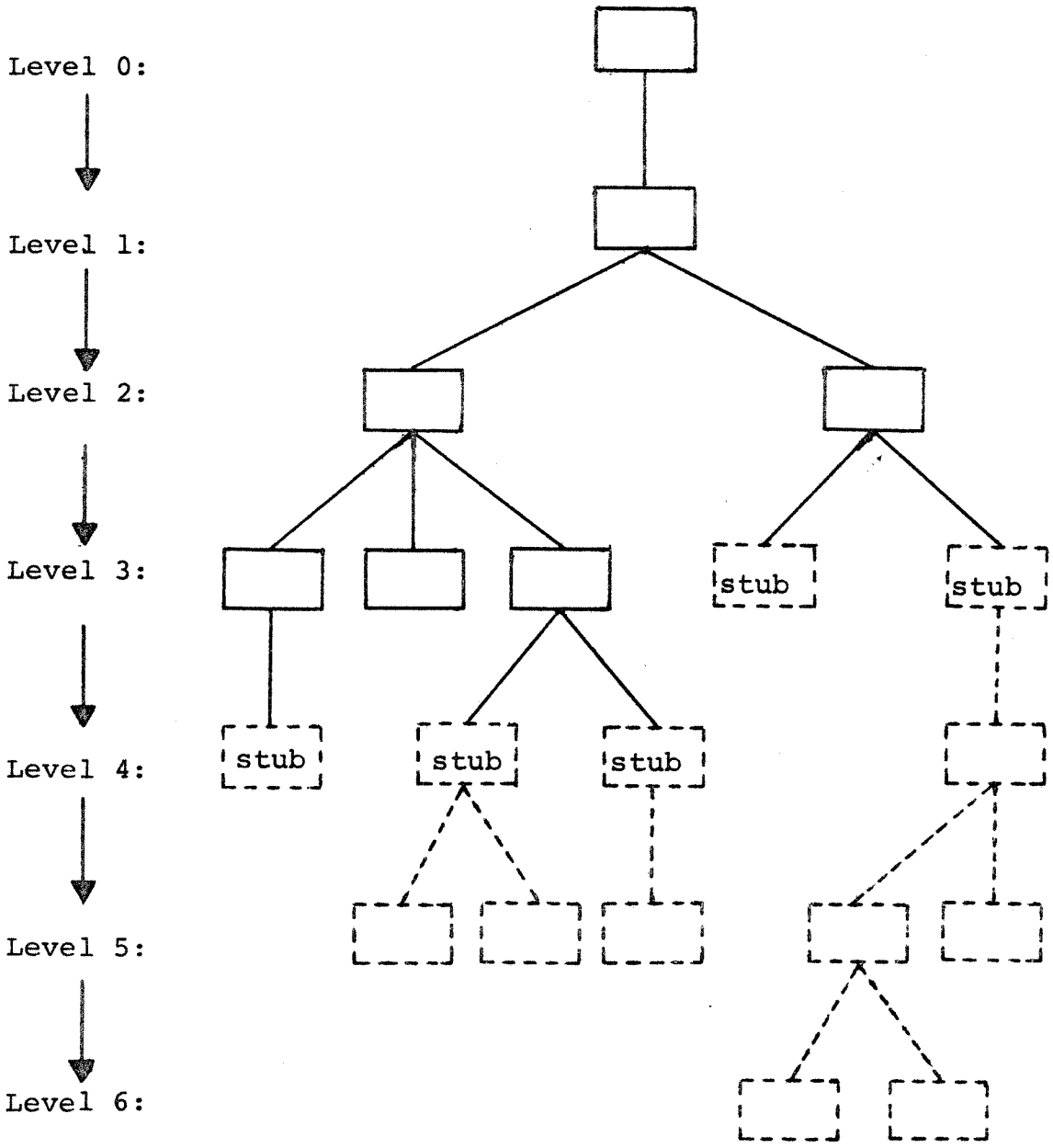
B. Coding

The hierarchical nature of a top down structured programming design facilitates the code development process. The carefully laid out structure produced from the earlier design phase serves as a guide to coding development. It functions like a road map and more or less forces code implementation to follow a specific sequence.

This sequence begins with the coding of the high level program segments which form the framework for the entire system. These contain the logic of the high level functions and these functions in turn refer to their subfunctions which are represented by the program segments at the next lower level. At this time, however, the lower level segments do not exist. They are either being coded separately or will be coded later. In the meantime, dummy program segments or stubs are used in their place. In this way, work can proceed on the high level program segments. They can be coded, compiled, assembled, link edited, and tested without the presence of any lower level

program segments. Once the upper level segments have undergone complete testing, the program segments that represent the next lower level can be created using the same scheme of referring to dummy program segment names or stubs for their subfunctions at the next lower level. Consequently, as the system is coded, it goes through repeated cycles of integration and system testing. The level by level process of code implementation and testing continues until the lowest level of the structure is reached. At this point, there are no more dummy segments or stubs to insert. After this level is coded and completely tested, the system is in working condition and is ready to enter production mode. (See Figure 4).

These are the factors that determine the philosophy of coding in a top down structured programming environment. There are also several factors which influence the methodology of coding in this environment. The first is the importance of maintaining unambiguous interfaces between the calling program segments at the higher level and the called program segments at the lower level. This prevents interface problems from occurring as the system undergoes its level by level integration process. The second factor is the one entry, one exit property of program segments. In every segment, control



TOP DOWN DEVELOPMENT

Figure 4

must enter at the top at a single point and exit from the bottom, also at a single point. Every program segment either returns to its caller or moves in line to the next segment. The last factor is the small set of programming commands that the structured programming environment allows. Because of this restricted set of building blocks, code generation during implementation may be more difficult, but this is more than offset by the virtues of better readability, better testability, and better maintainability in the code produced.

C. Testing

In a bottom up programming environment, the testing process begins at the lowest level. It starts with unit testing and is followed by integration testing which can be carried out only after all the units are completely coded. Unit testing is usually no cause for alarm, but integration testing is another matter. When all the code is brought together and the program stops, the job of isolating and finding the error or errors may be enormously difficult. Typically, the source code for most units was written long before integration was started, and both the code and the program logic may have since been forgotten. Also, the unit testing of certain units may be incomplete, and this automatically makes them suspect.

These uncertainties make error determination a long and tedious process, because large numbers of possibilities must be checked out and eliminated.

In this kind of environment, regression testing is a common practice. Every time new code is added, it is verified against test cases that had run successfully before. This is helpful in isolating errors since they could be caused only by the new code or by the new code uncovering paths in the lower level code that had been untouched previously. Integration performed in this way is piecemeal in coverage and very time consuming. Many regression tests may be necessary and moreover, the basic structure of the entire system cannot be verified until all the individual units are brought together and tested as a whole.

When the programming environment is top down, the program segments representing the high level functions are coded first. These segments are then put through a testing phase and checked for errors. Once the testing is completed, program segments representing the subfunctions at the next lower level are coded and added to the system. They are combined with the previously coded and tested segments and are run through a new series of tests. Since the new program segments are at a lower level and do not

refer back to the higher level segments, any errors discovered at this stage must be in the new code. This cyclic pattern of alternating between coding and testing is carried out level by level until the lowest level subfunctions have been coded and tested. Once the last level has been completely tested, the entire system is ready to be released for production work.

With top down development, integration and testing are performed on a level by level basis. Functions are added to the system in a logical sequence and in the order of dependency. They are added one at a time or in small groups rather than all at once. Moreover, they are added to a working system which means that errors discovered after a new integration step are most likely caused by the newly incorporated code. Also, because of the hierarchical structuring of the system, such errors do not impact the already tested functions.

The top down environment permits the continuous verification of the basic system logic and the interfaces between the different levels. It provides a good overview of the whole implementation and testing process, and consequently, gives better project control and predictability. Progress tracking is simplified by the alternation between coding and testing as development proceeds from

level to level, and confidence in the code produced is bolstered by the fact that the most critical code is tested the most thoroughly. Because integration is a continuous process, top down structured programming avoids the problem crunch that usually occurs at system integration time in other programming environments. It eliminates the need for each programmer to develop, write, and debug the drivers necessary to test their individual units. Early integration also provides benefits. It unveils the work of poor programmers early in the schedule so that corrective action can be taken. More importantly, it motivates good programmers because it makes their progress and high level of achievement very visible.

During the testing phases, the single entry point, single exit point restriction on each program segment makes error detection much simpler. It minimizes the connections and lowers the dependencies between segments. It produces simple interfaces which reduce the incidence of interface errors. Also, when debugging activities are being carried out, once it has been established that control is within a program segment, its point of entry and its point of exit are known precisely.

4.4 STANDARDS

Before beginning a project, every installation using structured programming should establish a set of programming standards. These standards set the guidelines and rules for design, coding, testing, and documentation. They specify what kinds of things are acceptable in each of these areas, and, taken as a whole, they set the level of program quality for the installation. As such, they provide designers, programmers, and analysts with common points of measurement for evaluating their work.

Items that should be included in the programming standards are the following:

* The use of only a predefined set of basic programming figures or structures. These structures may be nested; however, regardless of the number of levels of nesting, there can be only a single entry point and a single exit point.
* Disallowing the use of unconditional branches (GO TO statements or their equivalent). Unconditional branches are not necessary and should be used only in certain exceptional cases.
* The setting of a limit on the size of each

program segment or function. The length should be restricted to the amount of code that the human mind and eye can easily comprehend at any one moment.

* The use of an indentation scheme whose organization corresponds to the nested hierarchy of the functions. This makes the code easier to read and more easily understandable.

Each installation should have its own specific rules with regard to languages, file structure, data management, and so forth. In any case, management has to handle the enforcement of these rules. It must constantly check to see that these standards are strictly adhered to if the installation is to produce programs that are consistently high in quality.

4.5 BENEFITS

The top down structured programming approach to design and implementation provides many worthwhile benefits. These include lowered software costs, higher productivity, and better quality software. All these benefits accrue from the positive influence that top down structured programming exerts on each of the development phases. Starting with the design phase, this approach provides for a system design that is clear and well

defined. The top down ordering concept is utilized to reduce complex functions into simpler functions and to provide an orderly way to handle the implementation of complex program applications. There is early definition of the interfaces between program segments so that interface errors are kept to a minimum. All through the design process, clarity is maintained. This helps to eliminate costly programming errors due to misinformation or the misinterpretation of design specifications.

In the coding phase, the limited set of options permitted by structured programming greatly restricts coding flexibility. This reduction in programmer freedom is matched by a corresponding reduction in the error rate and an increase in programmer productivity. No unconditional branches are permitted, and all program segments are restricted to have only one point of entry and one point of exit. The resulting code can be scanned in a straight-line fashion. This improved readability makes the code easier to debug during testing and easier to understand when maintenance becomes necessary. In most cases, maintenance can be handled by trainees or inexperienced programmers. The code is usually sufficiently straightforward so that modifications can be made to one program segment without causing any deleterious side

effects in other program segments.

During the testing phase, error checking and error isolation are simplified by several factors. First of all, new code is added at most a level at a time to previously tested code. This limits the amount of code tested to a reasonable aggregate. Second, the top down ordering of functions stipulates that higher level functions can call lower level functions, but not vice versa. This means any error discovered during this round of testing are in all probability rooted in the new code. Third and last of all, limiting each program segment to one entry point and one exit point makes it easy for the programmer to follow the flow of control from segment to segment. This is especially useful if he is provided with the tools to perform online testing.

The most outstanding benefit by far, however, is the lack of a consolidated system integration phase. Prior to the use of top down programming, system integration was always considered a giant headache and placed at the end of the development phase. Integration was the first time that all the separate units of the system would be brought together for system test. It demanded peak manpower utilization, peak machine time requirements, and peak management involvement. It was a chaotic situation and the results often reflected this.

By contrast, top down programming allows for a process of continuous integration. Machine utilization remains relatively stable since coding and integration are done almost in parallel. The workload does not reach a peak but is spread out so that both management and staff can proceed at an orderly pace. This disciplined approach is more conducive to the production of high quality programs.

4.6 SUMMARY

The timely production of well designed, well written programs that are free of problems and easy to maintain is a highly sought after goal at all programming installations. Though this goal is exceedingly difficult to achieve, certain programming methods have been developed to make it more attainable. Top down structured programming is such a method. This approach combines the attributes of top down design with those of structured programming to create an extremely comprehensive programming methodology. It covers all areas of programming from design to integration to maintenance and provides techniques that improve the productivity in each of these areas.

The top down structured programming approach

utilizes the top down ordering concept to make the implementation of complex programming applications easier to handle. Top down ordering permits a complex function to be represented in terms of a multi-level functional hierarchy. The function at the top of the hierarchical structure is the complex function itself, while those at each successively lower level represent the more and more elementary subfunctions that are its components. These subfunctions become the program segments of the top down structured programming system. The program segments at any one level can call the segments in the level immediately below or be called by the segments in the level immediately above. Once the design has been established, program development follows the top down pattern set by the structure. This development sequence provides the high visibility that is necessary for good progress tracking and effective project control. Checkpoints are easy to define and use as gauges of progress. Problems are discovered early and can be dealt with immediately.

Aside from aiding management, top down structured programming also increases programmer effectiveness. It helps programmers to produce code that is uniform in quality and easy to understand. Moreover, the code has excellent maintainability. Modifications can be added

to one program segment without producing unexpected side effects in other segments because of the small degree of interaction between program segments and the clarity of the code. In addition, the limited coding options available under structured programming create such a disciplined and restricted coding environment that coding tasks can be assigned to inexperienced programmers.

As useful and as powerful as it is, the top down structured programming approach, nevertheless, suffers from a number of problems. The design phase is long and drawn out because this is the most important of the developmental phases. The limited instruction repertoire makes programs more difficult to code. For a given application, more code is usually generated, and a substantial amount of this extra code is repetitious in nature. This results in less efficient machine utilization since more storage and longer run times are needed. Structured programming also requires the establishment of restrictive standards and strict enforcement of these standards to be effective. All these disadvantages are a small price to pay, however, for a system of programs that is well designed, easy to debug, and easy to maintain.

4.7 REFERENCES

1. Baker, F. System Quality Through Structured Programming. Proceedings AFIPS 1972, Vol. 41, Part 1.

This report describes how structured programming is used in a Chief Programmer Team environment. Much of the material it contains is similar to what is in the author's article titled "Chief Programmer Team Management of Production Programming".

2. Bohm, C. and G. Jacopini. Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules. Comm. ACM, April 1966, Vol. 9, No. 4.

This is the foundation of what would later become structured programming. It is a substantial contribution to the field of programming and well worth reading.

3. Canning, R. (ed) The Advent of Structured Programming Management. EDP Analyzer, June 1974.

This report contains an analysis of different approaches to structured programming and describes some user experiences with this programming technique.

4. Dahl, O., E. Dijkstra, and C. Hoare. Structured Programming. Academic Press, New York, N.Y. 1972.

In this book, Dijkstra considers that if the art of programming is properly disciplined, it can be made

to have the rigor necessary for applying the theorem proving processes used in science. The correctness of a program (considered modular in structure) can then be established by using mathematical induction over certain already proven assertions. This work is fairly heavy reading, but well worth the effort.

5. Dijkstra, E. GO TO Statement Considered Harmful. Comm. ACM, March 1968, Vol. 11, No. 3.

This letter on the GO TO statement was written by Dijkstra when he became convinced that unconditional branches should be abolished from higher level languages.

6. Donaldson, J. Structured Programming. Datamation, December 1973, Vol. 19, No. 12.

This article describes how control paths can be simplified through the use of structured programming.

7. Gries, D. On Structured Programming. Comm. ACM, November 1974, Vol. 17, No. 11.

This is a good article for use as a reference since it covers the major advancements made in the area of structured programming.

8. Liskov, B. A Design Methodology for Reliable Software Systems. The MITRE Corporation. Proceedings AFIPS 1972, Vol. 41.

This paper is limited to the description of SPIL - A Language for System Design and Implementation.

9. McCracken, D. Revolution in Programming. Datamation, December 1973, Vol. 19, No. 12.

This article provides a general overview of structured programming.

10. Miller, E. and G. Lindamood. Structured Programming: Top down Approach. Datamation, December 1973, Vol. 19, No. 12.

This article gives a general overview of structured programming with emphasis on the top down design approach.

11. Mills, H. Mathematical Foundations for Structured Programming. IBM Corporation, Federal Systems Division, Gaithersburg, Md. 1972. FSC 72-6012.

This is the mathematical proof that simple structured programming control logic is capable of expressing any program requirement. It is an interesting piece of work, but limited in usefulness as a reference for information about the structured programming method.

12. Mills, H. How to Write Correct Programs and Know It.

IBM Corporation, Federal Systems Division, Gaithersburg, Md. 1973. FSC 73-5008.

This report is a very delightful, clearly stated overview of structured programming.

13. Mills, H. Top-down Programming in Large Systems. Debugging Techniques in Large Systems. R. Rustin (ed). Courant Computer Science Symposium 1, New York University. Prentice-Hall. Englewood Cliffs, New Jersey. 1973.

This is a very well written article on top down structured programming. It provides good references to other work being done in this area.

CHAPTER 5

CONCLUSIONS

The high cost of software development has forced the data processing industry to look for new and practical ways to produce software. The goal has been to beat back the spiraling costs by achieving gains in productivity, quality, and maintainability. Over the last several years, this effort has yielded significant improvements in programming and management techniques. It has brought about the introduction of modular programming, a complete package which combines a new programming methodology with a new approach to group organization; the Chief Programmer Team which is a new concept in team organization; and top down structured programming which is a new and highly disciplined methodology based on recent theoretical findings.

Each of these three new approaches has limitations and can be used effectively only when matched with the particular needs of an installation. In general,

modular programming is well suited to large installations where the bulk of the work is in development rather than in maintenance. The skill mix within the group is not critical and can vary from highly experienced programmers and analysts to junior programmers and trainees. This work environment offers job satisfaction to all its members since meaningful jobs can be found to match all skill levels. The loss of personnel due to normal turnover does not cause noticeable disruptions, and the workload can be shifted around to a moderate degree without any adverse effect on the schedules.

For small and medium sized installations, the Chief Programmer Team approach should produce a better fit. Effective operation under this approach requires the proper skill mix, however. The chief programmer and his assistant, the backup programmer, must be highly experienced and capable individuals. The members of the support team must also be experienced people. Otherwise, an inordinate amount of the workload and responsibility falls on the chief programmer and the backup programmer. The work environment here places a large amount of emphasis on technical problem solving. The team members benefit from the constant exposure to the highly skilled people around them. This creates an atmosphere for promoting

professional growth and makes it an ideal training ground for future team leaders. A drawback in this approach is the heavy dependency that is placed on the chief programmer. Even though he is closely supported and backed up by the backup programmer, his departure from the project may result in a major schedule setback.

Top down structured programming is a programming methodology that brings a disciplined approach to the processes of design, coding, testing, and integration. Its main objective is to divide a system in such a way that individual pieces can be designed, implemented, or modified without affecting the rest of the system. The net effect is the reduction of a complex problem into a hierarchy of small and manageable pieces. This programming method promotes production of code that is well designed, easy to debug, and easy to maintain. It is an approach that most users, regardless of size or work environment can adopt. It does, however, require a management that is attuned to its demands of strict discipline and need for well documented standards.

Modular programming, the Chief Programmer Team, top down structured programming - all of these new approaches to programming have their strengths and weaknesses, advantages and disadvantages. Any installation

that is interested in adopting one of these new approaches should investigate its needs carefully. Before coming to a conclusion, it should examine its operating environment, the typical kinds of applications to be done, the skill mix of its programmers and management, and the size of the operation, among other factors. Only then can an intelligent choice be made from among these three approaches or any others that may be under consideration. The proper decision will result in a host of real benefits to the installation in terms of improved program quality and reduced costs.

BIBLIOGRAPHY

Baker, F. Chief Programmer Team Management of Production Programming. IBM Systems Journal, 1972, Vol. 11, No. 1.

_____ System Quality Through Structured Programming. Proceedings AFIPS 1972, Vol. 41, Part 1.

Baker, F. and H. Mills. Chief Programmer Teams. Datamation, December 1973, Vol. 19, No. 12.

Bohm, C. and G. Jacopini. Flow Diagrams, Turing Machines, and Languages With Only Two Formation Rules. Comm. ACM, April 1966, Vol. 9, No. 4.

Boehm, B. Software and Its Impact: A Quantitative Assessment. Datamation, May 1973, Vol. 19, No. 5.

Cammack, W. and H. Rodgers. Improving the Programming Process. IBM Corporation, Poughkeepsie, N.Y. 12601. 1973. TR00.2483.

Canning, R. (ed). The Search for Software Reliability. EDP Analyzer, May 1974.

_____ Issues in Programming Management. EDP Analyzer, April 1974.

_____ The Advent of Structured Programming. EDP Analyzer, June 1974.

Clark, R. A Linguistic Contribution to GOTO-less Programming. Datamation, December 1973, Vol. 19, No. 12.

Dahl, O., E. Dijkstra, and C. Hoare. Structured Programming. Academic Press, New York, N.Y. 1972.

Dijkstra, E. Structured Programming. Software Engineering Techniques, NATO Science Committee. J. Burton and B. Randell (eds).

_____ GO TO Statement Considered Harmful. Comm. ACM, March 1968, Vol. 11, No. 3.

_____ The Humble Programmer. Comm. ACM, October 1972, Vol. 15, No. 10.

Donaldson, J. Structured Programming. Datamation, December 1973, Vol. 19, No. 12.

Freeman, W. Computer Models of Thought and Language. R.C. Schauk. Kenneth Mark College.

Goldberg, J. (ed) Proceedings of a Symposium on the High Cost of Software. Proceedings of a Symposium Sponsored by the Air Force Office of Scientific Research. Stanford Research Institute, Menlo Park, California. SRI Project 3272.1973.

Gries, D. On Structured Programming. Comm. ACM, November 1974, Vol. 17, No. 11.

Hamilton, M. and S. Zeldin. Top Down, Bottom Up Structured Programming and Program Structuring, MIT, The Charles Stark Draper Laboratory, Cambridge, Massachusetts 02139. E-2728, Revision 1.

Hoskyns Systems Research, Inc. Implications of Using Modular Programming. Central Computer Agency, Guide No. 1. New York, N.Y. 1973.

I.B.M. A Disciplined Approach to Application Development. IBM Corporation, Productivity Techniques, Bethesda, Md. 20034.

_____ Improved Programming Technologies. IBM Corporation, Data Processing Division, Installation Productivity Programs Department, White Plains, N.Y. 10604. 1973.

Liskov, B. A Design Methodology for Reliable Software Systems. The MITRE Corporation. Proceedings AFIPS 1972, Vol. 41.

McCracken, D. Revolution in Programming. Datamation, December 1973, Vol. 19, No. 12.

Miller, E. and G. Lindamood. Structured Programming: Top Down Approach. Datamation, December 1973, Vol. 19, No. 12.

Mills, H. Chief Programmer Teams: Principles and Procedures. IBM Corporation, Federal Systems Division, Gaithersburg, Md. 1971. FSC 71-5108.

_____ Mathematical Foundations for Structured Programming. IBM Corporation, Federal Systems Division, Gaithersburg, Md. 1972. FSC 72-6012.

_____ How to Write Correct Programs and Know It. IBM Corporation, Federal Systems Division, Gaithersburg, Md. 1973. FSC 73-5008.

_____ Top down Programming in Large Systems. Debugging Techniques in Large Systems. R.Rustin (ed). Courant Computer Science Symposium 1, New York University. Prentice Hall, Englewood Cliffs, N.J. 1971.

Myers, G. Composite Design: The Design of Modular Programs. IBM Corporation, Poughkeepsie, N.Y. 12601. 1973. TR 00.2406.

Naur, P. and B. Randell (eds) Software Engineering. Report on a Conference Sponsored by NATO Science Committee. Garmisch, Germany. Scientific Affairs Division, NATO, Brussels 39, Belgium.

Parnas, D. A Technique for Software Module Specification With Example. Comm. ACM, May 1972, Vol. 15, No. 5.

_____ On the Criteria to be Used in Decomposing Systems Into Modules. Comm. ACM, December 1972, Vol. 15, No. 12.

Parnas, D. Some Conclusions from an Experiment in Software Engineering Techniques. Proceedings AFIPS 1972, Vol. 41.

Weinberg, G. The Psychology of Computer Programming. Van Nostrand Reinhold Company, New York, N.Y. 1972.