Chief Programmer Teams

There is a myth these days that programming consists of a little strategic thinking at the top (program design), and a lot of coding at the bottom. But one small statistic is sufficient to explode that myth.

Including all overhead, five to ten debugged instructions are coded per man-day on a large production programming project. The coding time for these instructions cannot exceed more than a few minutes of an eight-hour day. What do programmers do with their remaining time? They debug.

Programmers usually spend more time debugging code than they do writing it. They are also apt to spend even more time reworking code (and then debugging that code) due to faulty logic or faulty communication with other programmers. In short, it is the thinking errors, more than the coding errors, which limit programming productivity.

The problem is as much one of organization as of technology. To address this, IBM has developed a programming organization called a chief programmer team.

A chief programmer team represents a new managerial approach to production programming. While the approach is made possible by recent technical advances in programming, it also incorporates a fundamental change in managerial framework which includes restructuring the work of programming into specialized jobs, defining relationships among specialists, developing
new tools to permit these specialists to interface effectively with a developing, visible project; and providing for training and career development of personnel within these specialties.

This approach contrasts sharply with that of conventional programming groups which frequently suffer from lack of functional separation, discipline, and teamwork. By moving the programming production process from private art to public practice, chief programmer team operations substantially improve the manageability, quality, and productivity of programming.

In addition to the organizational approach, chief programmer team operations are based on two major innovative disciplines. The first is provided by a development support library (DSL) in which all programs under development are maintained by a programming secretary in a visible, standardized form. The second discipline, introduced in a practical way by IBM, is structured programming (SP), which defines a top-down sequence for program unit creation and testing and a technical standard for the coding of each unit.*

Chief programmer team operations provide increased productivity by sharply reducing the debugging and reworking required in a project. The initial coding requires the same amount of time, but the design level thinking is transmitted deeper into the coding by technical and organizational means. SP displays program organization and interactions more effectively for the coding process. More competent, but fewer, people do the coding with carefully orchestrated teamwork. The result is increased productivity, and even more significant, improvements in the reliability and maintainability of the code produced.

This is accomplished by dividing the work of a programming project among special skills addressed to each type of work, rather than simply parceling out a project among programmer generalists, with all the attendant problems of communication and integration. Recognizing that program design capability is a scarce commodity, the work is organized around a senior architect/programmer. This key programmer operates in a disciplined team environment rather than as an individual. There are checks and balances in the restructuring to ensure the integrity of the team effort.

The nucleus of a chief programmer team consists of a chief programmer, a backup programmer, and a programming secretary. This nucleus is standardized to provide management continuity, not only for programming expertise but also for project recording and documentation. Requirements for additional personnel are defined by the chief programmer; a typical team will involve three to five programmers, a secretary, and other specialists. In addition, a project officer may be part of a team to help the chief programmer with administrative,

*See articles by Donaldson, and Miller and Lindamood in this book.
financial, legal, and personnel matters, thus allowing him to concentrate on technical management.

The chief programmer team allows for professional growth and technical excellence in programming. Since delegated clerical procedures are used to maintain programming system development in a structured form, more time and energy can be allocated to developing key technical skills and building the deliverable system. This creative environment provides good training for other programmers associated with a team and prepares them for future team leadership.

Team member responsibilities

The chief programmer is a technical manager to whom all team members report directly, but whose principal job is to design and code programs. The chief programmer codes central, critical segments of a programming system and specifies programs for other team members to complete using SP techniques. The programs done by others are reviewed and incorporated into the developing system under the immediate supervision of the chief programmer.

The chief programmer is a professional programming manager who maintains organization discipline and bears project responsibility. His managerial duties are simplified by the structure and the continuous project interaction of the team.

Project management exposure is reduced by the use of a backup programmer, so that a second person is totally familiar with the developing project and its rationale. The backup programmer, a peer of the chief programmer in program design and development, is involved in every aspect of the work and participates in making all important decisions. He can assume the leadership role at any time, if required.

He also participates in the system design and in the coding of the key parts of the system under the direction of the chief programmer. In addition, the backup programmer serves as a research assistant for the chief programmer in programming strategy and tactics, allowing the chief programmer to concentrate on the central problems of system development. Finally, he can provide test planning for the system independent of the chief programmer.

The job of programming secretary is standard in every chief programmer team, and is independent of the subject matter of the project. A programming secretary maintains the records of a project in the development support library in both an internal (machine-readable) and an external (human-readable) form.

The external project records of a chief programmer team are maintained in a set of filed listings that define the current status and previous history of the project. Current status is maintained in loose-leaf notebooks, each headed by a directory and followed by an alphabetized list of member modules. When
members and directories are updated and replaced in the status notebooks, the replaced copies are logged in chronological journals. All results of test runs are also maintained in journals.

The main function of a programming secretary is to maintain this current status of program and test data so that programmers can work more effectively and with fewer errors. A by-product of this function is a significant saving in clerical work on the part of the programmers.

In addition to maintaining the DSL, a programming secretary performs secretarial duties in maintaining all other project records. The workload balances well for one secretary on a team. In the middle of a project, DSL maintenance predominates; at the beginning and end, design and documentation create a great deal of paperwork.

It is significant in chief programmer team operations that the programming secretary is a full-fledged, professional team member, not simply a pooled assistant to the programmers on the team.

The reintroduction of senior people such as the chief and backup programmers into detailed program coding recognizes a new set of circumstances in comprehensive modern operating systems. The job control language (JCL), data management and utility facilities, and high-level source languages are so complex that there is both a need and an opportunity for using senior personnel at the detailed coding level.

The need is to make the best possible use of a very extensive and complicated set of facilities. The functions of such systems are impressive, but they are called into play by language forms that require much study, experience, and sustained mental effort to use effectively.

The opportunity also exists for a good deal of work reduction and simplification in the application being written, both in original programming and later in maintenance. For example, the intelligent use of a high-level data management capability may eliminate the need to develop a private file processing system. Finding such an intelligent use is not an easy job, but it can bring about substantial reduction in code required and easier system maintenance.

**Development support library**

The DSL is a system of office and machine procedures that permits the isolation and delegation of secretarial, clerical, keypunching, and machine operations in programming systems development. The office procedures create input for the machine procedures from programmer-generated material, and file output in project notebooks and archives of the external library. The machine procedures maintain and process library data on a disc file in the internal library, including procedures for performing all runs from initial source code entry through final system testing.
Programmers create or alter the project status by writing programs or data on coding sheets, by marking corrections in the status notebooks, and by requesting runs. The programming secretary is responsible for the preparation and execution of all runs and the filing of output. Fig. 1 illustrates the relationship between the people, the DSL, and the procedures. Because of this functional breakup of work, each programmer can work on more coding in parallel than is normally expected.

The DSL represents a concept in which people work on a common product rather than on separate, isolated products. Chief programmer team members communicate through this visible product. While the programming secretary is responsible for maintaining the notebooks and archives of the DSL, the chief programmer is responsible for its contents. This structure of responsibility permits a new level of management standardization in project record keeping.
The DSL permits a chief programmer to exercise a wider span of detailed control over the programming, resulting in fewer programmers doing the same job. This reduces communication requirements and allows still more control in the programming. With structured programming, this span of detailed control over code can be greatly expanded beyond present practice; the DSL plays a crucial role in this expansion.

As noted, the chief programmer team concept is primarily an organizational method of increasing programmer productivity. Several components of the method have been tried before. While the chief programmer team bears a superficial similarity to a close-knit programming team working under a lead programmer, two innovations distinguish it from such situations. First is the functional organization and disciplined approach used in the DSL operations. Second is the introduction of structured programming, which results in a new order of quality, productivity and understandability.

In chief programmer team operations, the traditional ad hoc mystique of a developing program is reduced. The visibility of the DSL motivates each team member to think more accurately and consistently about his specific job.

IBM has introduced a set of standards which enables structured programming techniques to be applied to production programming. These standards permit the chief programmer to read, understand, and validate all program data developed by other programmers on the team; this motivates better programming. The other programmers, in turn, read and understand programs written by the chief programmer that define the program stubs with which they must interface. While this organization results in the benefits of "egoless programming," as described by Weinberg, it goes further in ensuring that at least two programmers fully understand every line of the developing program.

The separation of skills forces a high degree of public practice. For example, the programming secretary is responsible for picking up all computer output, good or bad, and filing it in the notebooks and archives of the DSL where they become part of the public record. By contrast, in traditional programming operations, the bad runs go into the wastebasket, often destroying information of latent value, and certainly destroying information about errors of carelessness or ignorance. The identification of all computer runs and program data as public assets, not private property, is a key principle in chief programmer team operations.

Group of teams

About 100,000 lines of source code appears to be a practical maximum for a single team. Larger systems will require the extension of the organization to a group of teams.
In an approach now being tried, overall system design and development of key control code are being carried out by a single team of skilled programmer/analyst/managers under overall chief and backup programmers. When the core system is operational, some members of the original team will become chief programmers of subordinate teams developing major functional subsystems. The nucleus of the original team will complete the control coding and then become the technical monitor of the developing functional coding. That team will control all specification and design changes, and integrate the subsystems into the overall system as they evolve in a top-down fashion. As the system nears completion, programmers on the lower level teams may proceed to other assignments. Because of the original team’s detailed familiarity with the entire system and the use of the tools described above, it will supervise testing and turnover of the system.

Although the techniques have been described above in the context of a chief programmer team, they need not all be applied to realize substantial benefits. Fig. 2 illustrates the relationship of the individual ideas described.

DSL’s are the foundation for the entire method. They provide visibility of the developing programs and the basis for a more functional breakup of the programming process. Structured programming at the individual module level may be applied at any point in the development of a system, even during its operation as modules are rewritten to add new functions. SP requires a DSL to provide effective support for the hierarchical organization and inclusion of code as a module evolves.

When a complete new program system is begun, SP can be applied in a more extensive form (“top-down development”) to the development sequence of the entire system. It, too, requires a DSL and presupposes SP at the module level.

A chief programmer team is designed to make the most effective use of the three programming techniques. If one applies the techniques rigorously to development of a moderately sized system, it is hard to avoid creating an organization similar if not identical to that of a team.

Figure 2. Techniques and their relationship.
To institute the techniques in an existing organization, it is most practical to develop and install a DSL and to begin applying SP to new modules being written. As SP experience is gained and the support tools become familiar, an entire new program system can be developed. At this time, a chief programmer team nucleus can be established using two experienced programmers and a programming secretary familiar with SP. As the system evolves, additional programmers can be added to complete its development.

The information bank of The New York Times was produced under contract by a chief programmer team which specified and designed the system and developed over 83,000 lines of original high-level language source code. The task took 22 months. By today's productivity standards for systems of comparable complexity, such a task would require several times the 11 man-years of effort actually used by the team.

The information bank was developed using SP so that no integration period was required between the completion of detailed coding and delivery for acceptance testing. In other words, the integration work was completed parallel with, rather than after, unit coding. As a result of the high-precision coding techniques, the acceptance testing and subsequent system operation have been nearly error-free. For example, the file processing system (delivered one week after unit coding was completed) passed a week of acceptance tests without error, and ran 20 months until the first error was detected. In the first 13 months of operation of the on-line retrieval system, only one program error was detected that resulted in system failure. The chief and backup programmers produced code that had one detected error per man-year of effort.

Future implications

The New York Times' information bank project was a forerunner of many other internal and customer projects. These results show the possibility of a new level of manageability in programming projects through a combination of technical and organization standards. The results also show a significant decrease in error incidence and a corresponding increase in productivity, all with greater job satisfaction and less trauma in project completion.

There is a third property resulting from chief programmer team operations — harder to measure than manageability and quality, but even more important: the integrity and comprehensibility of the product for maintenance and growth. This occurs because an entirely new technical standard for design quality is enforced in structured programming systems.

At a more specific personal level, achievable targets for applications programming (as opposed to system programming — i.e., for system control programs) are 10,000 lines of source code and one error per man-year. This target includes system and program design, documentation and testing time as well as
actual coding time. These targets were achieved by the principal programmers on the chief programmer team performing on the information bank project.

While *The Times* project was still under way, it became apparent that the techniques and organization were effective. Within the IBM Federal Systems Division, programmers and programming managers were trained in structured programming. Similar courses have been given in other IBM divisions, and management techniques have been developed for SP projects.

A number of projects have since begun using SP techniques, and chief programmer teams are active in many of them. Several have already been successfully completed. One of the largest was the mission simulation system used in preparation and training for Skylab operations. The software for this totalled about 400,000 lines of source code produced over a two-year period using SP techniques. Productivity was again significantly higher than that previously experienced in comparable efforts.

More remarkably, the software was delivered on the original schedule in spite of 1,200 formal changes in the requirements, coupled with cuts in manpower and computer budgets. One of the striking facts about this development was that the rate at which computer time was used remained nearly constant from the 9th to the 24th month, a consequence of the continuous integration performed as part of the top-down development process. There was no overtime peak at the end of the project. Similar results are being achieved in other projects at IBM, both for products and for internal systems.

"... As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem; now that we have gigantic computers, programming has become an equally gigantic problem." E.W. Dijkstra, in the 1972 Turing Award Lecture, has articulated the problem. The problem is as much one of organization as of technology, and the chief programmer team is primarily an organizational solution.

Further application and extension of the concepts could move the programming process a long way toward a true professional discipline with a recognized, standard methodology.
Bibliography


