The Effect of Programming Team Structures on Programming Tasks

Marilyn Mantei
The University of Michigan

SUMMARY: The literature recognizes two group structures for managing programming projects: Baker's chief programmer team and Weinberg's egoless team. Although each structure's success in project management can be demonstrated, this success is clearly dependent on the type of programming task undertaken. Here, for the purposes of comparison, a third project organization which lies between the other two in its communication patterns and dissemination of decision-making authority is presented. Recommendations are given for selecting one of the three team organizations depending on the task to be performed.

1. Introduction

Two philosophies for organizing programming teams have achieved a moderate amount of popularity, if not utilization, in the data processing field. These are the egoless programming team proposed by Weinberg [28] and the chief programmer team proposed by Mills [18] and implemented by Baker [1]. In Weinberg's structure, the decision-making authority is diffused throughout project membership; in Baker's team, it belongs to the chief programmer. Communication exchanges are decentralized in Weinberg's team and centralized in the chief programmer organization. Neither structure is totally decentralized, democratic, centralized, or autocratic, but both Weinberg and Baker present arguments on why their methods will lead to superior project performance. Baker's project succeeds with a specific, difficult, and highly structured task. Weinberg's recommendations have no specific task in mind.

Research conducted in small group dynamics [7, 23, 27] suggests that a decision to use either team structure is not clear-cut and that there are strong task dependencies associated with each group's performance. The next two sections analyze Weinberg and Baker's organizations. In Section 4, a third, commonly encountered team organization is presented for the purposes of comparison. The fifth section conducts this comparison, recommending which of the three structures should be selected for a given property of a programming task.

Key words and phrases: chief programmer team, project management, software engineering, group dynamics, programming team structures
CR Categories: 3.50, 4.6

Author's address: M. Mantei, Graduate School of Business Administration, The University of Michigan, Ann Arbor, MI 48109.

2. An Analysis of Weinberg's Team Structure

Weinberg is a promoter of the egoless programming concept. His teams are groups of ten or fewer

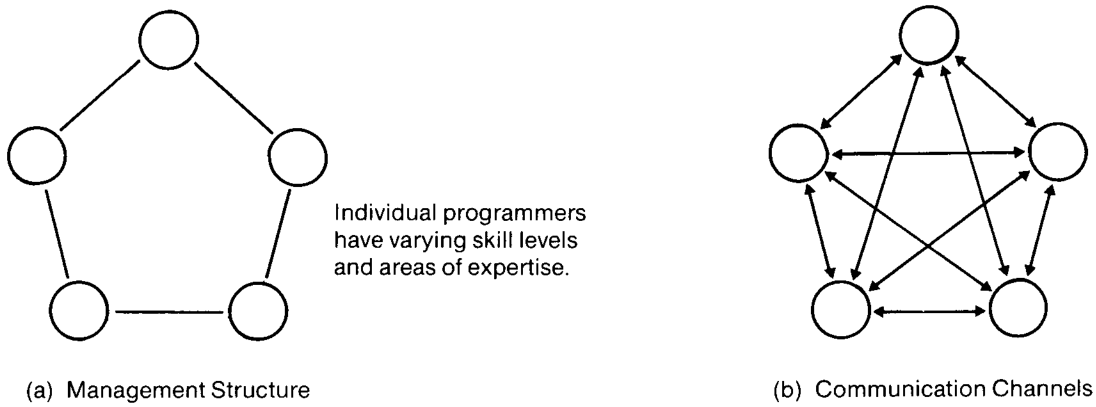

Fig. 1. Egoless Team Structure. Authority is dispersed and communication linkages decentralized.

programmers who exchange their code with other team members for error examination. In addition to code exchanges, goals are set by group consensus. Group leadership is a rotating function, becoming the responsibility of the individual with the abilities that are currently needed. Figure 1(a) illustrates the basic management structure of an egoless team; Figure 1(b) shows the communication exchanges that occur within this structure. The team proposed by Weinberg is acknowledged to be mythical in light of today's organization practices, but Weinberg feels that it is the appropriate organization for the best qualitative and quantitative code generation. Using the factors of amount of code produced, of time to produce code, and of error freeness to gauge programming performance, some task-related problems occur with Weinberg's team structure.

Bavelas [3] and Leavitt [14], in their experiments on centralized and decentralized group problem-solving behavior, found that decentralized groups take more time and generate twice as many communications as centralized groups. This suggests that a Weinberg group would function well in long-term continuing projects without time constraints (such as program maintenance). It would not, however, adequately perform a rush programming project.

A second weakness of Weinberg's proposal is the *risky shift phenomena* [5]. Groups engage in riskier behavior than individuals, both because of the dispersion of failure and the high value associated with risk taking in Western culture. In the case of a group programming team, decisions to attempt riskier solutions to a software problem or to establish high risk deadlines would be more easily made. In a software project with a tight deadline or a crucial customer, a group decision might cause the project to fail.

The democratic team structure works best when the problem is difficult. When the problem is simple, performance is better in an autocratic highly structured group [12]. Ironically, democratic groups attempt to become more autocratic as task difficulty increases. In the decentralized group, the additional communication which aided in solving the difficult problem is superfluous; it interferes with the simple problem solution. Tasks such as report generation and payroll programming fall into the category of simple tasks—for these, a Weinberg group is least efficient.

The decentralized group is lauded for its open communication channels. They allow the dissemination of programming information to all participants via informal channels. By virtue of code exchanges and open communication, Weinberg concludes that the product will be superior. March and Simon [16] point out that hierarchical structures are built to limit the flow of information, because of the human mind's limited processing capabilities. In the decentralized groups, as investigated by Bavelas, although twice as many communications were exchanged as in centralized groups, the groups often failed to finish their task. Similarly, individuals within a nonstructured programming group may be unable to organize project information effectively and many suffer from information overload. The structure and limited flow associated with hierarchical control may be assets to information assimilation.

Decentralized groups exhibit greater conformity than centralized groups [11]; they enforce a uniformity of behavior and punish deviations from the norm [20]. This is good if it results in quality documentation and coding practices, but it may hurt experimental software development or the production of novel ideas.

Despite the pressure to conform and an apparent lack of information organization, decentralized groups exhibit the greatest job satisfaction [23]. For long projects hurt by high turnover rates, job satisfaction is a major concern. Job satisfaction is also important for healthy relationships with the public or a customer—if indeed this is a necessary element of the programming project.

In summary, Weinberg's decentralized democratic group does not perform well in tasks with time constraints, simple solutions, large information exchange requirements, or unusual approaches. A difficult task of considerable duration which demands personal interaction with the customer is optimal for a Weinberg team.

3. An Analysis of Baker's Team Structure

Baker describes the use of a highly structured programming team to develop a complex on-line information retrieval system for the New York Times Data Bank; the team is a three-person unit. It consists of a *chief programmer,* who manages a *senior level programmer* and a *program librarian.* Additional programmers and analysts are added to the team on a temporary basis to meet specific project needs. Figure 2(a) illustrates the structure of the chief programmer team; the communication channels are shown in Figure 2(b).

The chief programmer manages all technical aspects of the project, reporting horizontally to a project manager who performs the administrative work. Program design and assignment are initiated at the top level of the team. Communication occurs through a programming library system, which contains up-to-date information on all code developed. The program librarian maintains the library and performs clerical support for the project. Rigid program standards are upheld by the chief programmer.

The Baker team is a centralized autocratic structure in which problem solutions and goal decisions are made at the top level. The task which the team undertakes is well-defined, but large and complex. Definite time constraints exist. Baker concludes that this compact highly structured team led to the successful completion of the project and that it has general applicability.

Several weaknesses exist in Baker's argument. Shaw [21] finds that a centralized communication network is more vulnerable to saturation at the top level. Information from all lower modes in this structure flows upward to the parent mode. Baker's team was intentionally small and worked with a highly structured system for managing project information; both these factors were critical to the success of the project. A third, equally important factor was the team leader's ability to handle project communication. This ability is closely related to the leader's software expertise. A less experienced leader or a more complex problem might have changed the project's success, even with staffing constraints and information management. Yourdon [29] points out that the effective chief programmer is a rare individual and indicates that most so-called chief programmer teams are headed by someone who is unlikely to adequately handle the communication complexity.

Centralized groups exhibit low morale [3]; this, in turn, leads to dissatisfaction and poor group cohesiveness. Members of highly cohesive groups communicate with each other to a greater extent than members of groups with low cohesion [15]. With a clearly defined problem that is split into distinct modules, this lack of communication will have little impact, but an ill-defined problem with many interfaces would suffer in a chief programmer team environment. The two software modules (the interface systems) on this project which might have served as indicators of this communication condition are, as a matter of fact, developed as a joint effort between the chief programmer and another team member.

Communication in a status hierarchy tends to be directed upward; its content is more positive than that of any communication directed downward [27]. In a tricky, difficult programming task, this favorable one-way flow of communication denies the group leader access to a better solution or, at least, an indication of problems in the current solution. Decentralized groups generate more and better solutions to problems than individuals working alone [25]—such as a chief programmer. The major basis for the success

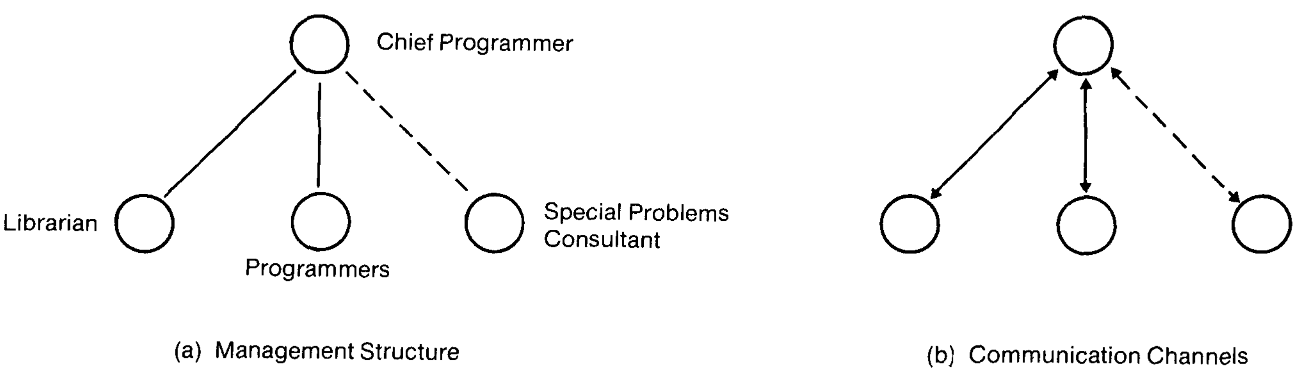

Fig. 2. Chief Programmer Team Structure. Authority is vested in the chief programmer and communication is centralized to this individual.

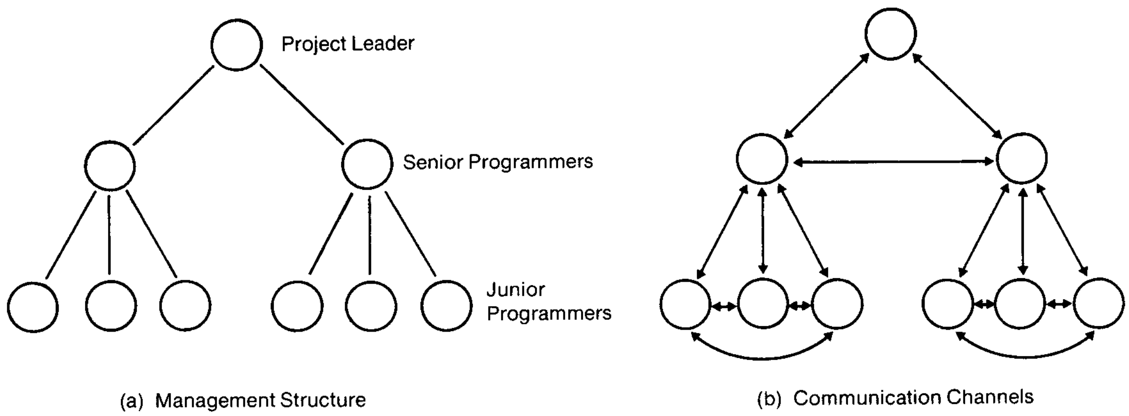

Fig. 3. Controlled Decentralized Team Structure. Authority is vested in the project leader and senior programmers, but communication at each level of the management hierarchy is decentralized.

of the New York Times Data Bank project was the team's ability to meet the delivery date. A centralized structure completes tasks more quickly than any decentralized form of control [14], but perhaps a more creative solution might have resulted from a different approach. Proponents of good software management stress concern for the software life cycle [8, 9, 13]. This implies that consideration be given not only to project completion schedules but to the software's usability, cost to the customer, and modifiability.

In summary, communication exists to a much lesser degree in centralized groups and is directed toward the manager. Both difficult tasks requiring multiple inputs for solution and unstructured tasks requiring substantial cooperation fare poorly in this kind of communication environment. Group morale and, thus, goal motivation are low in such a hierarchical structure. A simple, well-structured programming task with rigid completion deadlines and little individual interface with the client is perfect for the chief programmer team.

4. An Analysis of a Controlled Decentralized Team Structure

In practice, programming team structures vary considerably. Most take on some form of organization that draws from both Weinberg's egoless team and Baker's chief programmer team. A third, frequently used organization which we will call the controlled decentralized (CD) team is described in this section.

The controlled decentralized team has a project leader who governs a group of senior programmers. Each senior programmer, in turn, manages a group of junior programmers. Figure 3(a) illustrates the organization of this group; Figure 3(b) indicates the flow of communication that takes place in this type of group structure.

Metzger [17] describes this organization as a reasonable management approach. He makes two recommendations: First, he suggests that intermediate levels of management are preferable to requiring all senior programmers to report to the project leader and, second, he recommends that the programming groups be partitioned not according to code module assigned, but in terms of the type of role played in the project, e.g., test, maintenance, etc. Shneiderman [24] lists this structure as the most probable type of project organization. Like Yourdon [29], he suggests that the individual subgroups in the project participate in structured walkthroughs and code exchanges in the manner of Weinberg's egoless teams.

The CD team possesses control over the goal selection and decision-making aspects of the Baker team and the decentralized communication aspects of the Weinberg team. Setting project goals and dividing work among the groups are the tasks of the project leader. More detailed control over the project's functions is assigned to the senior programmers. Within each programming subgroup, the organization is decentralized. Problem solving is a group activity as is checking for code errors. Each group leader serves as the sole recipient or gatekeeper of project information for the subgroup and acts as a liaison with the leaders of the other groups. The communication and control problems of the egoless and chief programmer teams do not disappear in a CD structure but occur in the subgroups of the controlled decentralized team that correspond to the Weinberg and Baker teams: Thus, the properties of the subtask allocated to any of the subgroups interact, in a similar fashion, with the subgroup structure.

The decentralized subgroups of the CD team work poorly with highly structured or simple tasks. Group solutions are best directed at difficult problems. Much of the creative and difficult part of programming is planning the design and partitioning the work. In the CD structure this work is completed by the project leader. The senior programmers then take on their portion of the task and develop a group solution. Ironically, when the task is most difficult, the team structure is least effective. A poll of programming managers and academics indicated that the area they believed needed the most attention in software engineering was the planning and design stage [26], the work carried out by the CD team project leader.

With small problems, the CD team is unnecessary since its very structure presumes the existence of a larger project. As Brooks [6] points out, even though adding individuals to a project increases the communication problems and, thus, the effectiveness of the project's members, it is still necessary to have large teams for those programming tasks which are so large they could not be accomplished in a reasonable length of time by a few programmers.

Although control over projects is exercised from above, the group problem-solving approach at lower levels will take longer, and projects will be more likely to fall behind in meeting deadlines. The structure of the CD team would tend to centralize the egoless programming subgroups. Because of the senior programmer's gatekeeper role, he or she would emerge as an informal leader in group sessions. This, in turn, would lower individual satisfaction with the project and generate the ensuing problems of a high job turnover rate and group socialization difficulties. Because of this strong tendency toward centralization, shorter projects are best for the CD structure.

A controlled decentralized team is an effective error-purge mechanism. The code walkthroughs and group input at the code generation level will filter out many errors. Code generated in this fashion is more reliable than code coming from a chief programmer team operation.

Programming tasks that are not easily subdivided suffer in a CD team. Note in Figure 3(b) that communication between groups occurs at the senior programmer level. Projects requiring micro-decision communication about code interfaces cannot expect this communication to be conveyed effectively through a liaison person functioning at a macro level in the project.

In summary, the controlled decentralized team will work best for large projects which are reasonably straightforward and short-lived. Such teams can be expected to produce highly reliable code but not necessarily on time or in a friendly manner. They are ill-suited for long-term researchlike projects.

Team Structure and Programming Task Relationships

This section describes seven salient properties of programming tasks and compares the performance of each team structure discussed in relationship to these task properties. The relevant properties are:

(1) *Difficulty*. The program required to solve the problem can be complex, consisting of many decision points and data interfaces, or it may be a simple decision tree. Distributed processing systems and projects with severe core or rapid response time constraints fall into the *difficult* category. Much of the scientific programming would come under the *simple* category heading.

(2) *Size*. Programs may range from ten to hundreds of thousands of lines of code for any given project.

(3) *Duration*. The lifetime of the programming team varies. Maintenance teams have a long lifetime; one-shot project teams have a short lifetime.

(4) *Modularity*. If a task can be completely compartmentalized into subtasks, it is highly modular. Most programming problems can be split into subtasks, but the amount of communication required between the subtasks determines their modularity rating. A tape system for payroll reports is a highly modular task. A data management system for the same purpose has a low degree of modularity.

(5) *Reliability*. Some tasks such as patient monitoring systems have severe failure penalties, while other tasks, such as natural language processing experiments, need not be as reliable, although working programs are always desirable. The reliability measure depends on the social, financial, and psychological requirements of the task.

(6) *Time*. How much time is required for task completion? Is the time adequate or is there time pressure? The penalty for not meeting a deadline strongly affects this measure.

(7) *Sociability*. Some programming tasks require considerable communication with the user or with other technical personnel, such as engineers or mathematicians, while other tasks involve interaction with the team alone. Computer center consulting groups that develop user aids have higher sociability requirements than groups programming their own set of software tools.

Throughout this paper, the labels egoless programming team and chief programmer team have prevailed. For the purposes of comparison, these terms have been changed to names reflecting the decision-making authority and communication structure of the teams. The three teams are:

1. Democratic Decentralized (DD). This group is like Weinberg's proposed team; it has no leaders, but appoints task coordinators for short durations. Decisions on problem solutions and goal direction are made by group consensus. Communication among members is horizontal.

2. Controlled Decentralized (CD). The CD group has a leader who coordinates tasks. Secondary management positions exist below that of the leader. Problem solving remains a group activity but partitioning the problem among groups is a task of the leader. Communication is decentralized in the subgroups and centralized along the control hierarchy.

Table I. Recommended Team Structures for Programming Task Features.

	Programming Task Characteristics													
	Difficulty		Size		Duration		Modularity		Reliability		Time Required		Sociability	
Group Structures	High	Low	Large	Small	Short	Long	High	Low	High	Low	Strict	Lax	High	Low
Democratic Decentralized	X			X		X		X	X			X	X	
Controlled Decentralized		X	X		X		X		X			X		X
Controlled Centralized		X	X		X		X			X	X			X

3. Controlled Centralized (CC). This group is like Baker's team. Both problem solving and goal directions are generated by the team leader. Communication is vertical along the path of control.

The expected interaction of each of these team structures with the factors governing program tasks can be drawn from experimental research on small group dynamics. To assess performance quality, team structures are assumed to be evaluated on the quality of generated code and the time in which the code generation was completed.

Table I lists recommended group structures for each task variable. Under the category *task difficulty*, simple problems are best performed by a centralized structure which completes tasks faster. Decentralization works best for difficult problems. Groups are found to generate more and better solutions than individuals. Unfortunately, the CD team is centralized precisely where the problem is difficult. The DD team is the best solution for difficult problems. For simpler programming tasks, a CC or CD structure is recommended.

As programming tasks increase in size, the amount of cooperation required among group members increases. Group performance is negatively correlated with the cooperation requirements of a task. As tasks become *very large*, the DD group is no longer viable because of its cooperation requirements. CC and CD groups can be effectively regrouped into smaller structures to handle the task. When the task size requires a smaller number of programmers, the DD group performs better because of its high level of communication. For *very small* tasks, the CC group is best because it does not require the additional communication of democratic groups; but then, a group is unnecessary. An individual will do.

The duration of the task interacts with group morale. Short tasks may not require high group morale, whereas long tasks will suffer from high personnel turnover if morale is low. DD groups have high morale and high job satisfaction. This should be the preferred group structure for ongoing tasks. The CC and CD groups are effective for short-term tasks.

If task modularity is low, the DD group performs best because of its higher volume of communication. Cooperative (read DD) groups have higher orderliness scores than competitive (read CC) groups [10]. This orderliness is essential for maintaining the interfaces of a low modularity task. Nondirective leadership has been found to be most effective when a task has a high multiplicity of solutions. Directive leadership is best for tasks with low multiplicity solution choices [22]. A DD group can be characterized as having nondirective leadership, CC and CD groups as having directive leadership. High modularity tasks have a low multiplicity of solutions, and thus the CD and CC groups can be expected to exhibit the best performance given such tasks.

CC and CD groups perform well when confronted with high reliability requirement problems. Decentralized groups have been found to make less errors and produce better solutions to problems. A CC group is more error-prone and probably should never be used for projects in which relatively simple errors can result in disaster.

A decentralized group takes longer to complete a problem than a centralized group. If tasks have severe time constraints, a CC team is best. When time is not crucial, the low motivation of CC groups can interfere with task completion. Therefore, the more democratic groups are preferred, with the DD structure being the best choice.

If a task requires high sociability, the DD team structure is best. Groups learn faster than individuals (such as the team leaders of CC groups). Therefore, a DD group would understand a user's interface problem in a shorter period of time. DD groups are higher in social interaction and morale than CD or CC groups. These traits will enhance their social relationships with the task contacts.

6. Conclusion

Many programming task features interact with each other, e.g., a large project is often a difficult one. Group structures that are effective for one aspect of a task may be totally wrong for another. In selecting a team structure, it is important to use a decision-making algorithm to prioritize, weight, or combine the crucial task variables.

Little experimental work on programming team and task interaction has been carried out. Basili and Reiter [2] found relationships between the size of a programming group and several software metrics. They also found cost differential behavior arising from the software development approach taken, with structured techniques being notably cheaper. Only one programming task was performed by the experimental groups. Weinberg's suggestions on group organization are anecdotal and Baker's conclusions are confounded by the team personnel and the programming methods selected.

Most of the research on group problem-solving behavior was conducted in a laboratory setting with students and tasks of short duration. A problem exists in trying to apply these conclusions to the external work environment. In particular, programming tasks generally involve an entirely different time span than laboratory experiments. Becker [4] scathingly criticizes these "cage" experiments. Rogers [19] suggests substituting network analysis field work to understand the effects of group structures.

None of these task/structure recommendations have been tested in a software development environment. Despite all these shortcomings, the application of a body of research on group dynamics to the organization of personnel on a programming project is a step forward from the hit-and-miss guessing that is the current state of the art.

References

1. Baker, F.T. Chief programmer team management of production programming. *IBM Syst. J. 1* (1972), 57–73. Baker presents a case history of a program project management organization, the chief programmer team. This compact management strategy coupled with top-down program development methods achieves above average success in terms of productivity and error-free code.

2. Basili, V.R., and Reiter, R.W., Jr. The investigation of human factors in software development. *Comptr. 12*, 12 (Dec. 1979), 21–38. This paper examines the impact of a programming team's size and program development approach, disciplined or ad hoc, on the software product. The disciplined method resulted in major savings in development efficiency and smaller groups built larger code modules.

3. Bavelas, A. Communication patterns in task-oriented groups. *J. Acoustical Soc. America 22* (1950), 725–730. Bavelas describes an experiment in which the communication structures of a circle, wheel, and chain were imposed on small groups by the physical arrangement of cubicles and message slots. Each structure was then measured for its problem-solving efficiency.

4. Becker, H. Vitalizing sociological theory. *Amer. Sociological Rev. 19* (1954), 377–388. Becker refers to the small group laboratory studies as "cage studies" and recommends their use by sociological theorists only for an awareness of such studies' limiting conditions.

5. Bem, D.J., Wallace, M.A., and Kogen, N. Group decision making under risk of adversive consequences. *J. Personality and Social Psychol. 1* (1965), 453–460. This paper demonstrates, in a context of adversive consequences (loss of money, induced nausea, etc.), that unanimous group decisions concerning matters of risk shift toward greater risk-taking than individual decisions. Moreover, the authors provide evidence that the underlying process for the risky shift is a diffusion of the responsibility among group members.

6. Brooks, F.P., Jr. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, Reading, Mass., 1975. This work is a lyrical, enjoyable, and sage discussion of the problems and pitfalls that beset a mammoth software project—developing the IBM 360 operating system.

7. Cartwright, D., and Zander, D., Eds. *Group Dynamics: Research and Theory.* 3rd edition, Harper and Row, N.Y., 1968. This serves as an excellent compendium of the spurt of group dynamics research activity in the late 1950s which laid the groundwork for what we know about group behavior today.

8. Cave, W.C., and Salisbury, A.B. Controlling the software life cycle—The project management task. *IEEE Trans. Soft. Engr. SE-4*, 4 (July 1978), 326–334. This paper describes project management methods for controlling the life cycle of large software systems distributed to multiple users. It emphasizes responding to user satisfaction and user requirements and suggests methods to establish and maintain control in an extended dynamic environment.

9. De Roze, B.C., and Nyman, T.H. The software life cycle—A management and technological challenge in the department of defense. *IEEE Trans. Soft. Engr. SE-4*, 4 (July 1978), 309–318. De Roze and Nyman describe the software life cycle management policy and practices that have been established by the Department of Defense for improving the software development process.

10. Deutsch, M. The effects of cooperation and competition upon group process. *Human Relations 2* (1949), 129–152, 199–231. Deutsch describes an experiment which establishes two forms of group relationships, cooperative and competitive. Besides better communication, increased orderliness and higher productivity result when the cooperative group relationship exists.

11. Goldberg, S.C. Influence and leadership as a function of group structure. *J. Abnormal and Social Psychol. 51* (1955), 119–122. The experiment described in this paper compares group influence on group members in three organization structures: a star, a fork, and a chain. Individuals holding central positions were influenced less than other group members.

12. Guetzkow, H., and Simon, H.A. The impact of certain communication nets upon organization and performance in task-oriented groups. *Mgmt. Sci. 1* (1955), 233–250. The authors establish three communication structures: all-channel, wheel, and circle; they then examine their effect on solving a relatively simple communication problem. The restrictions of the wheel organization aided the solution process, whereas those of the circle hindered it. The lack of restrictions in the all-channel case also hurt the solution process.

13. Jensen, R.W., and Tonies, C.C., Eds. *Software Engineering.* Prentice-Hall, Englewood Cliffs, N.J., 1979. Here, several breakdowns of what constitutes a software life cycle are presented. The authors indicate that if the customer-use phase is included in this breakdown, the time spent on the code development constitutes a relatively small portion of the project.

14. Leavitt, H.J. Some effects of certain communication patterns on group performance. *J. Abnormal and Social Psychol. 46* (1951), 38–50. Leavitt compares problem-solving effectiveness in both wheel and circle communication structures. The wheel structure was faster but the circle structure accounted for fewer errors.

15. Lott, A.J., and Lott, B.E. Group cohesiveness, communication level, and conformity. *J. Abnormal and Social Psychol. 62* (1961), 408–412. This paper describes an experiment in which groups were scored on cohesiveness and then tallied for the amount of communication generated in a discussion session. Highly cohesive groups communicated more.

16. March, J.G., and Simon, H.A. *Organizations.* Wiley, New York, 1958. March and Simon focus on the members of formal organizations as rational men. From this, they point out that the basic features of organizational structure and function derive from characteristics of the human problem-solving process and rational choice.

17. Metzger, P.W. *Managing a Programming Project.* Prentice-Hall, Englewood Cliffs, N.J., 1973. Metzger suggests a project organization constrained in terms of the types of tasks that are undertaken in the development of a software system. He goes on to describe how these tasks should be managed via this hierarchical arrangement.

18. Mills, H.D. Chief programmer teams: Principles and procedures. IBM Rep. FSC 71-5108, IBM Fed. Syst. Div., Gaithersburg, Md., 1971. Mills suggests that the large team approach to programming projects could eventually be replaced by smaller, tightly organized and functionally specialized teams led by a chief programmer.

19. Rogers, E.M., and Agarwala-Rogers, R. *Communication in Organizations.* Free Press, N.Y., 1976. The basic research on group structures in small group network communication is summarized and critiqued in a thoroughly readable manner.

20. Schachter, S. Deviation, rejection and communication. *J. Abnormal and Social Psy-*

chol. 46 (1951), 190–207. This article describes an experiment in which three group members were paid to respectively 1) deviate from, 2) follow, and 3) change over to the group position taken on an issue. Groups with high cohesiveness scores produced greater rejection only of the deviant individual.

21. Shaw, M.E. Some effects of unequal distribution of information upon group performance in various communication nets. *J. Abnormal and Social Psychol. 49* (1954), 547–553. In this paper, the amount of independence and, thus, individual satisfaction are examined in various group structures. Low centralization in groups led to member satisfaction.

22. Shaw, M.E., and Blum, J.M. Effects of leadership styles upon performance as a function of task structure. *J. Personality and Social Psychol. 3* (1966), 238–242. Shaw and Blum describe an experiment in which they manipulated the leadership of two groups to be nondirective or directive. Given three tasks of varying solution multiplicity, directive leadership performed best with low multiplicity tasks.

23. Shaw, M.E. *Group Dynamics: The Psychology of Small Group Behavior.* McGraw-Hill, N.Y., 1971.

24. Shneiderman, B. *Software Psychology.* Winthrop, Cambridge, Mass., 1980. Shneiderman discusses the good and bad points of the Weinberg and Baker teams and a third conventional team. He notes that an egoless team may be difficult to maintain and a competent chief programmer hard to find, concluding that the currently existing conventional organization has strong chances for successful projects—especially with a competent manager.

25. Taylor, D.W., and Faust, W.L. Twenty questions: Efficiency of problem solving as a function of the size of the group. *J. Experimental Psychol. 44* (1952), 360–363. Taylor compares individual problem-solving to group problem-solving in a game of 20 questions. Even after several days of practice, groups of two and four individuals asked less questions to discover an answer than sole participants.

26. Thayer, R.H., Pyster, A., and Wood, R.C. The challenge of software engineering project management. *Comptr. 13,* 8 (Aug. 1980), 51–59. The three authors report on a survey of software project management experts who were asked to indicate the most important issues facing software engineering. The structure of programming projects was rated as unimportant; planning received the highest ratings.

27. Thibaut, J.W., and Kelley, H.H. *The Social Psychology of Groups.* Wiley, N.Y., 1959. The second section of this book presents a general theory for group formation and group dynamics—in particular, the status systems within groups, conformity requirements, group goal setting behaviors, and the roles played by individuals within the group. In all, not light reading for the nonsociologist.

28. Weinberg, G. *The Psychology of Computer Programming.* Van Nostrand Reinhold, N.Y., 1971. Weinberg provides homilies, advice, and some wisdom about the psychological considerations of the programming process. It is here that he suggests the egoless approach to programming and discusses its potential advantages—Weinberg is short on supportive research, but the book is fun to read.

29. Yourdon, E. *Managing the Structured Technique.* Prentice-Hall, Englewood Cliffs, N.J., 1976. Yourdon discusses the chief programmer team and Weinberg's egoless debugging techniques in a complete scenario for project management. He labels the chief programmer team impractical because of the dearth of true chief programmers.