Coordination as Linkage: The Case of Software Development Teams

Sara Kiesler, Douglas Wholey, and Kathleen M. Carley

This chapter examines coordination in software development teams as a practical context for talking about the linkages between individual and group productivity. We do not discuss individual-organizational nor group-organizational linkages, although many of the points we make pertain to those linkages as well.

Software development is a kind of technical work found in many organizations: the technical team project. Certain technical tasks transcend the ongoing functions of departments or the capabilities of individuals, and thus organizations create a project group or team to do the work. A software development project group can have two to several hundred members. Membership is typically diverse; the work may require the participation of programmers, software engineers, application experts, researchers, requirements analysts, software testers, documentation writers, project managers, customer support personnel, and perhaps others. Project members may be drawn from different locations and different departments and may even work on the project in different places. The projects have predictable stages but also experience unpredictable changes in the organizational and technical environment—changes in personnel, modifications in available software and hardware technology, changing client expectations, and new economic constraints (Brooks, 1987).

In software development, productivity depends on teamwork. Teamwork refers to work done as a team and to the attitudes, skills, and behaviors that subordinate personal prominence to the efficiency of the whole. Teamwork is crucial because every job and every stage is interdependent. High levels of individual productivity do not ensure success. Productivity depends on leveraging competencies through teamwork (Clark et al., 1987).

Coordination is the overt, behavioral instantiation (representation) of teamwork. That is, coordination is what people, technology, or organizations actually do to integrate team members and their work to form a group product. Measures of coordination include observations that different people and subunits working on a project agree to a common definition of what they are building, share information, hand off components of the work expeditiously, take responsibility for one another's performance, and mesh their activities. Coordination should be distinguished from exogenous forces—prices, monopoly position of the group, resources made available to the group, management priorities, and so forth—that affect group productivity directly rather than through linkages.

THE DOMAIN OF SOFTWARE DEVELOPMENT

Software development is a theoretically interesting context for examining linkages and it also has practical importance. The United States has more than 7,000 software firms; many other firms participate in the development of software systems (National Science Board, 1989). Business, education, government, and technical endeavors ranging from automated manufacturing to financial transactions to national defense require complex software systems. Most experts agree that the demand for software outstrips the ability of firms to produce it. Software systems are notoriously difficult to produce. Problems often force delays in the implementation of new applications, compromises in what those applications can do, and uncertainties about their reliability (National Research Council, 1990).

Coordination in Software Development

Simplified models of the software life cycle break its development into distinct phases. One such breakdown is that suggested by Davis (1987): (1) problem definition, (2) feasibility, (3) analysis, (4) system design, (5) detailed design, and (6) implementation and maintenance. A variety of tasks, each with its requisite skills, must be done during these different phases: analysis, design, coding, documentation, and testing. Analysis involves evaluating and translating organizational or individual needs into system capabilities. Design involves developing a set of distinct logical units, each of which can be developed and
tested separately; choosing software and hardware; structuring a database so as to minimize redundancy and improve ease of access; and so on. Coding means translating the design specifications into executable instructions that run reliably and efficiently on particular hardware. Documentation involves coordinating and maintaining consistency of the human-computer interface, writing manuals and specifications, and preparing the internal code description, as well as recording the rationale behind design and coding decisions. These tasks are highly interactive in that changes in requirements often require changes in design, code, and documentation. Design decisions often feed back to change or limit the capabilities that the system can offer. Changes in the hardware and software, or changes in a company's financial status, may force the team to return to the design phase. This process is iterative in that software systems must be enhanced and changed as the environments in which they exist change and as people put them to new uses.

Achieving a successful software system requires coordination among the various phases and tasks involved in the software development cycle and minimal backtracking. If the software system is small, and members are physically proximate and respect one another, effective coordination can occur because the group can work out problems together and keep all the implementation details in focus. This focus on sharing ideas through direct communication is what traditionally has been meant by teamwork; it is the main emphasis of cooperative team learning in high school and college classrooms (e.g., Bossert, 1988-1989). In many cases of modern technical work, however, this simple model of coordination is impossible. Kraut and Streeter (1990) discuss three reasons why this is so—project complexity, uncertainty, and interdependence.

Complexity

A fundamental characteristic of many software tasks is that they are too big for any one or two skilled programmers to undertake alone. Moreover, a single complex skill like programming is not the only skill required in the software development process. Software development also requires analysis to determine what the software should do; evaluation of alternative platforms; design to shape the basic structure of the programs and their communication with other programs, data bases, and users; tests to ensure that code meets requirements and that users understand the interface; creation of special tools for implementation; hardware and software maintenance procedures; written documentation; and an administrative infrastructure to set priorities on requests for features and to handle feedback from users.

Uncertainty

Complexity per se does not invariably lead to difficulties in coordination. As Kraut and Streeter (1990) note, automotive factories, textile mills, and tuna canneries employ hundreds of people to produce their products, yet many run smoothly. Software development is different in that it is more uncertain. Manufacturing involves routines, doing the same thing repeatedly. But the software development process is nonroutine activity, and specifications for it invariably are incomplete. Incompleteness partly results from limited knowledge of the software development domain (Curtis et al., 1988). At many points the information that designers or programmers need to make decisions is not available to them, although others in the project may have the knowledge needed for those decisions.

Software development is also uncertain because specification of what a software system should do changes over time (Brooks, 1987; Curtis et al., 1988; Fox, 1982). Competition, regulations, standards, company politics, plans, and financial conditions can lead to changes in specifications. Also, it is often only by using software that purchasers understand its capabilities and limitations. As they use the software, they often demand new capabilities that they were not able to envision at the software's creation.

Uncertainty in software development may be reflected in disputes among different groups involved in its development (Curtis et al., 1988; Kraut and Streeter, 1990). People associated with different parts of a project can have different beliefs about what the software should do. For example, analysts translate users' needs into requirements for system capabilities. As a result, they often adopt the point of view of the software's purchasers. On the other hand, designers and programmers may have more of an insider's focus and emphasize ease of development and efficiency of operation. These differences in points of view must be resolved for the team to succeed.

Interdependence

Complexity and uncertainty in software work would be less of a problem if software did not require integration of its components to such a large extent. Software consists of hundreds or thousands of modules or components that must mesh with each other perfectly for the software system as a whole to operate correctly. One mistake in part of a system can have disastrous, unanticipated consequences (Travis, 1990). This required integration, combined with complexity and uncertainty, requires in turn special coordination techniques that may not be neces-
Coordination of Software Development Teams

Coordination Through Selection and Training

Coordination is to be combined into a joint product, sometimes must be invested in the combination process itself. Some successful small groups have taken time and effort. Social psychologists who study small groups have evolved a framework for team selection and training. The selection of team members and the training and preparation of the teams are important aspects of coordination. (Steiner, 1972)

In software teams, the top 10 percent of programmers are said to be more than four times as efficient as the bottom 15 percent (Boehm, 1982). These individual differences may have a significant effect on team performance. If a team is formed from highly skilled and experienced workers, achievement tasks such as training, job design, and management are made simpler. Members of such a team may also influence each other's work by simply working together. The team members, in turn, may influence team coordination, making the team more efficient. More importantly, the team may become a whole (McGrath, 1984). Individual competencies combined into a whole team, as a whole (McGrath, 1984), Individual competencies combined into a whole team. Under a competency multiplier process, teams made up of highly competent people perform better than the sum of the parts. More individual abilities are particularly important when the team members are complex, uncertain, and interdependent. Highly skilled and experienced team members can solve routine problems and teach those solutions to one another (Clark and Stephenson, 1986; Hill, 1982; Hine, 1980). These members contribute to the team's ability to solve nonroutine problems and contribute to the team's ability to solve nonroutine problems. For example, team interaction bestows extra benefits on team performance. (Termer et al., 1976; Trzeciak and Eden, 1989). Competency multiplier effects also appear to be seen over time because competent members become better at what they do already good at.

Research on Individual-Group Linkages

Much of the existing research on software development and other technical work does not deal with linkages. There has been considerable work on individuals' cognitive problems resulting from creation, understanding, and debugging problems resulting from creation, understanding, and debugging programs. (See Curtis, 1985; for a sample of this kind of research.) This approach ignores the linkage issues inherent in software development. Results from studies of individual's problems in engineering problems (Scott and Simmonds, 1979) and designs (Curtis, 1985) do not generalize simply to software problems. (See Curtis, 1985.) The deal with the same cognitive processes in software development and other engineering problems. (Scott and Simmonds, 1979; Curtis, 1985; Verstegui and Williams, 1989; Beatty, 1986.) Hence, much of the research on individual-group linkages is not adequate to the domain of software development. (See Curtis, 1985.)

Outside the domain of software development, there have been a number of studies of individual-group linkages. These studies have long been shown that individual behavior is not the same as group behavior. However, the studies have long been shown that group productivity usually does not equal the sum of individual productivity. At the least, if individual labor effort is measured in dollar terms. These estimates can be used to show the financial benefit of labor savings that would be achieved by introducing an improved method of doing one's job. Even a small improvement in productivity may yield substantial savings. (Boehm, 1982; Jones, 1986.) Aggregated team goals may exist, but have no metric because the work is unique (e.g., building a space platform). Time-based indices do not, of course, address the quality of work.
and, together, more uniquely able than other teams (March, 1981). (See Figure 9-1.) Competent teams gain more from technological interventions and tools that increase individual competency and intermember learning, which contributes to an increasing gap between excellent and poor teams. In this manner, selection and training to acquire the most competent team members become a linkage factor, especially over time.

A strategy that focuses exclusively on individual selection and training to achieve teamwork is often impractical and has a number of disadvantages. Organizations often are prevented from hiring only the best people. The best people may lead to higher labor costs than are necessary. Moreover, those whose high talents are hidden initially cannot be discovered if the organization tries to hire only those with excellent resumes. Finally, even a group of highly qualified individual workers, placed on a team, may function poorly as a team unless attention is given to their organization as a team.

### Coordination Through Team Design

Organization as a team, or team design, refers to the organizational structure and formal procedures that provide "built-in" solutions to coordination. These solutions may include task decomposition, lines of authority, centralization of control, and standard operating procedures, or they may include technologies to standardize or rationalize the work itself. Team design through structure and formalization is theoretically an efficient alternative to direct communication when tasks are complex, uncertain, and interdependent (Aldrich, 1979; Cyert and March, 1963; Downs, 1967; March and Simon, 1958; Simon, 1962). For instance, instead of having to talk repeatedly about what each person should do, formal task decomposition allows a group facing a complex task to divide its work into manageable chunks. It should not be surprising, therefore, to find that recent solutions to effecting teamwork in software development and other kinds of technical work have emphasized team design.

A major emphasis in team design has been the development of formal procedures governing communication at various stages of the work. For instance, formal meetings may be held at predetermined times in order to consider decisions about changes in the design. Brooks (1987), Curtis et al. (1988), and Fox (1982) noted that problems in accurately and completely communicating stable software requirements to members of a software project are among the most difficult to resolve in software development. Formalization is thought to increase control and regulate information flow. Written specifications or plans, documentation, and formal meetings ensure adherence to the plan and system as they evolve and that all the components fit together.

Formalizing project management can also help managers monitor teams’ work. Each phase of the work cycle, from planning through operation and maintenance, is supposed to have well-defined products

---

2Modern software practices (e.g., logical models, well-defined interfaces, modularity, layered architectures, hierarchical management, object orientation) can be considered team designs because they are meant to regulate the number of connections that people and software components have. Modularity and information hiding, hallmarks of object-oriented design and programming languages, are thought to promote independence in programming, ease adaptation, and minimize backtracking (Dietrich et al., 1988; Parnas, 1972; Rumbaugh, 1991). Object-oriented design and programming also directly incorporate team design principles through inheritance. In software engineering, computer-aided software engineering (CASE) tools have been developed to facilitate the development of logical models, coordinate project design through a shared data dictionary, and automate input/output analysis (Sodhi, 1991; Zarrella, 1990). The degree to which current CASE tools actually facilitate team coordination still is under contention (Spurr and Layzell, 1990). In a recent comparison of software maintenance teams that did and did not use CASE tools, groups using CASE tools were less productive (Banker et al., 1991; see also Orlikowski, 1988).
and milestones. Thus, it is specified in advance what will be delivered at each stage and how the deliverables will be tested or scrutinized to ensure that they do what they are supposed to do. In software development, all official project documents may be under change control. For example, there are usually naming conventions that must be adhered to project wide. Similarly, code cannot be written without design reviews; code cannot be tested before code walk-throughs; changes cannot be made without issuing a modification request; no piece of code goes to system test without an integration test; and so on.

Another important element in team design is the authority structure, which can be used to resolve disputes and inconsistencies across units. There is some evidence from an extensive comparison of automotive product development teams that significant variance in the authority structure contributes to the superior performance of Japanese automoblie design teams over their American and European counterparts. Japanese team managers had greater authority and independence than American and European managers did (Clark et al., 1987). A concomitant of this idea in software development is that a chief designer or architect is the one person in a complex project who has sufficient knowledge of both the application domain and the possible software architectures to integrate the two. Weinberg (1971) advocated the chief programmer role, in which a senior designer/programmer has control over a software project. Problems arise when the design is distributed in more than one head or, worse (and probably more typically), is not in anybody’s head. According to Curtis et al. (1988), skilled designers often assume responsibility for communicating their technical vision to other project members and for coordinating the work of the project.

In sum, team design (including group structure, formal procedures, and hierarchy) is advocated in teams to routinize the transfer of information and increase control and reliability. Formality and written documentation also are attempts to reconcile differences of opinion, help people understand their goals and those of others, induce the evaluation of alternatives, and develop agreements that all can accept. The effort expended by a small group writing a formal design document can be more than offset by the communication forgone later when each project member does not need to describe his or her vision separately to the scores of people who need the information. Formal procedures also reduce errors. Thus, for example, in software development one might run automated consistency checks on a formal specification document (cited in National Research Council, 1990) or even use a computer-based system that tracks modification requests to trigger management intervention when a project schedule slips.

Benefits obtained from team design do not come without costs, however. Formal structures and procedures can place an extra burden on development costs by increasing the need for a coordination infrastructure: training, increased clerical and management staff, and increased project reports and archives. Fox (1982) estimated that in large software projects, 50 percent of the cost is for planning, checking, scheduling, managing, and controlling. Tools and techniques that formalize communication or management require that time and effort be spent in teaching people to use them and ensuring that they do. Change-control systems are potential time wasters or distractions from work. Management sometimes uses standardization and rationalization of tasks to increase control, which can sap motivation and increase dependency on outside experts. Design also can impede innovation by limiting the options explored by a team. Finally, the “care and feeding” of bureaucracy can become more significant to employees than the ultimate goals they are supposed to accomplish.

A particularly serious disadvantage of team design as a coordination strategy is that it can depersonalize interaction. For instance, with task decomposition, team members, or subgroups of the team, have different roles. Team members or subgroups working on their own tasks tend to develop divergent perspectives and habits of work (e.g., Brewer and Kramer, 1985; Tajfel, 1982). They may have little opportunity or eagerness to learn from others on the team, which will impede the exchange of expertise and discovery (Burns and Stalker, 1961; Carley, 1990, 1991, 1992; Faunce, 1958; Festinger et al., 1950; Jablin et al., 1987; Monge and Kirste, 1980; Newcomb, 1961). Task decomposition can also exacerbate demographic or skill differences that existed at the start (Barnlund and Harland, 1963; Dearborn and Simon, 1958; Jablin, 1979; Monge et al., 1985; Sykes et al., 1976).

Whether team design through structure, formalization, or technology actually works as well as it is supposed to theoretically, remains debatable. Boehm’s (1987) analysis of software productivity indicates that productivity due to changes in team design increased by just 7 percent between 1981 and 1986. Card et al. (1987) reported that software engineering technology improved reliability 30 percent but had no impact on productivity. Chapter 2 reaches the same conclusions as Card et al. did in 1987.

Coordination Through Team Communication

Experience, organizational theory, and behavioral research suggest that team design does not by itself solve all coordination problems in teamwork. No matter how successfully task decomposition, authority
structures, or standard operating procedures reduce the number of interfaces between team members, different members with different skills and perspectives still must negotiate what is to be built and fit together pieces of the design. Consensus formation, sharing of know-how, and integration of work outputs create communication demands that if not met at one level tend to surface at others.

Team design, while necessary for some purposes, is sometimes a misguided attempt to apply structure and formalization when they are not suitable. Formal coordination mechanisms are intended to simplify and disaggregate behavior and therefore increase group resiliency, but they can fail in the face of interdependence under uncertainty, which typifies much software work. Flexibility, texture, richness, expressiveness, and sometimes accuracy—all disappear during the codification of roles, rules, and procedures (Boisot and Child, 1988; Bruner, 1974). Under these circumstances, communication is needed for coordination (Clark et al., 1987; Daft and Lengel, 1984, 1986; Kraut and Streeter, 1990; Stohl and Redding, 1987; Van de Ven et al., 1976).

Direct communication is also referred to as coordination by feedback (March and Simon, 1958), mutual adjustment (Thompson, 1967), organicism communication networking (Tushman and Nadler, 1978), clan mechanisms (Ouchi, 1980), and informal communication (Kraut and Streeter, 1990). These terms convey the unique advantages of talking personally with others: spontaneity, interactivity, richness, friendliness. With communication, people develop deeper relationships and more opportunities to observe and learn from one another. Communication improves group commitment, socialization, and sometimes control. It makes possible the acquisition and maintenance of group culture, authority, and norms that people do not talk about overtly (Levitt and March, 1988; Nelson and Winter, 1982). Communication counters some of the costs to relationships of formal approaches to coordination. Research on communication in organizations has shown the heavy use made of communication in research and development teams where work is uncertain (e.g., Adams, 1976; Allen, 1977; Pelz and Andrews, 1966; Tushman, 1977).

Despite its advantages, constant communication in the traditional sense of face-to-face or telephone conversation is impractical in many software development teams. The ease of acquiring information is at least as important as the quality of the information in determining the sources that people use (Culnan, 1983; Zipf, 1935). Physical proximity is the major determinant of engineers’ work-related information exchange and influence on projects (Allen, 1977). Constant communication may be undesirable as well as impractical—who can be reached conveniently is not necessarily the same as who can contribute high-quality infor-

mation. Communication can be costly if highly skilled persons spend too much time communicating with others instead of completing their individual tasks (Scott and Simmons, 1975). New communication techniques can reduce the costs of direct communication. Computer networks with electronic mail and bulletin boards that allow for fast but asynchronous conversation permit project members to talk even though they are geographically dispersed or mobile. Nonetheless, as discussed in Chapter 2, communication networks that are installed to increase efficiency might actually encourage the proliferation of communications, leading people to spend more time screening messages, and thereby reduce the cost advantages of the networks. Also, these media are inefficient for some kinds of communication, notably for collaborative planning and problem solving under uncertainty (Finholt et al., 1990; Galegher and Kraut, 1990). Research on the coordination and productivity benefits and costs of network communication suggests that, appropriately managed, the net effect can be positive. However, networks and other new technologies for communication do not automatically bestow benefits on coordination.

The Dilemmas of Coordination as Linkage

As we have described, coordination does not have a simple one-to-one relationship with team performance and thus is not a simple answer to forming linkages between individual work in a team and productivity as a team. Three dilemmas characterize the linkages. First, too little or too much coordination impedes performance. Hence, the team has to invest in the right amount of coordination. Second, design and communication have different effects on teamwork; the team has to match the appropriate coordination strategy with the tasks and phases of the team’s work. Third, any coordination strategy tends to become habitual. Hence, the team must find ways to undo or unlearn design or communication strategies that might have been successful in the past but become inappropriate for a new task. We discuss these dilemmas in turn.

Amount of Coordination

Most teams use design and communication. If a team puts little or no effort into these forms of coordination, its performance will be poor. The more coordination, the better, up to a point. But coordination is costly; in experimental studies, team performance typically is above the level of the average team member but below the level of the most competent member because of coordination costs. At high levels, the
process of coordination is very costly in time, resources, hassles, and distractions (e.g., Abdel-Hamid and Madnick, 1989; Diehl and Stroebe, 1987). Thus, coordination has a curvilinear, inverted U-shaped relationship with performance.

Communication Versus Design

We propose, in addition, a dilemma of balancing approaches to teamwork. Communication and design are somewhat inconsistent with one another. For instance, teams may find task decomposition very efficient and comfortable. But if, because of their separate roles, team members do not talk to one another, friendships deteriorate and free riding increases. Members may begin to put their own prominence above the group’s, which is a form of public goods problem. Consider a 3 x 3 matrix, in which communication and design are orthogonal factors (see Figure 9-2). When communication and design are each low, performance is poor because the team is not coordinated. When communication is high, design should be only moderate to achieve high performance at low cost, and vice versa. Finally, when communication and design are each high, coordination costs interfere with performance (Figure 9-3). One can imagine this happening, for instance, in teams in which there are many direct working relationships, many meetings, and many formal procedures that have to be followed.

Most groups combine some measure of design and communication, but they may overemphasize one or the other. It may be that for every task and project, there is an appropriate level of design and an appropriate level of communication for every level of design.

In technical work, the timing of communication and design may be important. It has long been thought that group discussion is necessary at times when tasks are highly uncertain or equivocal—at the beginning of projects and during crises. Communication at these times can

---

1In public goods theory (Olson, 1971), a dilemma exists when contributions of effort and resources for a group are partly inconsistent with the self-interest of individual group members. Individual members may believe with good reason that they are better off letting others cope with unassigned group tasks, such as teaching new members and handling unplanned client interruptions. As long as someone else does the work, free-riding members still benefit from the group’s success. Also, in a complex and interactive project, one person’s contribution to the group at any particular moment may seem inconsequential. The public goods problem may increase when uncertainty is high and team design contributes to lack of communication (Macy, 1990).

---

**Figure 9.2** Patterns of communication and team design leading to different levels of performance.

**Figure 9.3** Complementariness of communication and team design at low and high levels of both, and substitution of communication and design at higher levels of both.
help create mutual understanding, commitment, and substitutability (because individuals have more common knowledge). Teams that communicate intensively initially and spend more time working out plans may do better than “fast starters” that begin coding and implementing quickly (see Hackman, 1987).

Team design might be a better strategy for coordination than direct communication once a team’s plans are in place. Design allows members the most autonomy and time to do individual work. Programming probably is the most individual task in software development. A lead programmer doing coding might be working alone or perhaps with one or two others. Since coding is a conjunctive activity (i.e., the project cannot go forward without it), the programmer is needed by others and is less substitutable than those who have been working on other jobs and jointly with others. Over time, the influence of the programmer increases (assuming this work is individualized), and the rest of the team gets more dependent on the programmer because the work is central and the role is nonsubstitutable. Development of project-specific skills and overall understanding in the programmer can be seen as addressing coordination over time in two ways. First, as more pieces of a project get built, the programmer’s competence becomes more critical to the rest of the group. Second, the programmer(s) will exert authority, which will lead to centralization and more design.

Design also addresses problems of heterogeneous skills in a team. Three elements of design are particularly important here: the role structure, the formal authority structure, and formal communication channels. The role structure specifies who does what. The formal authority structure specifies who reports to whom and who has access to what resources (Carley, 1990; Malone, 1987). Formal communication channels specify who is supposed to talk to whom. During coding and implementation phases of software development, these formal structures enable individuals to concentrate on their individual tasks and thereby successfully complete the project. These structures should be particularly useful when there is substantial heterogeneity among group members.

Entrainment

As teams develop ways of coordinating their work, they adopt habitual patterns of coordination, a process called entrainment (Kelly et al., 1990). Patterns of coordination become institutionalized and legitimated. As a consequence, it becomes more costly to renegotiate approaches to coordination. Particular styles of coordination and group cultures influenced by those styles emerge in all groups. With this emergence of a team coordination style, individual members are likely to become more similar to one another in their personal attitudes and ideas about teamwork (e.g., Kiesler and Kiesler, 1969). In experimental studies, entrainment is inferred from a group’s tendency to use the same work methods even though the task demands change. In research with ongoing groups, entrainment must be inferred by examining the extent to which coordination approaches become more similar and predictable over time.

Ironically, as teams become better at what they do and better coordinated, they also can become increasingly rigid in their approach to coordination. If their task assignments change, team members may be unable or unwilling to adjust their coordination strategies to the demands of the new tasks. They may be too internally focused and too comfortable, and their previous successful experience will not have suggested ways in which they should change. Research has only begun on this problem.

RESEARCH PROBLEMS AND DIRECTIONS

Much of the research discussed above was conducted with small, homogeneous groups working on well-specified collaborative tasks that can be done in one or a few sittings (McGrath, 1984). Except for the work on lateral coordinative mechanisms (which does not examine the role of groups in particular; e.g., Burns, 1989; Galbraith, 1972; Lawrence and Lorsch, 1967; Pfeffer, 1978), there has been relatively little research on coordination of large and ongoing teams within organizations. Also, little is known about technical teams in organizations that use computer-based technology. Such technology permits organizations to form large, dispersed, and diverse teams working on complex, uncertain, interdependent tasks that would not have been possible in the past. These teams have coordination problems that differ from those of traditional small groups and formal departments whose members are physically proximate. Laboratory and field research must employ technological and other resources to study the modern technical team.

Certain theoretical problems also must be solved if researchers and practitioners are to understand linkages between individuals and teams. Two of these problems are described in the next two sections.

Efficiency Versus Social Effects

Observations of today’s technical projects (e.g., Curtis et al., 1988; Sproull and Kiesler, 1991) suggest that multilevel theories may be required to capture fully how coordination acts as a link between individuals and group productivity. In a two-level framework, for instance,
coordination mechanisms are viewed as having efficiency effects and social effects.

Efficiency effects of coordination are the direct, intended benefits of coordination minus its direct costs. These are the benefits and costs discussed above. However, coordination mechanisms can also have systemic, long-term effects on the team, organization, or social system. (For this concept, see Maruyama, 1963; Mason, 1970.) For instance, suppose as a result of using electronic mail to coordinate work, dispersed teams also discover ways to mobilize to influence management policy. Here, communication initially intended simply as an efficiency amplifier for a team also has effects on employee participation and organizational politics. Or, as was observed in one study, management may realize that the communication system can be used to monitor individual team members' performance in ways that used to be too difficult, which changes its authority relationships with the team members (Rule and Brantley, 1990). Social effects can affect linkages and productivity qualitatively and in ways that were entirely unanticipated. For instance, while greater employee participation may have no direct effect on the performance within teams, it can increase interteam learning and exchange of expertise across teams.

**Linkages and Scaling Up**

Another theoretical problem in the study of linkages is the incomplete understanding of how to study behavior across individual, group, and organizational levels of analysis. Experimental studies of individual behavior and simple tasks are necessary to test causal hypotheses, but one cannot deduce from experimental findings what will happen with real groups in organizations. Experimental group behavior never replicates exactly that of ongoing groups in organizations.

One approach to scaling up from individuals and simple tasks to teams and more complex tasks is to add variables. Amount of discussion (a communication variable) and centralization (a design variable) are variables, for example, that would be appropriate at the group level. A more difficult scaling problem arises, however, when such variables do not scale at the same rate; then multivariate effects change, which causes a phenomenon in the large to look very different from the way it looks in the small. For instance, discussion between two persons working together seems qualitatively different from meetings of 100 or more members of a large team. Ship designers encounter this problem when they try to deduce the behavior of a full-size ship from tests of models. Two important factors in a ship's drag are waves made by the ship's prow and turbulence under the ship. Because wave effects and turbulence depend on fine details of the hull shape, designers cannot rely on mathematical calculations alone. Instead, they build scale models and tow them in water, measuring their drag. Although the model gives an estimate of drag, there is no way to measure how much of the model's drag is accounted for by turbulence and how much by making waves. To complicate matters, the two factors do not scale in the same way. The turbulence under the ship depends on the surface area and the speed to the 1.825 power, but the wave drag is a much more complex function of speed and ship size. Since the two effects are confounded in the model's drag and scale differently, scaling up from model tests is very hard. The ship in its full glory may act very differently than the model did, particularly if the model is small relative to the ship. Ship models and towing tanks are surprisingly large for that reason.

Based on evidence to date, the scaling problem is probably serious in researching the linkages between individual productivity and the productivity of large, dispersed project groups. For example, in asking how computer technologies and networks affect group coordination and productivity, researchers can test some hypotheses in the laboratory, but in reality, networks often inspire more groups, larger projects, more diverse groups, and more flexible group structures (Sproull and Kiesler, 1991). A social consequence of this is that peripheral employees, such as geographically or organizationally isolated employees, gain new opportunities to initiate and receive communication (Eveland and Biskin, 1988; Fanning and Raphael, 1986; Wasby, 1989). If management policies permit such interactions, peripheral employees can increase their membership in groups and their connections to groups. These interactions can increase information flow between the periphery and the center of the organization and among peripheral workers. In short, while increasing connections through network communication could increase the participation of everyone in principle, peripheral employees are likely to see a relatively greater impact than are central employees (Eveland and Biskin, 1988; Hesse et al., 1990; Huff et al., 1989). This chain of events looks very different from a linear scaling up from individual or even small group behavior in relatively simpler settings.

In sum, individual behavior and small group behavior may scale differently to organizational reality. Variables that seem trivial (perhaps because of low variance) in the laboratory may loom much larger in an organization—and vice versa. If so, one may see the same phenomena differently in the two settings, no matter how fine-grained and careful the research is. It is important to do both kinds of research, that is, to study individuals and small groups in the laboratory and in the field and to study large and ongoing groups in organizations. The purpose is not to discover exactly how variables and processes scale at
each level, but to ensure that researchers are always attuned to scaling problems.

**Studying Groups in Organizations**

Understanding of linkages in group productivity might be more effectively advanced if the tests of models in this domain were more ambitious scientifically. For example, an Israeli study involved a true experiment in the field on the effects of selection on tank crew performance (Tziner and Eden, 1985). The study involved the assignment of 672 soldiers to 224 crews, using a complex Latin square factorial design to control for differential performance ratings by the 28 unit commanders. Assignment on the basis of ability was varied experimentally. No other interventions were made in the natural military environment, but considerable control was exerted on data collection to increase its reliability and validity. There were four waves of measurement using previously validated instruments. The study showed that “spreading the talent around” is an inefficient way to distribute staff for interdependent groups, and the researchers were able to provide empirically supported advice counter to prevailing practices.

A kind of sociological/microeconomic study needed in the domain of software productivity is exemplified by a comparative study of product development teams in the automotive industry (Clark et al., 1987). The unit of analysis in this study was a major car development project; three U.S., eight Japanese, and nine European auto companies participated in the research. The researchers collected data from the companies on 29 projects (6 in the United States, 12 in Japan, and 11 in Europe) involving the development of new sedans, micro-mini cars, and small vans introduced from 1980 to 1987. The researchers used questionnaires and interviews with project managers, heads of R&D groups, engineering administration staff, and engineers, as well as archival data on lead time, engineering hours, technology, subcontracting, and outcomes such as model prices. This study confirmed that Japanese projects were completed in two-thirds the time and one-third the engineering hours of the non-Japanese projects, and it reconfirmed that if schedules are kept under control, cost overruns also tend to be restrained. These results do not refute a time-cost trade-off. Rather, the study points to the potential importance of particular project strategies, kinds of project organization, leadership, and staffing.

**CONCLUSION**

A changing but mostly large proportion of the variance in the productivity of software development and other technical teams derives from how such teams coordinate their work. Without coordination, individual work cannot be integrated and turned into a group product. Technical teams use team design and communication to coordinate their work, each of which can be considered a linkage process. Research has contributed much to the understanding of the additive and interactive effects of team design and communication on coordination. They are, in part, substitutes for one another. Too much of either one, or of both, creates costs that outweigh the benefits of coordination. There are many unknowns in this domain, however, especially when one tries to predict the side effects and outcomes of linkages over time. The very meaning of productivity in software development has changed as approaches to coordination have changed. Improvements in IT and formal methodologies used for coordination have increased the scope of software engineering projects. In 1963 the Mercury space project required 1.5 million object instructions, whereas a space station of the 1990s requires at least 80 million. Today, software development teams are generally much larger, more diverse, better trained, and more dispersed than they used to be. Moreover, their tasks are more complex, more uncertain, and more fluid than they were in the past—all this despite improvements in hardware and software that have made individual work and coordination less onerous. Hence, as new technological and nontecnological approaches to linkages are developed, there are new efficiency and social consequences, including changes in one’s expectations of team productivity.

**REFERENCES**

Abdel-Hamid, T.K., and S.E. Madnick  

Adams, J.S.  

Aldrich, H.  

Allen, T.J.  

Banker, R.D., S.M. Datar, and C.F. Kemerer  
Barnlund, D.C., and C. Harland

Beatty, C.A.

Boehm, B.W.

Boisot, M., and J. Child

Bossert, S.T.

Brewer, M.B., and R.M. Kramer

Brooks, F.P.

Bruner, J.

Burns, L.R.

Burns, T., and G. Stalker

Card, D.N., F.E. McGarry, and G.T. Page

Carley, K.M.


Chase, M.L.

Clark, K.B., W.B. Chew, and T. Fujimoto

Clark, N.K., and M.M. Stephenson

Culnan, M.J.

Curtis, B.

Curtis, B., H. Krasner, and N. Iscoe

Cyert, R.M., and J.G. March

Daft, R.L., and R.H. Lengel


Davis, W.S.

Dearborn, D.C., and H.A. Simon

Diehl, M., and W. Stroebe

Dietrich, W.C., I.R. Nackman, and F. Gracer

Downs, A.

Eveland, J.D., and T.K. Bikson

Fanning, T., and B. Raphael

Faunce, W.A.

Festingher, L., S. Schachter, and K. Back

Finholt, T., L. Sproull, and S. Kiesler

Fox, J.M.

Galbraith, J.K.
Porter, L.W., E.E. Lawler, and J.R. Hackman  

Rule, J., and P. Brantley  

Rumbaugh, J.  

Scott, R.P., and D.B. Simmons  

Simon, H.A.  

Sims, M.L.  

Sodhi, J.  

Sproull, L., and S. Kiesler  

Spurr, K., and P. Layzell, eds.  
1990  *Case on Trial*. Chichester: John Wiley & Sons.

Steiner, J.D.  

Stohl, C., and W.C. Redding  

Sykes, R.E., K. Larrotz, and J.C. Fox  
1978  Proximity and similarity effects on frequency of interaction in a class of naval recruits. *Sociometry* 39:253-269.

Tajfel, H.  

Thompson, J.D.  

Travis, P.  

Tushman, M.L.  

Tushman, M.L., and D. Nadler  

Tziner, A., and D. Eden  

Van de Ven, A.H., A.L. Delbecq, and R. Koenig, Jr.  

Verastegui, R.J., and D.J. Williams  

Wasby, S.  

Weinberg, G.M.  

Zarela, P.F.  

Zipf, G.K.  