Robert E. Kraut and Lynn A. Streeter

Coordination in **Software Development**

Since its inception, the software industry has been in crisis. As Blazer noted 20 years ago, "[Software] is unreliable, delivered late, unresponsive to change, inefficient, and expensive . . . and has been for the past 20 years" [4]. In a survey of software contractors and government contract officers, over half of the respondents believed that calendar overruns, cost overruns, code that required in-house modifications before being usable, and code that was difficult to modify were common problems in the software projects they supervised [22]. Even today, problems with software systems are common and highly-publicized occurrences.

While there is no single cause of the software crisis, a major contributor is the problem of coordinating activities while developing large software systems. We will argue that this coordination becomes much more difficult as project size and complexity increase. Coordination difficulties are not limited to software development, though, but are an inherent aspect of work in any large organization. Viewed from this perspective, some of the mechanisms used to coordinate work in large organizations in general ought to have applicability to software development. In particular, we examine the respective roles of formal and informal communication mechanisms in coordinating work on software projects. We will argue that most of the existing coordination support tools have used formal communication procedures, and that there is a need for nurturing informal communication procedures as well.

Coordination has been defined as the direction of "individuals' efforts toward achieving common and explicitly recognized goals" [3] and "the integration or linking together of different parts of an organization to accomplish a collective set of tasks" [23]. In software development, it means that different people working on a common project agree to a common definition of what they are building, share information, and mesh their activities. They must have a common view of what the software they are constructing should do, how it should be organized, and how it should fit with other software systems already in place or undergoing parallel development. To build the software efficiently, they must share detailed design specifications and information about the progress of software modules. In sum, they must coordinate their work so that it gets done and fits together, so that it isn't done redundantly, and so that components of the work are handed off expeditiously.

Characteristics of Software Development

Achieving a successful software system requires tight coordination among the various efforts involved in the software development cycle. Yet this coordination is difficult to achieve. As Curtis, Krasner and Iscoe [11] note in their study of large software development projects, communication bottlenecks and breakdowns are very common. Indeed, several characteristics of software development make these coordination problems not just common, but inevitable [6, 8].

Scale. A fundamental characteristic of many software systems is that they are very large and far beyond the ability of any individual or small group to create or even to understand in detail. If a software system were small, effective coordination could occur because a single individual or small group could direct its work and keep all the implementation details in focus. Indeed, large projects are more successful if a single, often exceptional, individual with both application-domain knowledge and software knowledge guides and coordinates the project [11]. But this ideal is impossible for many large software systems, where system size is measured in millions or tens of millions of lines of code and the life of the

project is measured in years.[1]

Efforts of this scale invariably lead to specialization and a division of labor. These organizational responses in turn lead to compartmentalization of interdependent actors through geographic, organizational, and social boundaries. Within these boundaries, unique subgroup perspectives, cohesiveness, ethnocentrism, and unwillingness to trade information increase (e.g., [7]). Barriers—geographic, organizational, or social—reduce people's opportunities and eagerness to share information and to learn from distant colleagues [18]. While compartmentalization promotes organizational efficiency in large groups by shielding people from unnecessary information, it nonetheless creates new coordination tasks. Compartments limit people's breadth of experience, leading to errors, narrowness, and insufficient opportunity for comparing knowledge, and can reduce the motivation to interact with relevant others and to accept new ideas.

Uncertainty. The inherent uncertainty in software development compounds the coordination problems produced by large scale alone. By uncertainty, we mean the unpredictability of both the software and the tasks that software engineers perform. Unlike much manufacturing, software development is a nonroutine activity. Many software systems are one-of-a-kind projects with no existing prototypes, applications or systems to simply modify or change.

Further, uncertainty increases because specifications of the software's functionality change over time [9, 11]. Change in software specifications arises because the external world that the software was designed to support changes, as business needs, user desires, computer platforms, input data, and the physical world itself all change. The likelihood of change is greatest whenever software is used directly by people, because it is often only by using software that purchasers and users understand its capabilities and limitations. When software is used in circumstances for which it wasn't designed specifically, the users are likely to demand new capabilities that had not been envisioned during the initial design.

Software development also is uncertain because specifications for it are invariably incomplete. Incompleteness partially results from limited domain knowledge and division of labor typical of software projects [11]. Too few people working on a software project have sufficient knowledge about the domain in which they are working. A project group writing software for a heads-up display for pilots needs in-depth knowledge of aircraft and aviation, as well as knowledge of computer science. Typically, analysts with varying degrees of domain knowledge interview customers and users, and then write specifications for software architects and designers. In this process, relevant domain information is inevitably lost.

Some of the users' needs will not be uncovered by the analysts, and some of the analysts' discoveries will not be reflected in the specifications. Thus, a major coordination problem in software development is that at many points the information that software architects and programmers need to make decisions is not available to them through documents, although users, analysts, and others in the project may have the knowledge necessary for these decisions.

Finally, software is uncertain because the different subgroups involved in its development often have different beliefs about what it should do and how it should do it. For example, analysts translate users' needs into requirements for system capabilities. This task is exceedingly difficult and open to error and misrepresentation, since it involves synthesizing, representing, and often reconciling different user needs and views, as is the case when different groups of end-users have different levels of computer skill. While analysts may try to adopt the point of view of the software's users, designers and programmers often have a more technical focus, with an emphasis on ease of development and efficiency of computation. As more groups become involved in software development, disagreements among them inevitably increase. These differences in points of view must be resolved for coordination to succeed.

Interdependence. The large size and uncertainty in software work would be less of a problem if software didn't require precise integration of its components. Much software is built of thousands of modules that must mesh with each other perfectly for the software system to operate correctly. The recent disruption of the AT&T long distance network [20] shows how unanticipated interactions among software modules can have disastrous consequences. Poor coordination between subgroups producing software modules could lead to failure in integrating the modules themselves.

Informal communication. Both practical experience and organizational theory suggest that previous efforts in software engineering have not solved the coordination problems in large software projects. The combination of large size, uncertainty, and interdependence requires special coordination techniques that may not be necessary in more routine production environments. At the risk of oversimplification, one can say that most proposed remedies for the software crisis have taken one of three approaches: (1) technical tools, ranging from new workstations to syntax-directed editors and higher level languages, to improve the productivity of individual developers, (2) modularization, both technical, such as object-oriented programming, and managerial, such as the organizational separation of the requirements, coding, and testing functions, to encapsulate the behavior of program elements and individual software professionals, and thereby reduce the needs for coordination, and (3) formal procedures, both technical, such as version control software, case tools, and specification languages, and managerial, such as test plans, deliv-

[1]The software used to run ground control for the Apollo spacecraft in the 1970s contained about 23 million lines of code [12]. The code for AT&T's 5ESS switch is about 10 million lines of code. The software to allocate lines in a telephone network contains over 10 million lines of code. Each generation of software is typically larger than the one that preceded it. Assuming typical productivity (measured in lines of new or changed production-quality code per staff year), a software system with one million lines of code might require 500 staff-years of effort (Martin, personal communication), if one considers the analysts, programmers, support staff, testers, document writers, managers, and administrative personnel involved in a large project.

ery schedules, and requirements documents, to control communication among development personnel.

While these techniques have undoubtedly contributed to a modest increase in software productivity over the past twenty years [6] they only partially address the problems of coordination in software development. Tools to increase the productivity of individual programmers by definition do not solve coordination problems. Likewise, while successfully layered architectures and structured programming techniques may reduce the number of interfaces between modules, different people with different perspectives still must agree on what is to be built and must fit together pieces of software. Problems in consensus formation, information sharing and coordination that don't show at one level invariably surface at another. Finally, formalization, while necessary and applicable for many tasks, may be misapplied to tasks that are difficult to routinize.

Prior research shows that formal and informal communication are best suited for different types of activities. By formal communication we mean communication through writing, structured meetings, and other relatively non-interactive and impersonal communication channels. In the case of software development, formal communication includes techniques such as written specification documents, formal specification languages, status review meetings, and automated reporting and tracking of program errors. These are in contrast to informal communication, by which we mean personal, peer-oriented, and interactive communication. Formal communication is useful for coordinating routine transactions within groups and organizations, but it often fails in the face of uncertainty, which typifies much software work. Under these circumstances, informal communication may be needed for coordination [23].

Analyses of communication in research and development settings, although not looking at software development per se, have shown the heavy and effective use that professionals make of informal communication for exchanging information. The major findings are easy to summarize. First, informal, interpersonal communication is the primary way that information flows into and through research and development organizations [1, 2, 21]. Second, in the world of research and development as in many other domains [10, 25], the ease of acquiring information is at least as important as the quality of the information in determining the sources that people use. Therefore physical proximity of a source is a major constraint on engineers' work-related information [2]. Third, getting information and coordinating activity through informal, interpersonal communication is valuable both for individuals and for their organizations, especially as research and development tasks become more uncertain [19, 21].

The previous discussion points to a major and perhaps unresolvable tension in large software development projects. Because of interdependence, different groups involved in a software development project must be tightly coordinated. Because of the high degree of uncertainty typical of software projects, informal, interpersonal communication should be a valuable method for achieving this coordination. But because of the large size of these projects, the inefficiencies of pairwise face-to-face communication may preclude the use of informal communication as a practical technique for solving coordination problems. And because of the tight coupling necessary between software modules, speech, which is the primary medium of informal communication, may be too imprecise to communicate well and too ephemeral to serve as a record of the information exchange.

This article is an empirical examination of the conditions under which different techniques are used to coordinate software development and the conditions under which they actually succeed or fail in improving coordination. For the reasons just described, we concentrate on the contrast between relatively formal, impersonal techniques, such as design and requirements documents and status tracking methodologies, and relatively informal, interpersonal techniques, such as peer discussion and unstructured electronic mail. Structured, interpersonal meetings, such as status reviews or design reviews, are intermediate on this formality dimension.

A Survey Study of Coordination in Software Development

We surveyed the intergroup coordination practices across 65 projects in one large software development company. The survey focused on three factors: 1) coordination practices used, 2) structural characteristics of projects that might interact with the practicality and utility of various coordination techniques, and 3) the success of the projects on several dimensions. We were particularly interested in features of the projects and the coordination practices that influenced the sharing of information and of goals.

The research site. The research site was the software development division of a research and development company that employed approximately 3,000 managers, analysts, software engineers, programmers, testers, and documentation specialists. Collectively, the staff worked on the development of a wide range of products for the telecommunications industry, using a wide range of techniques. In general, projects were organized around a waterfall development model and had standard development environments, code reviews, and quality programs. In terms of scale, they ranged from two- to four-person projects, developing software for PCs, on the one hand, and large mainframe systems, with 14 million lines of code already developed and 150 people on staff on the other. The median project had over 15 people on staff at the time of the survey, in 1990. In terms of the software life cycle, the projects ranged from those in the specification stage with active negotiations with clients and other development organizations, to more maintenance-oriented projects, where new releases were meant to fix bugs and add small numbers of features.

While all projects used both formal and informal communication to coordinate activity, the balance differed across projects. For example, in projects on the more formal end, different units of the company wrote requirement specification documents

and designed software architecture. The requirements unit conveyed information about needed software capabilities to the development unit through these formal specification documents. In more informal projects, the same 30-person department was responsible for assessing software capabilities and for actually writing the code. Their communication was through informal, interpersonal contacts supplemented by sketchy requirements documents. Similarly, some departments made extensive use of electronic mail and bulletin boards to distribute project knowledge, while others did not use these facilities.

Sample. The sample consisted of 150 supervisory groups involved in some aspect of software development, representing approximately 80 different software systems or major subcomponents of a system. We call the people working on one of these systems a project. The coordination survey was sent to 750 people (150 first-level supervisors and 600 technical staff). People from all phases of software development, from requirements to field support, were included in the sample. Eighty-eight percent of the sample returned the questionnaire after three mailings, and of these, 563, or 75% of the total sample, returned usable data.

There were 65 projects for which at least two people provided data. Projects with only one respondent were dropped. The number of respondents per project ranged from 2 to 47 with a mean of 7.6 and median of 4. Depending on the question, analyses are based on 563 individuals or 65 projects. At the project level, the data were averaged over all respondents who reported working on the same software system.

Measures. The survey measured the following aspects of software development within a project. Table 1 shows examples of all measures.

1. *Structural characteristics of projects.* These include *project size,* in terms of number of employees, *project age,* and *stage* in the software life cycle. Two other structural characteristics were important and require additional explanation. One, *organizational interdependence,* is the extent to which

Table 1. Independent and dependent variables, with examples of scale items

STRUCTURAL CHARACTERISTICS
PROJECT SIZE
1. Number of people working on this project across the company
PROJECT AGE
1. Maximum number of years that any project member worked on the project
PLANNING STAGE
1. Percent of project staff having as their major work activity either requirements analysis and specification or high-level software architecture and design
ORGANIZATIONAL INTERDEPENDENCE
(5-items, Alpha = 0.69) 1. Extent to which your work is interrelated with people in your division, but outside your group 2. Extent to which work is related with the work of others outside your division, but within the assistant vice presidential area 3. Extent to which work is related with work outside the assistant vice presidential area, but within the vice presidential area
PROJECT CERTAINTY
(8-item, Alpha = 0.65, Adapted from [23]) 1. Clearly defined body of knowledge or subject matter guiding work on the project 2. Extent to which people in a particular district do the same type of work 3. Stability of the detailed specifications for the project
COORDINATION TECHNIQUES
FORMAL IMPERSONAL PROCEDURES
(5-items, Alpha = 0.63) 1. Project documents and memos 2. Project milestones and delivery schedules 3. Modification request and error tracking procedures 4. Data dictionaries
FORMAL, INTERPERSONAL PROCEDURES
(5-items, Alpha = 0.74) 1. Status review meetings 2. Design review meetings 3. Code inspections
INFORMAL, INTERPERSONAL PROCEDURES
(2-items, Alpha = 0.56) 1. Group meetings 2. Colocation of requirements and development staff
ELECTRONIC COMMUNICATION
(2-items, Alpha = 0.38) 1. Electronic mail 2. Electronic bulletin boards
INTERPERSONAL NETWORK
(1-item) 1. Number of supervisors from outside the project talked to in the previous two years

Table 1.

OUTCOMES
PROJECT MEMBERS INFORMED
(5-items, Alpha = 0.83) 1. Management on the project has an accurate view of how well the project is going 2. Immediate manager is a good source of information about relevant work from other areas of company 3. People are well informed about project status
COORDINATION SUCCESS
(5-items, Alpha = 0.65, Adopted from [13]) 1. Lack of duplication and redundancy in the work of groups on the project 2. Extent to which the group avoids working in a "crisis mode"
CLIENT SATISFACTION
(12-items, Alpha = 0.85) 1. Ease of learning product 2. Absence of defects in the software 3. Company responsiveness in providing fixes to problems you encountered 4. Overall quality of the products and services provided 5. Suppose that you had a need for another software product in this area. How likely would you be to choose our company as a provider?
MANAGERS' EVALUATION
(2-items, Alpha = 0.91) 1. The quality of the software development process: how efficiently and effectively a project does software development, in terms of use of personnel, meeting schedules, maintaining good communications, etc. 2. The quality of the software product delivered to the client: from the client's point of view, was the software delivered on time, does it have the right mix of features, and is it error-free?
SOFTWARE PRODUCTIVITY
(1-item) 1. Number of new or changed lines of uncommented code per staff member per quarter
SOFTWARE QUALITY
(5-item, alpha = 0.76) 1. Number of field errors reported per thousand lines of new or changed uncommented code (reverse-scored) 2. Number of errors found in system test per thousand lines of new or changed uncommented code (reverse-scored) 3. Number of days to fix a moderately severe field fault (reverse-scored)

members of a project get inputs from and pass outputs to other groups within the company, outside of their immediate supervisory group. The other, *project certainty*, is the extent to which a project is stable and consists of tasks that are well understood and easily accomplished by local expertise.

2. *Coordination techniques used.* Respondents were asked how extensively they used various coordination techniques on their projects. These include: 1) *formal, impersonal coordination techniques*, such as written requirements documents, modification request tracking, and data dictionaries; 2) *formal, interpersonal techniques*, such as requirement review meetings, status review meetings, and code inspection meetings; and 3) *informal interpersonal techniques*, such as unscheduled group meetings or co-location of requirements and design staff. Among the informal techniques, we examined separately 4) *electronic communication*, such as electronic mail and electronic bulletin boards, and 5) *interpersonal networks*.

The interpersonal network measure requires additional explanation. Respondents were given a list of 100 first-level supervisors randomly selected from the software development units and asked to identify all those with whom they had talked in the previous two years. Projects in which people had many contacts outside the project had extensive interpersonal networks.[2]

3. *Outcome measures.* This article concentrates on two measures of the consequences of using different coordination techniques. One, the informed scale, is based on respondents' assessments of how informed they and their managers were. The other, the coordination success scale, reflects how coordinated they believed their projects were. To assess the validity of these measures, we included three additional outcome measures available for subsets of the projects: (1) senior managements' assessments of the quality of the software product and process, (2) software metric data on productivity and quality, and (3) client ratings of the quality of the software product.

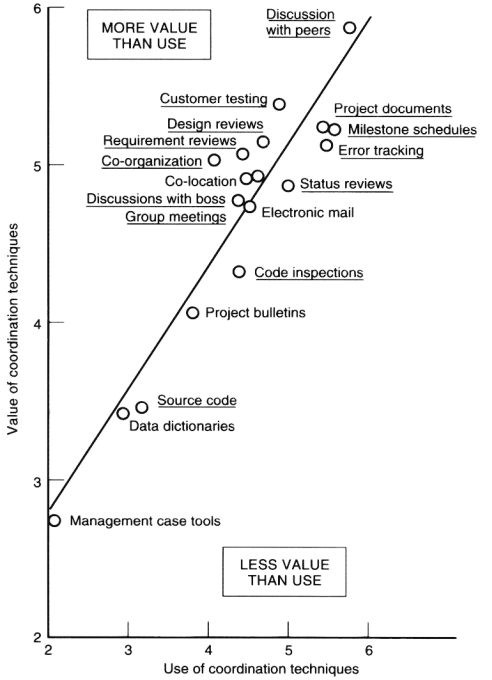
A. The informed scale measures respondents' assessments of how adequately informed they and the project's managers were about project status and responsibilities.

B. The coordination success scale measures respondents' assessments of how well their projects were going and integrating with the work of other organizations.

C. Higher managements' evaluations of software quality. The nine senior

[2]Some reviewers have questioned the validity of using the number of contacts outside a supervisory group as a measure of a project's interpersonal network. They would have preferred an estimate of an information gatekeeper's network instead [2, 21]. While focusing on the highly communicative gatekeepers might have produced better data than computing contacts for the average member of the staff, such a proposal requires some way to pre-identify gatekeepers for sampling, a problem we did not know how to solve. However, consider the worst case in which a few highly effective gatekeepers maintain the interproject contacts. We reasoned that even here gatekeepers will often be providing the names of other people to contact (as well as substantive information) to members of their groups. Thus, even in this case, the number of outside people known to the average project member is a sensible measure, and unlike other measures, the mean is an unbiased estimate of the project's interpersonal network.

managers responsible for software projects in this company rated projects about which they had personal knowledge in terms of (1) the quality of the software produced and (2) the quality of the software development process. These managers' span of control ranged from 200 to 600 people, averaging at about 500 people. We have data for 59 projects. In their instructions they were told that "the quality of the software development process refers to factors such as having good coordination among different groups working on the project, meeting milestones and schedules, knowing and controlling project status, and having good personnel morale." On the other hand, quality of the software product was defined to be "producing software products that meet the needs of the customers, that do what they were intended to do, that are easy to use, that meet performance targets, that are appropriately bug-free when delivered and that are easy to change and maintain." Quality also included client satisfaction with the software and the organization producing it. Managers' assessments of these two dimensions were highly correlated, and were averaged across questions and across managers to produce a single managerial assessment for each project. On average, each manager rated 17.9 projects, and each project was rated by 3.1 managers.



Note. Figure shows the relationship between the extent to which software engineers used a technique to spread information and coordinate their work and how much they valued it. Underlined techniques had value that was significantly different from what one would predict from the amount they were used. Techniques above and to the left of the regression line were judged as relatively valuable given the amount they are used, while techniques below and to the right of the regression line were judged as relatively valueless relative to the amount they are used.

Figure 1. Comparing the use and value of coordination techniques

D. Software metric data. Software metrics were collected quarterly as part of project management for all of the major software projects in this company. We used a subset of these metrics that contained data for the largest number of projects. These included *productivity measures*—lines of new or changed source code per staff-month—and *software quality measures* including the number of errors per 1,000 lines of code found during code inspection, during testing, and in the field, and time to fix field faults of moderate severity. We standardized each of the measures to have a mean of 0 and a standard deviation of 1, and we averaged two years' worth of data surrounding the time of the questionnaire administration (that is, eight quarters) to obtain a stable estimate for each project. Data are available for 53 projects.

E. Client satisfaction data. Corporately-collected survey data were available from project managers and end-users of the software in client sites for the 18 largest of the 65 software projects. We used a subset of the items collected on this corporate survey for our purposes. Clients were asked to rate the overall quality of the software, responsiveness and helpfulness of the developers, the degree to which the software met client needs, the adequacy of training and documentation, and whether they would buy from the software company again. These items were combined to form a client satisfaction scale. Between ten and several hundred clients provided data for each project. Respondents' data were averaged to provide a single client satisfaction measure for each project.

Results

The coordination techniques that projects with different characteristics use and the conditions under which they find them valuable are discussed

Table 2. Predicting the use and value of coordination techniques

Project characteristics	Coordination Techniques: Formal impersonal procedures		Formal interpersonal procedures		Informal interpersonal procedures		Electronic communication		Interpersonal networks
	Use	Value	Use	Value	Use	Value	Use	Value	Use
Project age	0.17	0.04	0.13	0.14	0.16	−0.03	−0.03	−0.06	−0.08
Project size	0.32*	0.02	0.50*	0.08	−0.20	0.07	0.21	0.10	−0.30*
Planning stage	−0.30*	−0.01	−0.05	0.36*	0.27*	0.28*	0.14	0.18	0.14
Group interdependence	−0.02	0.10	−0.18	−0.18	−0.01	−0.02	0.34*	−0.03	0.42*
Project certainty	0.24*	0.39*	0.16	0.32*	−0.08	0.46*	0.10	0.06	0.29*
Use of the coordination technique		0.56*		0.40*		0.48*		0.77*	

Note. Entries are the standardized beta weights in a regression analysis predicting the use and value of coordination techniques (columns) by project characteristics (rows). Use of a technique was entered into the equation predicting its value. Interpersonal networks were measured differently from the other coordination techniques. A distinction between use and value was not made.
*$p < 0.05$
$N = 65$ projects

in this section. In a later section, we will attempt to determine whether the use of any of these techniques actually aids coordination or contributes to the success of the projects.

Techniques for spreading project information and coordinating work. We examined the conditions under which projects used formal procedures and informal ones to coordinate work, and the value they judged these procedures to have. For each technique respondents made two ratings: (1) the extent to which the people used the technique in their projects to get work done and (2) the value of each technique for "spreading project information and coordinating the work" regardless of its use.[3]

Each rating was made using a 7-point Likert scale with "1" indicating "no use" or "low value" and "7" indicating "used a lot" or "high value". Analyses are based on two linear regressions, one predicting techniques used from characteristics of projects and the second predicting their perceived value from project characteristics and use. The project is the unit of analysis.

As Table 2 shows, projects tended to use formal impersonal procedures such as project documents, milestone and delivery schedules, and error tracking significantly more when the projects were certain, larger, and had passed the requirements and design stages of their life cycles. The more they were used, the more these formal procedures were judged valuable, and they were judged especially valuable when projects were more certain.

Formal information exchange meetings such as requirements and status reviews were used most in large projects. They were judged most valuable when projects were more certain and when they were in planning stages. On the other hand, projects tended to use informal, interpersonal communication, such as unscheduled meetings, very frequently, and regardless of project size, certainty, or life cycle. This informal communication was judged especially valuable when the project was certain and when it was in the planning stages. Finally, electronic communication was used more when projects were heavily dependent on input from other groups in the company. No project characteristics predicted the value of electronic communication, once one controlled for its use. Note that the association of use and value was reliably stronger for electronic communication than it was for any other coordination technique. Finally, projects had more extensive interpersonal networks when the project was smaller, when it depended on input from other groups in the company, and when it was certain.

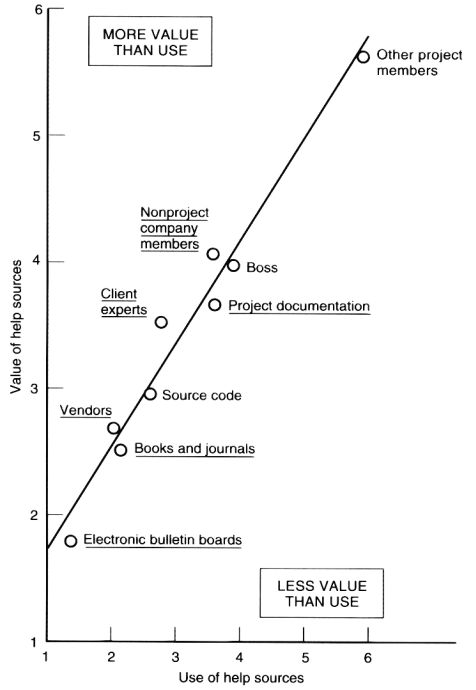
Figure 1 extends these results by looking at the use and value of particular techniques. As Figure 1 shows, use and value were highly correlated. People tended to judge techniques they used more as more valuable for two reasons. First, people select valuable coordination techniques to use. Second, people have a strong tendency to like anything they are familiar with, independently of its intrinsic value [24]. Our analysis attempts to correct for this inherent use-value bias or correlation.

Figure 1 shows the line found by regressing the use of all the coordination techniques on value. Those techniques that are above the regression line were judged to be more valuable than predicted by current use, while those below the line were judged to be less valuable than one would expect, based on their use. Techniques that were judged to be statistically significantly more or less valuable than would be predicted by their use (that is, outside the 95% confidence interval of the regression line) are underlined. Techniques that are significantly more valuable than indicated

[3]Note that the procedure was different for the interpersonal networks measure. For this measure, respondents completed a communications roster, rather than estimating how frequently they used interpersonal networks. They did not rate the value of the networks.

by their use include both informal discussions with geographically local colleagues (discussions with boss, discussions with peers, group meetings) and other interpersonal procedures that make new points of view available to the local work group. Thus requirements reviews, design reviews, and customer testing allow personnel with different responsibilities and points of view, such as requirements analysts, systems engineers, architects, programmers, testers, and customers, to comment on each other's work interactively, but in a structured setting.

The co-organization of requirements and development—the placement of these functions under a single line of management—allows for more frequent communication among the personnel responsible for these functions. Techniques that respondents judged to be statistically significantly less valuable than predicted by their use tended to be formal coordination techniques, both interpersonal ones in which the information conveyed is relatively routine (code inspections, status reviews) and impersonal ones (project documents, examination of source code, milestones schedules, and error tracking).



Note. Figure shows the relationship between the extent to which software engineers used a source to get help on a work problem and how much they valued it. Underlined sources had value that was significantly different from what one would predict from the amount they were used. Sources above and to the left of the regression line were judged as relatively valuable given the amount they are used, while sources below and to the right of the regression line were judged as relatively valueless relative to the amount they are used.

Figure 2. Comparing the use and value of sources of help

How do software professionals get help? The way project members deal with specific work problems also reflects coordination patterns in software development. We asked people to describe their most recent project-related problem. A variety of technical and managerial problems were reported. Technical problems included such difficulties as dealing with corrupt data in a database, deciding on a new programming language to use on a project, determining whether a particular piece of data was needed for an interface, or investigating why a software module ran too slowly. Managerial problems included specifying a human interface that was being jointly defined by two separate companies, calculating a complete cost estimate for a new project, or handling a conflict in responsibilities between a developer and a document writer.

From the list of nine information and consultation sources, respondents rated (1) the extent to which the source was used to solve the particular problem and (2) the potential usefulness of the source, whether or not it was actually used. Again, ratings were made on 7-point Likert scales with "1" indicating that the source was "not consulted" or was of "low" potential usefulness, and "7" that the source was "strongly consulted" or of "high" potential usefulness.

Figure 2 parallels Figure 1 and shows which information sources were valued more than one would expect from their use (above the line), and which were valued less than one would predict from their use (below the line). By far, other people were the most used and valued sources of help in software development projects and, compared to various documents, were valued more than was predictable from their use. Consistent with previous research (for example, [2]), software engineers overwhelmingly get their information from other people, and the ease of getting the information is a critical determinant. Thus, in our data, three of the four information sources used most were other people. Other project

Table 3. Intercorrelations among outcome measures

	Coordination success	Managers' evaluations	Client satisfaction	Software productivity	Software quality
Coordination success	1.00				
Management ratings	0.13** (65)	1.00			
Client satisfaction	0.61** (18)	0.16 (18)	1.00		
Software productivity	−0.13 (51)	0.11 (51)	−0.26 (16)	1.00	
Software quality	−0.05 (51)	−0.06 (45)	−0.11 (16)	0.36** (51)	1.00

Note:
Entries are Pearson correlation coefficients.
Number of projects on which each correlation is based is in parentheses.
$^{**}p < 0.01$

Table 4. Path coefficients for predicting outcome measures

Independent variables	Dependent variables					
Project age	0.25†	0.00	0.17	−0.02	0.30**	0.22
Project size	−0.06	−0.37**	0.28*	−0.05	−0.11	0.23
Interdependence	−0.28*	0.53**	−0.02	−0.09	−0.11	−0.47**
Planning stage	−0.09	0.34**	−0.28*	−0.13	0.00	0.01
Project certainty		0.22†	0.27*	0.56**	0.30**	0.24
Interpersonal networks				0.35**	−0.08	0.32†
Formal procedures				−0.22	0.01	0.26†
Certainty* networks				−0.37**	0.17	0.03
Members informed					0.46*	−0.26†
Adjusted R^2	0.09	0.35	0.27	0.40	0.55	0.30

Note: Entries are standardized Beta Coefficients
$^{\dagger}p < 0.10$
$^{*}p < 0.05$
$^{**}p < 0.01$

members, often those in close physical proximity to the respondent, were used as information sources far more than any other source.

On the other hand, people who were difficult to access (other company employees not on the project, subject matter experts from outside the company, and vendor representatives) were used as information sources substantially less than other project members. Yet these outsiders were judged to be valuable sources of help, more so than was predicted by the degree to which they were used. In contrast, all forms of written documentation were judged as less valuable than personal contacts, and some of them were judged to be significantly less valuable than one would predict from their use (project documents and memos, books, and journals, and electronic bulletin boards). Again, these results suggest the value of getting information by interpersonal means from outside one's immediate work group.

Interrelations among outcome measures. This article focuses on coordination success, but coordination success is only one indicator of success in software development projects and may even be irrelevant if it bears no relationship to other measures of success. Moreover, the measure of coordination success used in this study is based on the responses of workers and first-level supervisors in each project. To assess the validity of this measure, we computed the Pearson correlations among all the outcome measures available. This analysis is presented in Table 3.

The most important finding revealed in Table 3 is that staff members' assessment of their project's coordination strongly correlates with customers' satisfaction with the software development company and the software it produces. A second interesting finding is that senior management's assessment of the quality of projects is unrelated to other measures of project success. Finally, measures of software productivity and software quality, based on software metric data, are interrelated: projects

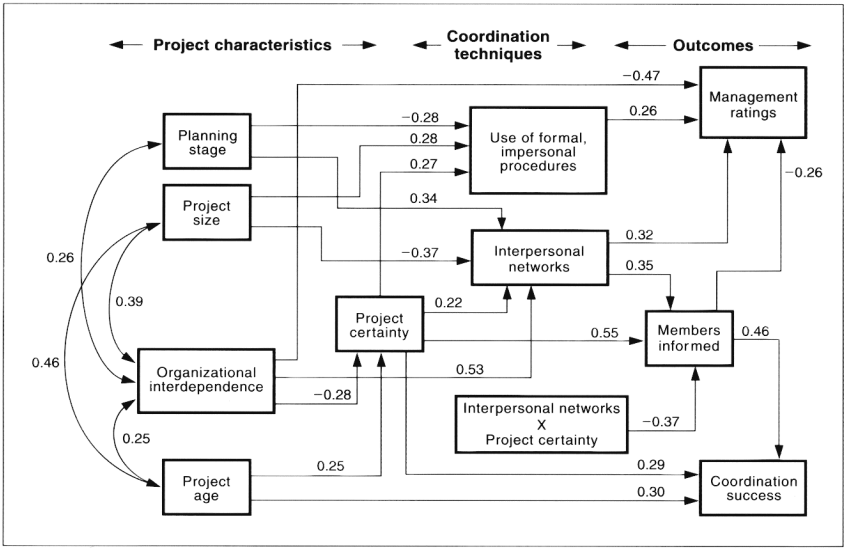


Figure 3. Factors influencing successful coordination

that produce many lines of code also produce code of good quality (that is, fewer errors, faults, and shorter time to fix field faults). Yet these software metric data are unrelated to clients' assessments of the project. Why the software metric data failed to correlate with other outcome measures is not clear. It may be that the metrics collected and used in this company are not the appropriate ones. Indeed, it is generally the case that much more validation against known outcomes is needed in the area of software metrics. It may be also that comparing even valid metrics across projects is problematic; metrics may only be useful as a comparison within a single project.

Predicting Coordination Success. In this section we use path analysis to understand the project characteristics and coordination techniques that predict successful project coordination. The coordination success scale—our measure of the degree to which a project is well coordinated—was discussed previously.

Path analysis uses a system of linear regression equations to test causal hypotheses by holding constant the effects of antecedent variables when estimating the causal impact of a focal variable. In the path analysis, we assumed the following causal ordering: structural characteristics, such as project age, size, and interdependence could potentially influence a project's certainty. Together these project characteristics could potentially influence the coordination techniques a project adopts, which, in turn, could potentially influence how well informed project members were and how successfully the project was coordinated. This causal ordering is consistent with both the prior literature on task certainty and coordination (e.g., [21, 23]) and the literature on software engineering (e.g., [8, 12]). However, like all path analyses based on cross-sectional data, the ordering can be debated. For example, the lack of success on a project could cause managers to increase its size (one of the dangers Brooks [9] warned about), or the coordination techniques that a project adopts may affect the project's certainty.

Figure 3 presents the path model showing only significant relationships among the variables from Table 2 and the outcome measures. The numbers on the arcs are the standardized beta weights from Table 4. The standardized beta weight is a measure of strength of association. It shows the direction and magnitude of change in a dependent variable (in standard deviation units) when an independent variable increases by one standard deviation unit. For example, project size positively predicts use of formal, impersonal procedures (with a standardized beta weight of 0.28), whereas use of formal impersonal procedures predicts managerial ratings (with a standardized beta weight of 0.25), but does not predict coordination success (the two boxes are not connected). Note that the information shown in Figure 3 combines two separate path analyses, one predicting coordination success and the other predicting higher management's ratings of projects. The results of the two analyses are displayed together for completeness and coherence.

The first conclusion to be drawn

from Figure 3 is that projects with different characteristics differed in the degree to which they were coordinated. As one might expect, projects that were older, smaller, and less interorganizationally interdependent were better coordinated. In addition, projects that were more technically certain (that is, stable, homogeneous and confronting routine problems) had project staff who were better informed and better coordinated. Interestingly, both project age and project interdependence had part of their effects on coordination success through project certainty. That is, older projects were on average more certain, and this factor made coordination easier to accomplish. Similarly, projects that were less interdependent have more control over the directions they set and the resources available to them. These factors made project members more informed about decisions that affected project success. These factors also made projects more certain and, in turn, better coordinated.

We had seen earlier that projects with different characteristics rely on different coordination techniques. In particular, we have seen that larger projects, more certain projects, and projects that have passed beyond the planning phases of the software life cycle were more likely to use formal, impersonal coordination techniques such as written documents, milestone and delivery schedules, and management tracking of errors and change requests. An interesting finding revealed in Figure 3 is that the use of formal procedures is not associated with better intergroup coordination once one controls for the conditions under which they are used.

Figure 3 shows the factors that predict whether members of a project had a dense interpersonal network that extends beyond the project's boundaries. (See also Table 2.) In projects that were highly interdependent, members of the project by necessity knew many people outside their projects. In large projects, the interpersonal network was within the project and the average project member knew far fewer people outside of their project.[4] Finally, more certain projects have more extensive interpersonal networks.

The maintenance of an extensive interpersonal network was associated with better project outcomes. In particular, in projects in which members talked to others outside of the project, both project members and their management knew more about project status and commitments. This greater awareness aided their intergroup coordination.

Another interesting finding shown in Figure 3 is the statistical interaction between project certainty and interpersonal networks. This interaction means that the beneficial effects of interpersonal networks were most pronounced when projects were uncertain. This finding is consistent with organizational research—that informal, interpersonal communication is necessary for coordination primarily under conditions of uncertainty.

Predicting Managers' Assessments. Although not the primary focus of the study, it is interesting to note that senior managers' assessments of the quality of the software process and product that they managed was unrelated to either their staff's assessments of the success of their project's coordination, client's satisfaction with the company, or the software quality metrics.

Figure 3 and Table 4 show the factors that significantly predicted senior managers' assessments of the projects. Managers judged older projects that were self-contained and that relied heavily on formal, impersonal coordination techniques as more successful.

Discussion

The results suggest the value of both informal and formal interpersonal communication as mechanisms for sharing information and achieving coordination in software development. While much of the recent attention in software engineering has been on methods for formalizing communication among specialists, the data from this study suggest that, to be successful, these methods must at least be supplemented with interpersonal communication. Software development requires personal communication across functional boundaries to cope with uncertainty. Managerial and technical problems continually arise in the process of creating software, and while people can solve some of these problems themselves or by examining relevant documents, other problems demand information or cooperation from other people. An employee who doesn't understand the work flow that software was designed to support, who needs to decide which of two modules to change to fix a bug, who needs another's module changed, or who needs more information to clarify a functional specification document, generally requires help from someone else in the software development process.

The standard response when one confronts a problem that cannot be solved alone is to go to a colleague close by. However, not all the necessary knowledge, relevant viewpoints, or requisite power can be gathered from local colleagues, who are often consulted as much for convenience as for relevance or competence. Personal contact with those outside of the immediate supervisory group is one way to get this information. Thus, projects with denser cross-boundary networks were better informed and coordinated. Extensive personal networks were particularly beneficial in uncertain projects.

However, direct contact between organizational members is not a panacea for coordination problems in large software projects, both because of the excessive transaction costs resulting from the many pairwise conversations and because of the ephemeral nature of the information transferred in them. Thus, a large project will also need formal communication like requirements review meetings, design review meetings, and opportunities for customers to test software. These settings allow the diverse groups involved in the project to come together in a controlled way. In the present study, project members judged these meetings across functional areas to be especially valuable, relative to their use. Not all meetings were judged as valuable, however. In particular, meet-

[4]In these large projects one suspects that the role of gatekeepers is extremely important for maintaining contacts outside the project. Unfortunately, here we had no way to identify gatekeepers.

ings in which routine information was exchanged or that only involved members of the project team, such as status review meetings or code inspections, were judged as less valuable than one would predict from their extensive use.[5]

In the company described here, formal, impersonal modes of coordination, such as project documents, memos, and bulletins, milestone and delivery schedules, and mechanisms for monitoring and approving requests to modify the software were a standard part of the software development process and hence heavily used. Surprisingly, once one controlled for the conditions under which they are used, namely in larger, more certain projects that had moved beyond the planning stages, greater reliance on them was not associated with better coordination. Moreover, project staff viewed these formal coordination procedures as less valuable than one would expect from their extensive use. These findings suggest some of the limitations on formal procedures and processes.

Senior managers are likely to be major beneficiaries of formal project management procedures. These procedures are often used to enable senior managers to establish control and gain feedback about the software development process. In the company we studied, staff members often complained, for example, that software metric data and status reviews were used primarily by senior managers and had little impact on the day-to-day software development process. These senior managers rated most highly projects that were relatively self-contained (i.e., had few organizational dependencies) and about which they received standardized reports (i.e., the projects that relied heavily on formal project management procedures). We interpret this pattern to mean that they preferred projects where they thought they were in control. However, the managers may have been misled by an illusion of control, since their judgments of projects were unrelated to their customers' satisfaction or with their staff members' assessments of them.

Implications. One may draw both practical and theoretical implications from the results of this research. Our data suggest strongly that personal communication is important for successful coordination in software development, but it may be too expensive, too ephemeral, or too local to be an effective communication mechanism. Thus, without technological assistance, greater use of personal communication channels might in fact have deleterious effects on development. For example, if people are working near peak efficiency, creating more pairwise communications might result in overload. The challenge in software engineering should not be to devise methods to minimize personal communication as, for example, formal specification languages are intended to do. Rather a goal should be to make interpersonal communication more efficient and effective by remedying the problems of expense, ephemerality, and parochialness.

Consider, for example, the well-known tendency for informal communication to be mediated by physical proximity and, as a result, to be too local. Engineers typically get their information from people who are in their immediate area. Thus, more time is spent with neighbors than is productive. Software engineers need to acquire information and understanding from those who are remote and otherwise would be barred from the core of the development process, such as problem domain experts, users, vendors of computers and of auxiliary software, documentation specialists, or training personnel. The expertise that these specialists can bring to bear is potentially relevant throughout the development process, although in a waterfall development model, it is typically applied at only a few points. For example, users may be interviewed when the functional requirements are being defined and again during software testing, but may be totally uninvolved during the coding phases of development, when many of the substantive details of design are fleshed out.

There are both managerial and technological solutions to this problem. Both participatory design [16] and the hiring of domain experts to be part of the software development team are attempts to make the users' experience and perspective available throughout the project. However, as users become increasingly integrated into the software project, they may lose their distinctive point of view and hence their value. Other experts have advocated that samples of users frequently test intermediate versions of the software, so that the naive users' perspective is continually injected into the software development process [14].

Formal project meetings, in which different stateholders make or review highly consequential but uncertain decisions on functional requirements or software architecture, are an important mechanism currently used to insure that different points of view are represented. Decisions reached at such meetings have important consequences for the project as a whole. Yet these meetings have important limitations. The meetings themselves are often inefficiently run, they often have disproportionate attendance from local representatives, and the rationale for important decisions is often lost to people who didn't attend the meetings. These problems suggest that computer and communication tools for conferences or distributed meetings are likely to be useful, by opening up the meetings, making them more efficient, and providing an archive of the meeting (see [15, 17]) for recent discussions on computer-supported meetings).

In addition to recommendations for improving the software development process, this study has some implications for software engineering research as well. Our conclusions about coordination in software development are consistent with the large literature about coordination in organizations generally. Our findings about the importance of interpersonal communications for dealing with problems of uncertainty have been identified in organizations as diverse as hospitals [13], social services agencies [23], and research and

[5]It is important to point out, however, that participants' assessments of meetings is only one index of value. It is possible for a project to benefit from meetings, even though participants individually don't benefit or don't perceive the benefit. Thus, code inspections, for example, may reduce errors and therefore downstream expense, even though the benefit may accrue to maintenance organizations, and not to the initial developers.

development organizations [21]. We believe that the large literature on organizational theory, developed in non-software settings, has more relevance to software development than had been recognized previously in software engineering.

References

1. Adams, J.S. The structure and dynamics of behavior in organizational boundary roles. In M.D. Dunnette, Ed. *Handbook of Industrial and Organizational Psychology.* Rand-McNally, Chicago 1976, 1175–1199.
2. Allen, T.J. *Managing the Flow of Technology.* MIT Press, Cambridge, Mass., 1977.
3. Blau, P. and Scott, W.R. *Formal organizations.* Scott, Foresman, San Francisco, 1962.
4. Blazer, R. *Imprecise program specification.* Report ISI/RR-75-36, Information Science Institute, December, 1975.
5. Boehm, B.W. Software engineering. *IEEE Trans. Comput.,* (Dec. 1976), 1226–1241.
6. Boehm, B.W. *Software Engineering Economics.* Prentice-Hall, Englewood Cliffs, NJ, 1981.
7. Brewer, M.B. and Kramer, R.M. The psychology of intergroup attitudes and behavior. *Annual Review of Psychology 36,* 1985, 219–243.
8. Brooks, F.P. *The Mythical Man-Month.* Addison-Wesley, Reading, Mass., 1975.
9. Brooks, F.P. No silver bullet: Essence and accidents of software engineering. *IEEE Comput. Soc. 20,* (Apr. 1987), 10–18.
10. Culnan, M.J. Environmental scanning: The effects of task complexity and source accessibility on information gathering behavior. *Decis. Sci. 14* (1983), 194–206.
11. Curtis, B., Krasner, H., and Iscoe, N. A field study of the software design process for large systems. *Commun. ACM 31,* 11, (Nov. 1988), 1268–1287.
12. Fox, J.M. *Software and its Development.* Prentice-Hall, Englewood Cliffs, NJ, 1982.
13. Georgopoulos, B.S., and Mann, F.C. *The Community General Hospital.* Macmillan, NY, 1962.
14. Gould, J.D., and Lewis, C. Designing for usability—Key principles and what designers think. In *Proceedings of CHI'83.* ACM Press, New York, 1983, pp. 50–53.
15. Hiltz, S.R., and Turoff, M. *The network nation: Human communication via computer.* Addison Wesley, Reading, Mass., 1978.
16. Kyng, M. Designing for cooperation: Cooperating in design. *Commun. ACM 34,* 12 (Dec. 1991), 65–73.
17. McCloud, P. An assessment of the experimental literature on electronic support of group work: Results of a meta analysis. *Human-Comput. Interaction 7,* 3 (1992), 251–280.
18. Newcomb, T.R. *The Acquaintance Process.* Holt, Rinehart and Winston, New York, 1961.
19. Pelz, D.C. and Andrews, F.M. *Scientists in Organizations: Productive Climates for Research and Development.* Wiley, New York, 1966.
20. Travis, P. Why the AT&T network crashed. *Telephony 218,* 4 (January 22, 1990), 11.
21. Tushman, M.L. Special boundary roles in the innovation process. *Admin. Sci. Q. 22,* 4 (1977), 587–605.
22. U.S. General Accounting Office. *Contracting for Computer Software Development—Serious Problems Require Management Attention to Avoid Wasting Additional Millions.* U.S. Department of Commerce, National Technical Information Service, P880-105638, Washington, D.C., 1979.
23. Van de Ven, A.H., Delbecq, A.L., and Koenig, R. Jr. Determinants of coordination modes within organizations. *Amer. Soc. Rev. 41* (1976), 322–338.
24. Zajonc, R.B. Attitudinal effects of mere exposure. *J. Personality and Soc. Psych. 9,* 1968, 1–28.
25. Zipf, G.K. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, Cambridge, Mass., 1949.

About the Authors:

ROBERT E. KRAUT is a professor of social psychology and human-computer interaction at Carnegie Mellon University. Current research interests include organizational communication and the social impact of information technology, including office automation and employment quality, technology and home-based employment, the communication needs of collaborating scientists, the design of information technology for small-group intellectual work, and the impact of national information networks. **Author's Present Address:** Carnegie Mellon University, School of Computer Science, 5000 Forbes Avenue Pittsburgh, PA 15213; email: robert.kraut@cmu.edu

LYNN STREETER is a senior director at U.S. West Technologies and manages the applied research computing effort. Current research interests include speech perception and analysis, human/computer interface design, communications in large software projects, information retrieval, and information services. **Author's Present Address:** U.S. West Advanced Technologies, 4001 Discovery Drive, Boulder, CO 80303; email: lstreet@advtech.uswest.com