HUMAN–COMPUTER INTERACTION, 1995, Volume 10, pp. 249–292

Object-Oriented Analysis and Design in Software Project Teams

James D. Herbsleb, Helen Klein, and Gary M. Olson
University of Michigan

Hans Brunner
U S WEST Technologies

Judith S. Olson
University of Michigan

Joe Harding
Harding Consulting

James D. Herbsleb is a psychologist and computer scientist with an interest in empirical research on process and technology innovations in software engineering; he is a Member of the Technical Staff at Software Engineering Institute, Carnegie Mellon University. **Helen Klein** is a PhD Student in Computer and Information Systems at the University of Michigan; she is interested in object orientation and the user's model. **Gary M. Olson** is a psychologist interested in the contributions of cognitive, social, and organizational research to both the design and evaluation of computer systems; he is a Professor of Psychology and Director of the Collaboratory for Research on Electronic Work at the University of Michigan. **Hans Brunner** is a cognitive psychologist with interests in user–system dialogue management, multimedia learning systems, and the psychology of computer programming; he is Director of the Cognitive Sciences group in the Applied Research Department of U S WEST Technologies. **Judith S. Olson** is a psychologist and information scientist with an interest in computer support for group work; she is a Professor and Chair of the Department of Computer and Information Systems in the University of Michigan Business School and a Professor of Psychology at the University of Michigan. **Joe Harding** is an anthropologist with an interest in the ethnographic study of organizational culture; he is President of Harding Consulting.

CONTENTS

ABSTRACT

Software development poses enormous cognitive, organizational, and managerial challenges. In this article, we focus on two of the most formidable of these challenges and on the promise of object-oriented (OO) technology for addressing them. In particular, we analyze the claims made about OO design (OOD) and (a) dissemination of domain knowledge and (b) communication and coordination. In order to address the validity of these claims, we conducted an in-depth observational study of OOD in an industrial setting as well as a series of interviews with experienced OOD practitioners. Compared to similar projects using traditional methods, our study found evidence in the OOD project for a reduced need for clarification in design discussions; differences in participation, in how meeting time is spent, and in the sequential order of design discussions; and a much greater

tendency to ask *why* questions. We discuss the implications of these findings for tools, grain size of design units, interactions with clients, and organizing for OOD.

1. INTRODUCTION

The design of large, complex software systems is necessarily a social process. There are many ways in which people can be organized to do this, and Brooks (1975) stressed how difficult and costly the coordination of people can be. In modern organizations, ad hoc teams or work groups are a standard organizational solution to assembling a collection of individuals with the knowledge and skills required to carry out nonroutine tasks. In our experience (Herbsleb & Kuwana, 1993; Kuwana & Herbsleb, 1993; G. M. Olson, J. S. Olson, Carter, & Storrøsten, 1992), ad hoc teams are the most common way in which software analysis and design are carried out for all but the tiniest of software projects. But the gap between the general prescription to form such teams and the effective use of such teams in real work settings is very large. This is because design as an activity and software as a product are among the most difficult intellectual endeavors attempted by organizations.

In this article, we focus on those issues that are at the intersection of cognitive and social processing in design teams. There are many reasons why socially distributed intellectual work is difficult. A team must develop shared understandings of the problem, of the application domain, and of the emerging design. The overall task must be divided into parts so that individuals or subgroups can work on them in parallel. Change must be managed and knowledge of changes disseminated. Individual efforts, which are partial or subproblem solutions, must be integrated, and the entire design must be checked for consistency. All of this requires extensive communication–not only by means of written and spoken language but also by means of various artifacts such as diagrams and prototypes. The internal workings of individual minds and the rich artifacts in the environment are important objects of study, because they are the key elements in carrying out the task. Studying the pattern of interactions among these components, however, provides a better overall description of the activity as a whole.

One way to talk about these factors is to use Hutchins's (1990) description of them as systems of distributed cognition. Hutchins has studied in great detail such routine but difficult intellectual tasks as team navigation aboard Navy ships (Hutchins, 1990) and flight operations in an airline cockpit (Hutchins, in press). Hutchins has examined the details of how the individuals and the components of their material environment work together as cognitive systems. We do not provide the same kind of detailed ethnography for our materials as he has, but we find the general metaphor

of teams and their tools as systems of distributed cognition to be useful in framing some of the issues.

Because software analysis and design are recognized as being extremely difficult intellectual challenges, modern software engineering methods attempt to provide various frameworks and tool sets for organizing the work. One family of methods that is receiving considerable attention is object-oriented (OO) methods. Numerous books and articles have appeared recently providing prescriptions on how to use OO methods for the analysis and design stages of software development. These proposals are especially interesting because these methods are often proposed as good solutions to just those problems of distributed cognition that make team work difficult.

In the remainder of this introduction, we characterize the current state of knowledge about how object orientation addresses the team aspects of software development. We do this by reviewing:

- The well-known problems of team software development.
- The reasoning behind claims that OO design (OOD) technology helps to solve these problems.
- The existing evidence that bears on these claims.

As we see later, the conclusion of this review is that the major problems of team development are communication and coordination, acquiring and sharing domain knowledge, and finding an effective organizational schema. For each of these issues, it is possible to construct plausible arguments for the advantages of object orientation. There is not much relevant evidence, however, and the evidence that exists is not uniformly supportive.

This review sets the stage for the remainder of the article. We use data from seven sources–including projects in which OO methods were used and others in which traditional methods were used–to examine the role of object orientation in team software development. Our goals are to compare the team aspects of traditional and OO software development and to evaluate the claims made for the superiority of object orientation.

1.1. Major Problems Faced by Software Design Teams

Communication and Coordination

There is substantial evidence that problems in communication and coordination are among the most troublesome and pervasive in software development (e.g., Curtis, Krasner, & Iscoe, 1988; Krasner, Curtis, & Iscoe, 1987; Walz, Elam, & Curtis, 1993). Furthermore, the ability of teams and organizations to communicate and coordinate effectively has a major impact on their ability to respond to other challenges, such as fluctuating

requirements (Curtis et al., 1988). It is also clear that subtle features of the work process (e.g., informal channels of communication) are extremely important, particularly when projects are uncertain (Kraut & Streeter, in press). These features are likely to be very important to the bottom line, because poorly coordinated projects tend to be the ones in which the eventual users are not satisfied with the product (Kraut & Streeter, in press).

Another indication of the importance of communication and coordination in software development comes from G. M. Olson et al.'s (1992) study of how software development teams from four projects in two organizations spent their time during meetings. It was found that about one third of the total time was taken up with clarifying what was meant in response to explicit questions. Furthermore, almost one fifth of meeting time was taken up with meeting management and project management. These numbers indicate that improving the efficiency of communication and coordination holds enormous potential for savings.

In many contexts, communication through a variety of shared representations (e.g., diagrams, checklists, text, and drawings) is crucial. The importance of shared representations has been supported by informal empirical studies, particularly in the context of user interface design (e.g., Karat & Bennett, 1991). Careful laboratory studies of groups performing design tasks have shown a significant improvement in the outcome when a simple shared text editor was used in a face-to-face design task, as compared to the outcome using only traditional tools like paper and whiteboard (J. S. Olson, G. M. Olson, Storrøsten, & Carter, 1993). So, there is ample reason to suspect that shared representations–and the tools that allow people to interact with them–are important in organizing software development efforts. Nevertheless, it is unclear, in our current state of knowledge about team software development, precisely what artifacts need to be shared and at what stages the sharing should take place.

Capturing and Using Domain Knowledge

One of the major challenges to a distributed cognitive system engaged in software design is effectively capturing and propagating domain knowledge through the development team. Malhotra, Thomas, Carroll, and Miller (1980) observed that interactions between clients and designers tend to occur in cycles as the client discusses goals and the designer tries to refine them and move them toward a concrete solution. In a study of upstream software development activities, Walz et al. (1993) observed the difficulties software designers have in capturing users' task knowledge. Important aspects of the domain and of users' tasks very frequently went unrecorded and were forgotten–only to be raised again, much later. One major cause of this was lack of

application domain knowledge that would have allowed the team to recognize the importance of information contained in orally expressed requirements and scenarios.

Curtis et al. (1988) identified the thin spread of application domain knowledge as one of the most significant and pervasive problems in large-scale software development. Although individuals often understood the small part of the domain that was directly relevant to their own software design work, the individual who was in command of the overview and able to integrate this knowledge was rare, highly valued, and known to exert enormous influence over the design. The basic difficulty is nicely summarized in the often quoted statement by one of the system engineers interviewed by Curtis et al.: "Writing code isn't the problem, understanding the problem is the problem" (p. 1271). Or, as Brooks (1987) put it, "The hardest single part of building a software system is deciding precisely what to build" (p. 17).

The problems arising from an inadequate understanding of the application domain were observed by Curtis et al. (1988) at the team, project, and organizational levels as well as the level of individuals. In particular, managers who were unable to keep their domain knowledge current reported difficulties. Additional problems emerged when different organizations collaborated, because they often had incompatible views of the domain, based on their own particular experience and expertise. Curtis et al. concluded that managing learning of the application domain is "a major factor in productivity, quality, and costs" (p. 1275).

Realizing the difficulty of the task of understanding what one should design is apparently one of the hallmarks of software design expertise. Jeffries, Turner, Polson, and Atwood (1981) found that expert software designers tended to devote more effort to understanding a problem than did novices. Novices typically produced suboptimal designs, in part because they rushed from an inadequate understanding of the problem into development of the first solution that occurred to them.

Further evidence of the importance of domain knowledge arises from an examination of minutes and videotapes of large software development projects in three organizations. Herbsleb and Kuwana (1993; Kuwana & Herbsleb, 1993) examined the questions that software engineers asked one another during meetings that took place during the software requirements and design stages. Herbsleb and Kuwana found that by far the most frequent type of question was aimed at trying to find out what the software was supposed to do. Questions aimed at understanding user scenarios were also very frequent, particularly early on in the definition of requirements. These results indicate that a sufficiently detailed knowledge of the application domain and how the software will function in that domain were the major sources of the designers' uncertainties.

Organizational Matters

Another of the persistent problems in software development is how project teams should be organized (e.g., Brooks, 1975). Organizational theory supplies many possible ways to structure project teams–from traditional authoritarian hierarchies to unstructured individualism (see, e.g., Constantine, 1993). It is often suggested that factors such as the certainty of the project (Galbraith, 1973) should guide the selection of organizational structure and coordination mechanisms. For several historical and practical reasons, there has been relatively little empirical work comparing the efficacy of various organizational structures for software development teams (Curtis & Walz, 1990).

From the perspective of distributed cognition, one of the most important aspects of team structure is how it supports the creation, sharing, and manipulation of representations, as nicely illustrated by Hutchins's (1990) navigation example. In order to coordinate the navigation task, the information must be propagated by the team across representations in several different media (e.g., instrument readings, logs, human voice) before ending up as plots on a navigational chart. Although the task of software development is much less routine than navigation, it also depends crucially on the propagation of information across shared representations (e.g., requirements documents, design documents, diagrams, plans, prototypes, scenarios, source code).

In addition to routinely accomplishing the immediate task, the organization must arrange for training new personnel and for effective responses to situations in which part of the process may break down. Both the capacity for sharing representations and essential redundancy are made possible by "open" tools and an "open" style of work (i.e., arrangements are such that others can readily observe the execution of tasks). The less experienced members can learn simply and naturally by watching the open interactions and the uses of open tools that are going on around them. Team members are also in a position to guard against one another's mistakes when someone in a particular role is overloaded or distracted. These are very general considerations that point out the potential organizational importance of open tools and open interactions.

1.2. Claims About Object-Oriented Design and Teams

Advocates of OO software engineering methods claim many advantages for this approach. Many of these claims concern the technical merits or cognitive advantages (Rosson & Alpert, 1990) of OOD. Several of the most important claims about OO, however, address the team aspects of software design–the topic of this article. We begin with a

brief description of the distinguishing properties of OOD, which are the basis of the claims.

The two major differences between traditional methods and OO methods are OO-style abstractions and the central role played by domain modeling in OOD. The abstractions used in OOD differ from other kinds of abstractions used in software design in that objects contain both data and behavior, whereas abstractions in traditional methods tend to focus on either data structures or processing but not both. OOD abstractions can therefore encapsulate functionality inside the object (i.e., functionality is available externally only through a well-defined interface). In addition, OOD abstractions support inheritance relations, whereby objects acquire attributes and behavior from abstractly defined classes.

Traditional methods and OOD differ not just in the primitives themselves but also in the particular way these abstraction primitives are used in the design of a software system. OOD typically begins with a model of the application domain, borrowing the concepts and vocabulary of users and domain experts. This structure, insofar as possible, is left intact in the design and even in the code itself. The claims discussed in the following sections derive from the nature of OOD abstractions themselves as well as from the style of development centered around a domain model.

Claim: OOD Enhances Communication and Coordination

Because a single representation underlies all stages of development, communication among all the participants (i.e., users, designers, implementers, maintainers) is claimed to be enhanced. Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen (1991), for example, claimed that the "greatest benefits [of OOD] come from helping specifiers, developers, and customers express abstract concepts clearly and communicate them to each other" (p. 4). Similar remarks can be found in many other books advocating OOD methods (e.g., Coad & Yourdon, 1991, p. 3; Jacobson, Christerson, Johnsson, & Overgaard, 1992, p. 43; Martin & Odell, 1992, p. 34; Wirfs-Brock, Wilkerson, & Wiener, 1990, pp. 10–11). The basic idea is that, regardless of where one participates in the development process, there is a core of the software design that remains the same and provides a common vocabulary and set of concepts with which to discuss the system. Because this conceptual core has its origins in the problem domain, it is ground with which users are also familiar. So, it improves their ability to understand and provides a basis for their participation.

Communication and coordination should also be enhanced, goes one argument, by the property of encapsulation of objects. As Wirfs-Brock et al. (1990) pointed out, OOD encapsulation should reduce interactions among parts of the system and allow the work to be distributed over developers more easily. Designers do not need to communicate or coordinate their activities with respect to the internal details of objects but should

be able to focus their discussions on defining their functions and interfaces. In ideal circumstances, so goes the argument, the need to communicate and coordinate may even be reduced.

The claim about enhancing communication receives some support from Bruegge, Blythe, Jackson, and Shufelt's (1992) informal case study. In the study, a class of 30 software engineering students collaborated on a single project for one semester. Object modeling technique (OMT) diagrams were used to ground discussions of alternatives within groups, and they often and effectively served as the medium of communication among groups. Perhaps most significantly, the clients, who were city and county planning officials, had no training in the semantics of OMT and yet were able to understand the models and even to suggest useful changes to the design and present them in the form of OMT diagrams. Bruegge et al. also found substantially reduced intergroup communication (measured in terms of use of an electronic bulletin board during system integration) compared to a similar but smaller project that used a combination of structured analysis, structured design, and OOD techniques. Although software development by students as a class assignment has many obvious differences from actual industrial development, these data are interesting because the two projects were quite similar in most respects except design method.

Although the use made of domain models in OOD is claimed to benefit communication, developing a domain model presents its own potential communication problems because developers who are ordinarily domain novices must communicate with and learn from domain experts. Research on expertise shows several dramatic shifts as one moves from being a novice to being an expert (Chi, Feltovich, & Glaser, 1981). First, compared to novices, experts tend to organize their knowledge in very different ways (e.g., McKeithen, Reitman, Rueter, & Hirtle, 1981), and these ways of organizing tend to be deeper and more abstract. One should expect that developers will approach the modeling task with an incorrectly structured model of the domain–rather than just an incomplete one. Second, the knowledge of experts tends to be procedural in form rather than declarative (e.g., Anderson, 1983). This means that it is more skill-like than fact-like, it is hard to verbalize, and it makes heavy use of recognition rather than analysis. This kind of implicit, procedural knowledge is often very difficult to communicate. Simply talking about the domain is not sufficient. These expert–novice differences strongly suggest that acquiring the knowledge necessary to construct an accurate domain model and to use such a model effectively in support of communication is a very challenging task.

Claim: OOD Helps to Propagate Application Domain Knowledge

The basic argument that OO technology helps to spread domain knowledge through the design team is that, because the shared representation

has its basis in the concepts and vocabulary of the application domain, knowledge of this domain is propagated throughout the project as the participants in the development effort construct the domain model. It is not expected that everyone will become a domain expert, of course, but, if the model accurately reflects the application domain, and the design team understands the model, then, arguably, a significant core of knowledge should be disseminated.

Lubars, Potts, and Richter (1993, p. 13) made this sort of argument when they claimed that development practices based on domain models are one way of increasing the domain knowledge of a project team. OOD textbooks variously describe this claimed advantage as encouraging "software developers to work and think in terms of the application domain through most of the software life cycle" (Rumbaugh et al., 1991, p. 4), enabling "the designer to think like an end user rather than to think like a computer" (Martin & Odell, 1992, p. 34), and encouraging "designers and programmers to begin thinking about the real-world aspects of a problem as early as possible" (Wirfs-Brock et al., 1990, p. 14).

There is some very informal evidence tending to show that this approach can be successful. For example, Champeaux, Anderson, and Feldhousen (1992) reported that designing by refining analysis models was generally viewed as "providing an invaluable anchoring of system design to the problem being solved" (p. 384). Champeaux et al. reported that this was regarded by the developers themselves as "a true breakthrough" brought about by OOD techniques.

This mode of learning about the domain assumes that OOD abstractions provide a suitable medium. There is research that sheds some light on the role of objects in human cognition, and it seems to suggest that object-based representations may be better than representations based on other sorts of primitives. In careful experiments, Gentner (1981; Gentner & France, 1988) showed that, when people are asked to repair a simple sentence with an anomalous subject–verb combination, they almost always change the verb and leave the noun as it is, independent of their relative positions. This suggests that people take the noun (i.e., the object) as the basic reference point. Models based on objects may be superior to models based on other primitives, such as behaviors.

In work relevant to how easily designers can use inheritance hierarchies of objects and classes to represent domain knowledge, Miller (1991) described how nouns and verbs differ in their cognitive organizational form. Nouns–and hence the concepts associated with them–tend to be organized into hierarchically structured taxonomies, with class inclusion and part–whole relations as the most common linkages. These are also, of course, the most common relations in OO representations. In human cognition, these hierarchies tend to be fairly deep for nouns–often six to seven layers. These hierarchies support a variety of important cognitive behaviors, including the inheritance of properties from superordinate

classes. In contrast, verbs tend to be organized in very flat and bushy structures. This again suggests a central place for objects, in that building inheritance hierarchies will mirror the way humans represent natural categories only if the basic building blocks are objects rather than processes or behaviors.[1]

On the other hand, human understanding of hierarchies tends to be organized around *basic-level* classes (i.e., intermediate levels of abstraction that form an anchor point for human classification and reasoning). As described by Rosch (1978; Rosch, Mervis, Gray, Johnson, & Boyes-Braem, 1975), basic-level categories have large numbers of differentiating attributes, whereas, at levels both lower and higher, the differentiating attributes are very modest in number. This means that, in human conceptual hierarchies, there are relatively few attributes at higher levels of abstraction to be inherited. The tendency in generating class hierarchies for inheritance is to push attributes and behaviors as high as possible. But, to the extent that this is successful, it will lead to hierarchies radically different from those that both users and developers have naturally. Furthermore, construction of abstract classes that accurately mirror the real world may entail the very difficult task of discovering new, deeper conceptual structure in the application domain.

Claims About How OOD Teams Should Be Organized

In the preceding two sections, we discuss claims about the inherent benefits of OOD. In this section, we look at claims about how an OOD development effort should be organized in order to realize these benefits. Many claims focus on the development organization as a whole–suggesting, for example, that traditional organizational cultures that focus on projects must be replaced with a culture that focuses on components (Meyer, 1992). This new organization may require new roles, such as *application engineer* (Nierstrasz, Gibbs, & Tsichritzis, 1992), to explicitly assign responsibility for extracting and representing domain knowledge in a useful form. New life cycle models have also been proposed to explicitly incorporate reuse of classes and a highly iterative development style (e.g., Henderson-Sellers & Edwards, 1990).

At the level of the development team, there have been several suggestions about appropriate structures and roles. Booch (1991), for example, identified four kinds of roles on the development team (i.e., system architects, class designers, class implementers, and application programmers). It is particularly crucial, of course, to ensure that a coherent system architecture is maintained throughout the process. It is very often recom-

1. Interestingly, a thesaurus, which attempts to provide an index to a broad range of knowledge and which is set up in a hierarchy, is organized by nouns.

mended that one or more highly skilled "visionaries" (Booch, 1991) who are widely respected by the team assume this role and that the project manager be "tightly coupled" to this group (Jacobson et al., 1992).

1.3. Goals of This Study

Not much empirical evidence exists that effectively tests the value of software methods (Fenton, 1993). As the preceding section shows, although there is some support for some claims, there is good reason for a healthy dose of skepticism as well. Our goals are to understand the impact of OOD on the distributed cognitive aspects of software design and to begin to assess the claims about the ways in which OOD is thought to enhance the functioning of software development teams. In particular, we look at:

- Patterns of communication and coordination, particularly in design meetings.
- Dissemination of domain knowledge, primarily as revealed by the questions developers ask one another.
- Design team organization and leadership.

We approach this task primarily with an in-depth observational study of software development in an industrial setting. We also compare these data, whenever possible, with data from our previous research on software development, in which traditional design methods were used. We also conducted interviews with other industry users of OO methods to help us put the findings from the observational study in perspective. The details of the data and of the methods used are presented in the following sections.

2. RESEARCH METHODS

2.1. Data

We address these questions with data from seven sources, some of which are new and some of which have been analyzed for other purposes elsewhere. Figure 1 summarizes the kinds of data we have from each source.

OOD Project. We observed an OO development project at U S WEST Technologies. The project used OOD techniques to develop an architecture to support development of a variety of distributed multimedia applications. We observed the project for 8 months–from project announcement to delivery of the Phase 1 architecture. The project group included a project leader, a chief architect, from 4 to 6 developers, and a

Figure 1. **Kinds of data available from various sources.**

	Data Source				
Project	Time Sheets of All Activities	Videotapes of Meetings	Minutes of Meetings	Weekly Interviews of Participants	General Surveys
OOD project	X	X		X	
Traditional project					
A		X			
B		X			
C		X			
D		X			
E (Japan)			X		
Nine OOD sites					X

few others called in for specific tasks, like testing and documentation. The group members varied widely in their experience with OO methods–from highly accomplished OO designers (e.g., the chief architect) to developers who were altogether new to OO. The chief architect had developed a system that was to provide some of the core design ideas in this system. As a result, he spent considerable time explaining the previous system to other team members.

The primary client for the project was internal. Because very high priority was placed on supporting a wide range of applications, as opposed to any single application, the client did not have much direct control over the requirements of the project. The intended users of the architecture were software engineers, so the (similarly trained) developers could imagine scenarios of use that were indeed accurate analyses of needs.

Three kinds of data were collected from this project. First, we obtained team members' time sheets reporting their weekly time spent on requirements, analysis, design, coding, testing, and other activities. Team members also noted how much of the time in these categories they worked individually or in a group.

Second, we obtained 93 videotapes and 41 audiotapes of the meetings that took place in this project–almost all of the encounters of more than one team member throughout the 8 months. Six videotapes of meetings were selected for in-depth analysis–meetings at which "design" was clearly taking place, not project management or mere presentations of work done elsewhere. The number of developers present ranged from 4 to 7. In addition, three design meetings–one from very early in the project, one from midproject, and one late in the project–were selected for extracting questions (to be described).

Third, each week we conducted semistructured interviews with all of the core members. We asked, among other things, what issues were

important during the prior week. We videotaped and took detailed notes of these interviews. The interviewee could request that the videotape be terminated for a period of time, and all material was kept in strict confidence, to be reported only in the aggregate. We call these interviews at U S WEST *weekly interviews* to distinguish them from the very different *field interviews* we conducted at other OOD sites.

Traditional Projects A, B, C, and D. We previously observed 10 meetings from four projects in which traditional software development methods were used. The sites in which these projects were observed were Andersen Consulting and MCC (G. M. Olson et al., Storrøsten, 1992; G. M. Olson et al., in press). These projects and their analyses provide here a comparison of some of the behavior in places where OOD is used and where traditional methods are used. Like the meetings selected for analysis described previously, these meetings were also primarily concerned with design, and they spanned many of the same kinds of topics, such as user interface design, network issues, and determining the feasibility of various requirements. They involved about the same number of people (3 to 7). From these projects, we have videotapes, which we analyzed in depth (as described later).

Traditional Project E. We previously also analyzed the kinds of questions that arose in 38 meetings of a traditional software development team in Japan (Herbsleb & Kuwana, 1993; Kuwana & Herbsleb, 1993). For these meetings, we have minutes rather than videotapes. The minutes were recorded by a scribe (a role that rotated among members of the development team). The minutes were generally recorded a day or two after the meeting by using detailed notes and other documents to reconstruct the discussion.

Comparability of OOD Project and Traditional Projects A Through E. Obviously, the conditions under which the software was developed in the six projects differ in many respects. We compared data from the projects for a variety of purposes. In order to aid the interpretation of these comparisons, we discuss here several of the similarities and differences that seem most relevant to the comparability of these data sources.

The most important attributes of these projects are summarized in Figure 2. Most of the applications were tools, but the OOD project and one traditional project were building architectures. Note that almost all were quite similar in the intended uses of the software, in that all but one were designed for other software engineers. Most projects, but not all, were staffed by developers who ranged from novice to expert in the development method being used.

Figure 2. **Comparison of project characteristics.**

	Project					
		Traditional				
Characteristic	OOD	A	B	C	D	E
Type of application	Distributed multimedia architecture	Client-server architecture	Reverse engineering software tool	Knowledge base editor	Software design tool	Software development environment
Intended users	Software engineers	Software engineers	Software engineers	Knowledge base editors, maintainers	Software engineers	Software engineers
Range of experience with methods	Novice to expert	All highly experienced	Novice to expert	Novice to expert	All highly experienced	Novice to expert
Leadership roles	Technical lead and chief architect	Manager	Manager and technical lead	Manager and technical lead	Manager	Combined role of manager and technical lead

The difference that stands out most in Figure 2 is the leadership roles in the OOD project. No other project had a role called *chief architect.* In trying to understand this role, it is important to note that the role called *technical lead* in the OOD project had much more of a management component than did this role in other organizations. For example, the technical lead was in charge of the weekly management-oriented status meetings but did not attend the weekly technical-issues meetings, which were run by the chief architect. The technical lead had the major responsibility for securing resources, planning, tracking progress, and setting milestones; the chief architect had the major responsibility for the overall and day-to-day technical decisions. So, the role of technical lead in the OOD project corresponds in many ways to that of manager in projects A through E. In some respects, the role of chief architect in the OOD project corresponds to that of technical lead in projects B, C, and E. As the role was interpreted and enacted in the OOD project, however, it also seemed to encompass a higher degree of responsibility for and ownership of design decisions, with the goal of ensuring system integrity.

Other OOD Sites. In order to add breadth to the data from our single OOD case study, we interviewed several people from other companies who had recently been through the experience of adopting OO methods.

We interviewed both developers and managers of OOD projects. We chose our interviewees according to the following criteria:

- Three or more years of experience using OOD techniques.
- Experience with systems through all phases of the life cycle of the systems.
- Experience with one or (preferably) more OOD development efforts of significant size (3 or more person-years).
- Use of some OOD development rationale or methodology.

The content of this interview was developed first by one 5-hr interview with one development consultant who had been involved with three medium-size OOD projects. From a large set of 54 potential questions, we constructed a smaller set of 22 questions, the focus of which was to elicit the interviewee's experiences relevant to the utility of OOD in addressing the problems of software development. Nine interviews were conducted with people from a variety of organizations (consulting firms, utilities, financial institutions, and manufacturing firms). These interviews, conducted either in person or by telephone, lasted an average of 3.25 hr. We refer to these as our *field interviews* in order to distinguish them from the very different *weekly interviews* discussed earlier.

Although the interviewees drew on an extensive body of experience with OOD in industrial settings, we want to point out that they were not unbiased observers. They were often OOD enthusiasts who were committed to these methods. They were in general reporting their impressions, which were not necessarily based on any objective measurements. Although we believe that this source of data is a very valuable adjunct to our detailed observational studies, one should bear these potential biases in mind when interpreting the results.

2.2. Analyses

Time Sheets. As mentioned earlier, in the OOD project we observed, the developers filled out weekly time sheets. The developers recorded the number of hours spent in requirements, analysis, design, coding, and testing. They also recorded how much time in each category was spent in individual work and in group work. For the purposes of data analysis, we combined the requirements and analysis categories because these are very closely related and because the number of hours in each was quite small relative to our other categories. When we refer to *requirements,* it should be understood to include requirements analysis as well as requirements acquisition. We calculated how many hours went into each type of activity overall for the project. We also divided the 32-week project into eight 4-week periods so as to examine how activities changed longitudinally.

Figure 3. **Brief descriptions of 22 categories of activity in design meetings.**

Category	Description
Issues	Major questions, problems, or aspects of the designed object itself that need to be addressed
Alternatives	Solutions or proposals about aspects of the designed object
Criteria	Reasons, arguments, or opinions that evaluate an alternative solution or proposal
Project management	Statements having to do with activity not directly related to the content of the design, in which people are assigned to perform certain activities, decide when to meet again, report on the activity (free of design content) from previous times, and so forth
Meeting management	Statements having to do with the orchestrating the activity of the meeting, indicating that group members are to brainstorm, decide (and vote), hold off on a discussion, and so forth
Summary	Reviews of the state of the design or implementation to date, restating issues, alternatives, and criteria
Walk-through	Gathering of the design so far or the sequence of steps the user will engage in when using the design so far, used to either review or clarify a situation; usually follows the user's task or the flow of data or messages inside a system architecture
Digression	Joking, discussions of side topics, interruptions
Goal	Statement of the purpose of a group's meeting
Other	Time not categorizable in any of the above categories
Clarification of _____ (12 types)	Questions and answers in which someone either asked or seemed to misunderstand; includes repetitions for clarification, associations, and explanations; clarifications serve to clear up misunderstandings from other individuals; each clarification is coded according to what is being clarified (e.g., clarification of issue, clarification of alternative), so there is one clarification category associated with each of the preceeding 10 categories; there are two additional types of clarification–clarification of artifact (stems from confusion about things like diagrams and design documents) and general clarification (not associated with any of the above categories)

Note. Adapted from G. M. Olson, J. S. Olson, Carter, and Storrøsten (1992).

Videotapes of Meetings. The design discussions were analyzed using the same categories and techniques used in previous research. Because these categories, their definitions, and our means of establishing their reliability were described in detail elsewhere (G. M. Olson et al., 1992), we only summarize the definitions briefly here (see Figure 3).

For the six meetings we selected for detailed analysis, we transcribed all discussion and categorized it according to the scheme outlined earlier. We analyzed the amount of time spent in each of the 22 categories of activity, the number of episodes of each type of activity, and the frequencies of transitions among these activities. In order to examine patterns of participation in these meetings, we also analyzed the amount of time spent talking for each participant–both as a total and by activity category.

In addition to analyzing activities and transitions, we also constructed design rationale graphs for each of the six OOD meetings (see G. M. Olson et al., 1992). In these graphs, we identify the issues, the alternatives proposed for each issue, and the criteria mentioned for each alternative (see Figure 7 for an example of a graph associated with a single issue). These graphs are not necessarily trees, because a criterion may bear on more than one alternative. Constructing these graphs enables us to compare some of the parameters of the OOD discussions with traditional-method meetings.

We also extracted and analyzed the questions developers asked of one another in a sample of three technical design meetings. Our coding scheme was presented in more detail elsewhere (Herbsleb & Kuwana, 1993), so we only summarize it here. We extracted all 229 questions from these meetings and, for each question, identified the *target*–the thing, event, or task about which the question was being asked. Targets form the basic unit for most of our analyses. As with previous data sets we have analyzed, about half (46%) of the questions had more than one target–for a total of 338 targets.

For each target, we identified the attribute about which the question was asked–whether who, what, when, why, or how was asked. Then we classified it according to the stage in the traditional software development life cycle when it would be created (i.e., requirements, design, implementation, testing, and maintenance). We used standard criteria drawn from industry guidelines and software engineering textbooks. Last, for the questions that had more than one target, we categorized the relations among the targets as follows. *Realize* is the relation between some function and the means of carrying it out (e.g., editing a piece of text might be a function carried out or realized by cutting, pasting, deleting characters, entering characters, etc.). *Interface* relations concern how targets communicate with one another. *Evolve* is a relation between a target and a subsequent version of the same target. A question about whether or not two targets are identical in some way has a *same* relation. Last, questions about persons assigned to tasks have *task assignment* relations.

Minutes of Meetings. All of the questions in the minutes of our sample of 38 design meetings at NTT Software Laboratories were extracted and categorized in the way already described.

Weekly Interviews. As mentioned, members of the U S WEST team were confidentially interviewed each week about the project generally and about the concerns that were most important during the prior week. Using detailed notes of these interviews, we tabulated the number of mentions of each concern.

Field Interviews. The interviewer took detailed notes of each interview, transcribed the notes immediately after the interview, and used the transcripts to write summaries of interviewees' answers to all of the questions. The responses were examined for common themes, and the number of interviewees who took a particular position was tabulated. The unit of analysis was 9 interviews rather than 13 interviewees because those interviewed together almost always expressed a consensus and cannot be considered independent sources of data.

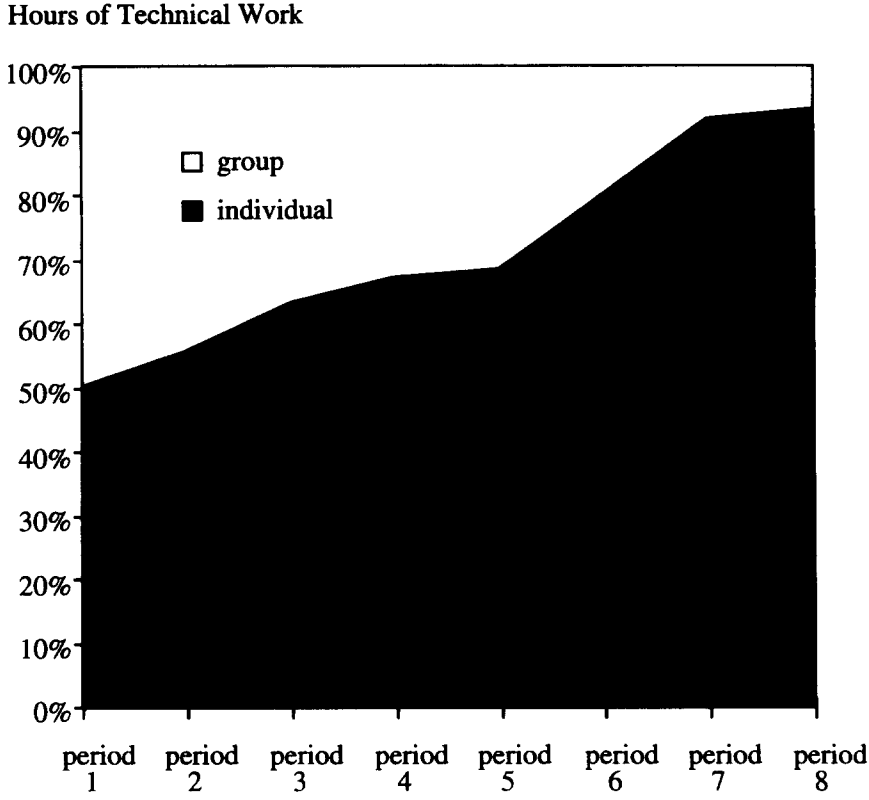
3. RESULTS AND DISCUSSION

3.1. Highly Collaborative Nature of Design

We begin by reporting some basic parameters about where and when the group work occurred in the project we observed. To begin to address questions about the demands of distributed cognition for openness in tools and communication, we examined how effort is allocated between individual and group work and how this mix changes for different tasks and over time. In this section, we report on data taken from weekly time sheets that asked these questions.

It is clear from Figure 4 that, over the life of the project, the overall percentage of group work (i.e., summed across categories of activity) was high but decreased fairly continuously from an initial value of about 50% of all technical work to only about 10% by the end of the project. Figure 5 shows one component of this change. Activities traditionally associated with later life cycle stages (coding, testing) tended to involve more individual work, whereas earlier stages (requirements, design) involved a larger proportion of group work. In addition, as shown in Figure 6, each type of activity also tended, in general, to move from a higher proportion of group work in the beginning to a higher proportion of individual work toward the end. As can be seen in both Figures 5 and 6, design activities tended to have the largest proportion of group work overall, and this was the case for almost every time period. Furthermore, as was not the case with other activities, in each period but the last more than half of the time spent on design was spent in group work.

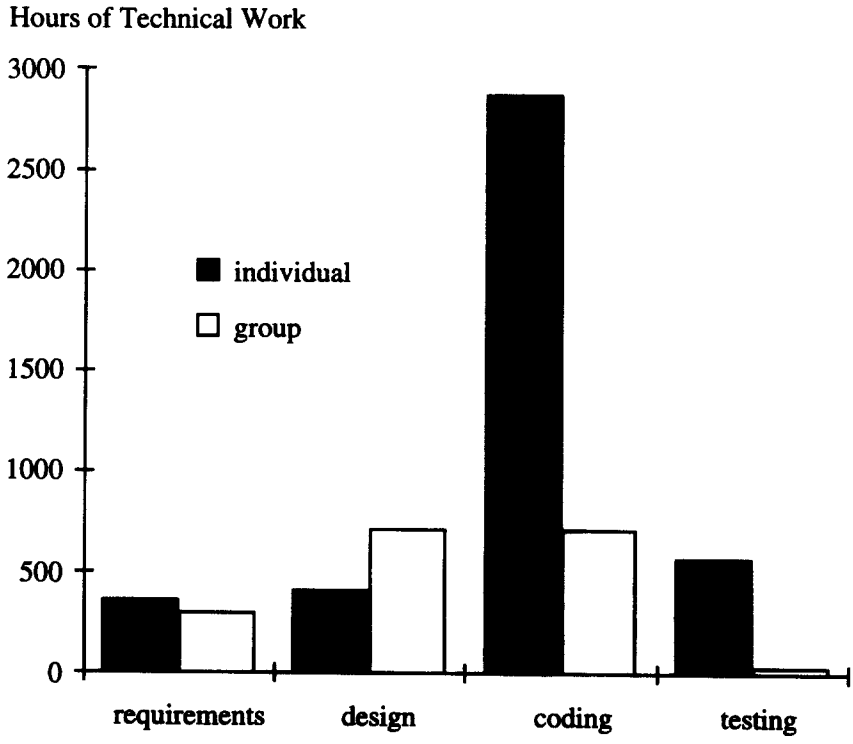
Figure 4. **Percentage of time overall spent in individual and group work. Each period is 4 weeks.**



3.2. Overall Issue Structure

An example of a design rationale graph representing the discussion of one issue at one meeting is shown in Figure 7. As already mentioned, we analyzed every issue that arose in a sample of 6 design meetings in this fashion. The basic parameters of issue discussions in the OOD project and in a similar analysis of 10 design meetings in which traditional methods were used (G. M. Olson et al., 1992) are summarized in Figure 8. We find the degree of similarity startling. Although design using OOD methods tends to involve a somewhat different distribution of design activities, as we see later, the basic parameters of how many alternatives are discussed for each issue are quite similar. The one difference evident in Figure 7 is a suggestion that the OOD meetings had fewer issues for which a larger number of alternatives (three or more) were considered.

Figure 5. **Hours spent in individual versus group work by activity.**



3.3. Communication and Coordination

We addressed communication and coordination issues mainly by looking at how developers spent their time in design meetings. We compared both time budgeting (i.e., time spent in, and number of episodes of, various design activities) and sequential structure of design activities. Many of the findings suggest important differences in design discussions when OOD and traditional methods are used. Some of these, primarily our findings about clarification, have relevance to claims about OOD and its role in supporting communication and coordination. Last, we report some results from our field interviews that bear on these issues.

The time spent in the various activities and the transitions among them are presented graphically in Figure 9. The area of the circle represents the amount of time spent in each category of activity. The thickness of each arrow indicates the frequency of transitions between categories. In order to reduce clutter, only those transitions that account for .7% or more of the total number of transitions are drawn. (The number of transitions rises dramatically just below this cutoff.) For purposes of comparison, a simi-

Figure 6. **Ratio of group work to individual work over time for each type of development activity. Values above the dashed line indicate more time spent in group work than in individual work. Each period is 4 weeks.**

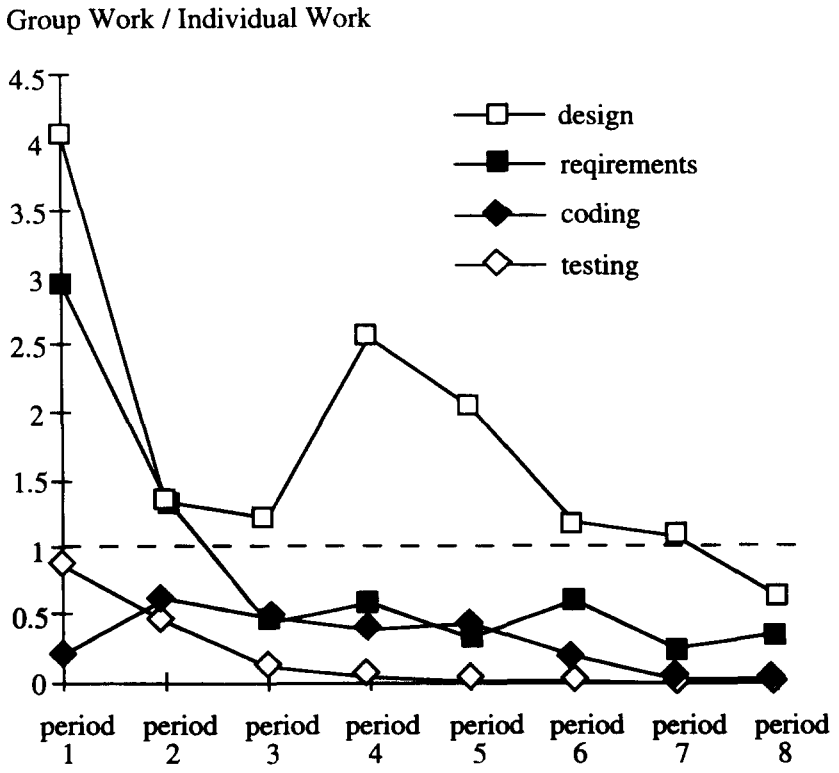


Figure 7. **Example of a design rationale graph.**

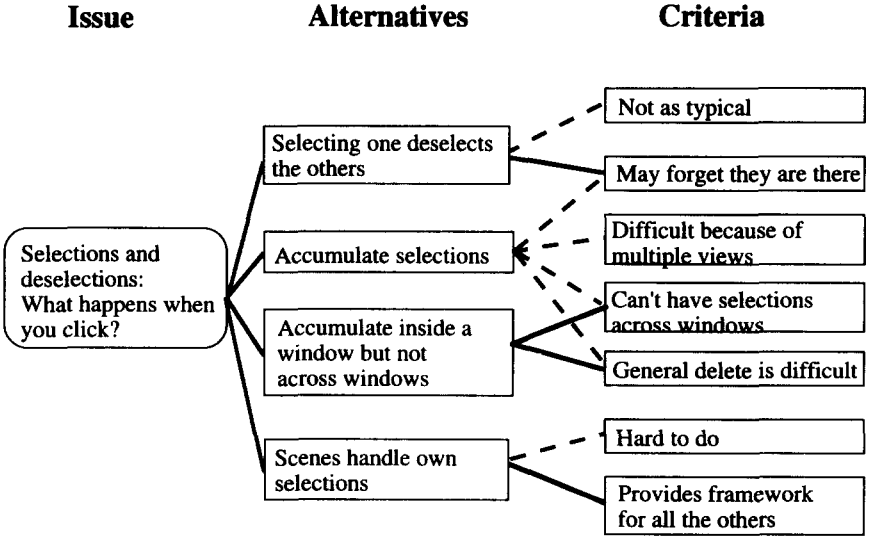


Figure 8. **Comparison of design rationale parameters of OOD meetings and traditional-method design meetings.**

Parameter	Meeting	
	OOD	Traditional
Median number of issues	10	10
Range of number of issues	6 to 24	1 to 44
Modal number of alternatives per issue	2	2
Percentage of issues with one or no alternative	25	21
Percentage of issues with three or more alternatives	29	40
Percentage of alternatives receiving any explicit evaluation	63	63

Figure 9. **Times and transitions in design activities in U S WEST OOD project.**

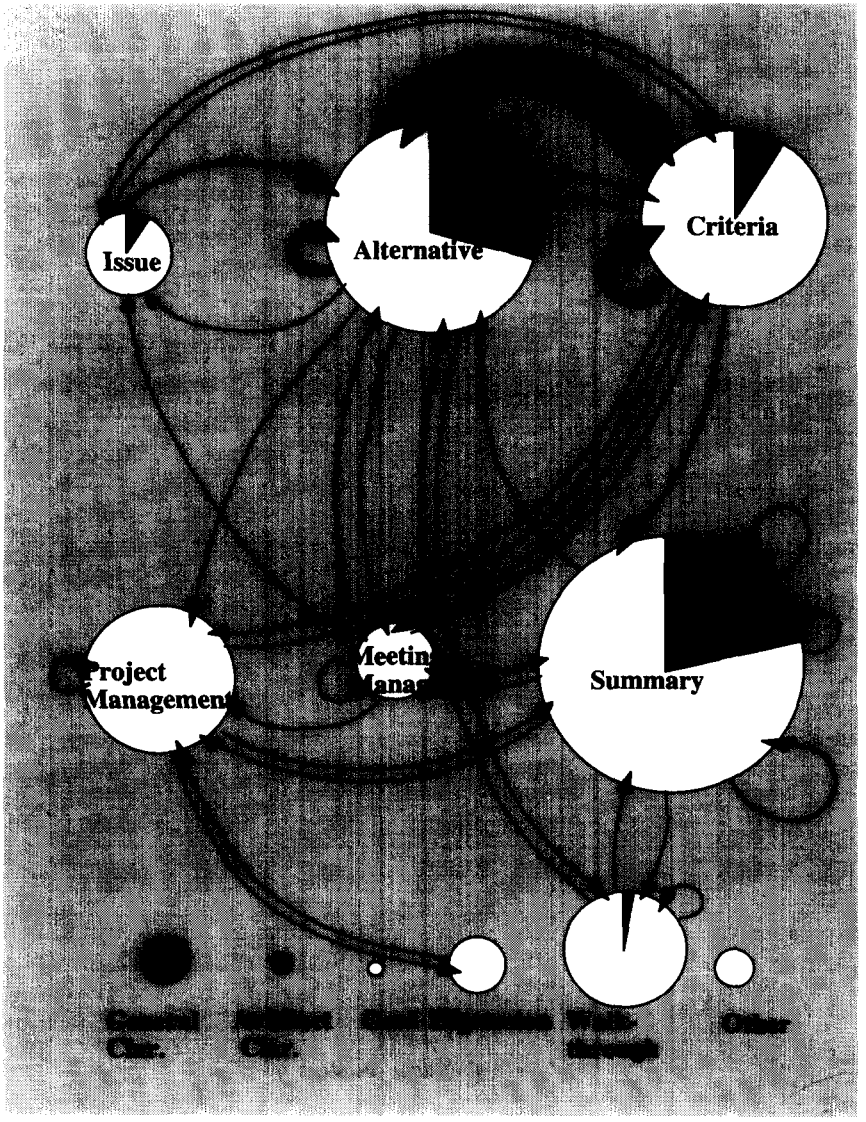
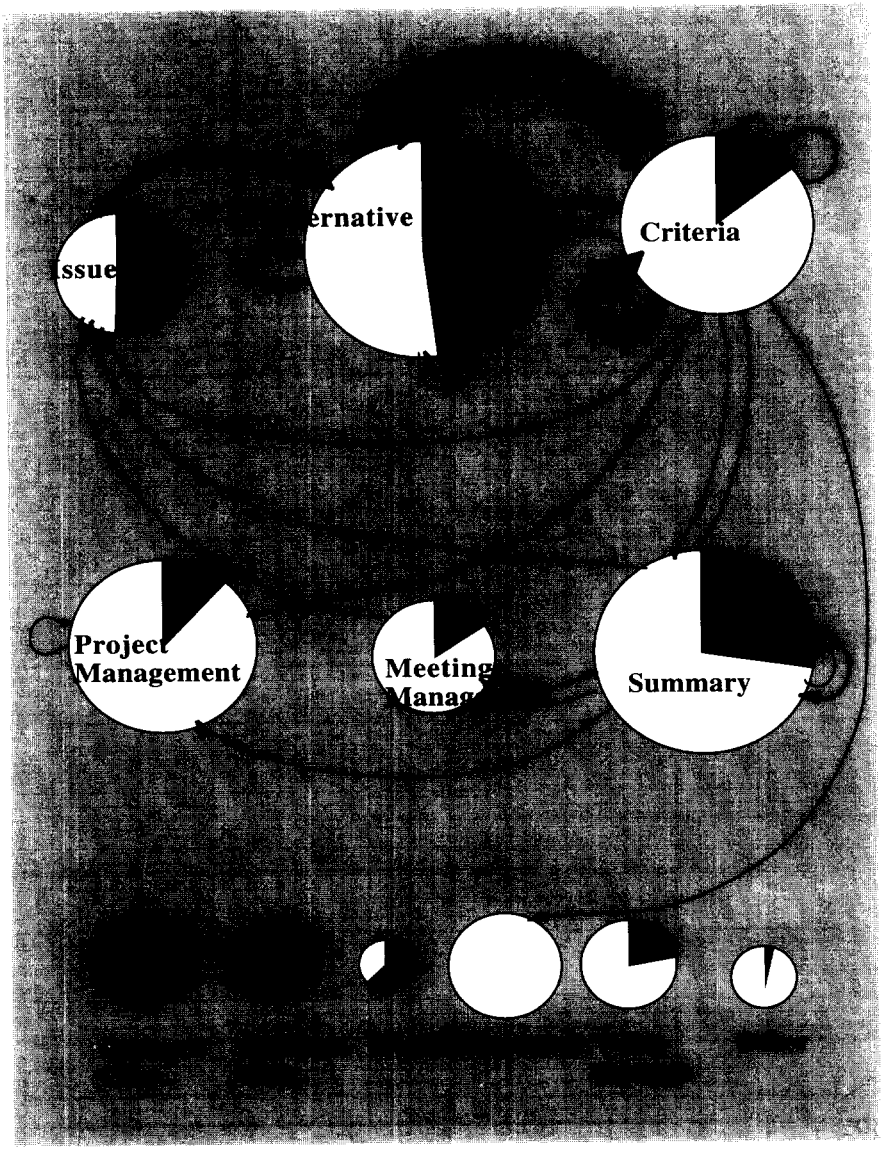


Figure 10. **Times and transitions in design activities in traditional projects.**



larly constructed diagram for 10 design meetings in which traditional methods were used is reproduced in Figure 10. In both cases, clarification is combined in the diagram with the clarified activity. In each circle, the white portion is the activity itself; the black portion represents clarification of that activity.

Our previous analyses of times and transitions in design meetings in which traditional methods were used revealed a remarkable degree of similarity across different organizations and projects (G. M. Olson et al., 1992). In broad outline, the times and transitions in the OOD meetings generally resemble this pattern, as one would expect given that the groups were engaged in essentially the same task. Work on design proceeds by statement of an issue and discussion of alternatives, interspersed with discussion of criteria that bear on the selection of alternatives. Departures from design work are primarily to summarize or to carry out some form of management activity.

Despite these broad similarities, there are several ways in which the OOD meetings stand in striking contrast to the relatively homogeneous set of traditional-method meetings. These differences are suggestive of how OOD may influence the design process. The major differences associated with OOD are:

- Fewer episodes of clarification in design discussions.
- More episodes of summary and walk-through.
- Earlier mentions of criteria in issue discussions.
- More integral role of summary and walk-through in design discussions.

Next we explore these differences in detail.

Time Budgeting and Frequency of Design Activities

Types of activity were distributed differently in meetings in which traditional methods were used and in meetings in which OOD methods were used,[2,3] $G^2(15) = 152.95$, $p < .001$. The standardized residuals (for the

2. The G^2 statistic is comparable in most cases to chi square. We used G^2 because it is superior in some ways for fitting hierarchical models. The difference reported here is also highly significant as determined by chi square.

3. We originally constructed a contingency table of 22 categories of activity × 2 conditions. An appropriate test of association would be a log-linear model that includes only main effects. An association between variables would then manifest itself as a significant interaction. We chose to eliminate the digression and clarification-of-digression categories as theoretically uninteresting. In addition, in order to calculate the appropriate statistics, it was necessary to partition the table and consider only those rows and columns that did not include clarification of goal, clarification of project management, clarification of meeting management, and clarification of other because these were very infrequent in the data set, and their inclusion generated many unacceptably small estimated values. (See G. M. Olson, Herbsleb, & Rueter, 1994, for a discussion of the application of log-linear modeling to sequential data. See, generally, Fienberg, 1980; Gottman & Roy, 1990.) Because of the necessity of partitioning the contingency table, we can claim a reliable statistical association only with respect to the frequencies of the remaining categories. We have insufficient data to reach any conclusions about the categories we removed from the analysis.

Figure 11. **Statistical association between design method and frequency of particular design meeting activities.**

Activity Category	Method	
	OOD	Traditional
Issue	-1.84	1.28
Issue clarification	-2.13	1.48
Alternative	1.51	-1.05
Alternative clarification	-2.85	1.98
Criteria	-0.74	0.52
Criteria clarification	-3.11	2.17
Meeting management	1.61	-1.12
Project management	-1.47	1.02
Goal	-1.18	0.82
Summary	4.94	-3.43
Summary clarification	0.38	-0.26
Walk-through	4.53	-3.15
Walk-through clarification	-1.76	1.22
Other	-2.61	1.82
Artifact clarification	-1.83	1.27
General clarification	-2.23	1.55

Note. Numbers are standardized residuals for the main-effects model.

main-effects model) give us an indication of which activity categories contribute most heavily to this difference (see Figure 11). A particular category contributes powerfully to the difference if its values for the two conditions have standardized residuals with absolute values greater than 1 and are different in sign. From Figure 11, one can see that, with the exception of summary clarification, on which the two conditions do not differ much, each type of clarification[4] contributes to the difference, and each is in the direction of higher frequency in the traditional meetings.

Examination of the residuals in Figure 11 reveals other categories that contribute substantially to the overall difference in number of episodes in various categories of activities. We generally think of issue, alternative, and criteria as the core design categories, as work in design meetings generally progresses by posing and resolving design issues. In these categories, compared to those of the traditional-method meetings, the OOD meetings had more episodes of alternatives, fewer episodes of issues, and a similar number of episodes of criteria. Another important departure from the pattern we have seen in traditional-design discussions is a much greater number of episodes of walk-through and summary in OOD meetings.

4. We can conclude nothing, of course, about the clarifications not shown in Figure 11 because we have insufficient data for these particular categories.

In our sample of meetings, we can see a very similar pattern in the amount of time spent in each category of activity. Because the unit for this analysis is the meeting, we have only 16 observations (10 traditional, 6 OO) for each comparison. Because of this small sample size, it is not surprising that t tests do not reveal any statistically reliable differences. In the meetings we examined, overall there was substantially less time spent in clarification in the OOD meetings (14%) than in our previously analyzed meetings (33%). Also in accord with the analysis mentioned earlier, a much larger proportion of time in OOD meetings was spent in summary and walk-through. Together, summary and walk-through accounted for almost twice the proportion of time in the OOD meetings (about 40%) as in the traditional meetings (about 22%).

The overall pattern of results suggests that the quality of communication among members of the team was very high in general. There were substantially fewer episodes of asking and answering questions about what was meant. This is particularly significant in light of the fact that the vocabulary and concepts used to describe the design were at the outset unfamiliar to everyone except the chief architect. And, it seems very unlikely that the relatively small proportion of clarification can be attributed to simplicity of the design issues. The technical issues involved in designing a general architecture-level solution for the synchronization, dynamic linking, network transport, data consistency, and user interface problems inherent in distributed multimedia applications were extremely complex.

Transitions Among Design Activities

Figures 9 and 10 give a global view of transitions among design activities in OOD and traditional-design meetings. This view shows that the pattern of transitions among categories was broadly similar in the two data sets. With traditional methods, design discussions began with issue and iterated through alternatives–the discussion of each alternative often mentioning one or more relevant criteria. Summary tended to follow criteria and precede issue, thus appearing to serve to terminate discussion of one issue and lead to discussion of the next. This suggests that summary functioned as a transitional category between design discussions.

The OOD meetings were significantly different in their distribution of transitions,[5] $G^2(47^6) = 144.41$, $p < .001$. Again, we can examine the stan-

5. The full table for this analysis is 22 antecedent categories × 22 consequent categories × 2 conditions (traditional, OO). But, we were unable to analyze the full table statistically because there were several categories of activity that were very infrequent in both conditions, generating a table with many estimated values less than 1. We selected only the eight most frequent categories of activity–issue, alternative, criterion, clarification of alternative, meeting management, project

dardized residuals generated by the main-effects model to identify which transitions were most responsible for the statistical association and how the frequencies of these transitions interacted with type of design method. Again using values greater than 1 and different in sign as an indicator of major contributors to the statistical association, we find several differences.

One set of differences in the sequential structure surround occurrences of issue. In the traditional meetings, there is more of a tendency to move from statement of an issue to an alternative. In the OOD meetings, on the other hand, issue episodes tend to lead more often to discussions of criteria and summary and directly to clarification of alternative. This suggests that issues are sometimes addressed by first figuring out what properties a good answer should have (criteria) before considering alternatives–a pattern quite unusual in traditional design. This interpretation is reinforced by the fact that there are also more transitions in OOD from criteria to alternative and to clarification of alternative.

Another set of differences in sequential structure surround summary and walk-through. In the traditional meetings, criteria are more likely to be associated with walk-through and summary, both leading to and back from criteria. The sequential relations also suggest that summary and walk-through frequently play a transitional role, often leading to some sort of management activity. In the OOD meetings, the connection of issue and clarification of alternative to summary suggests that summaries play a more integral role in exploring design issues and alternatives, rather than criteria. These results are consistent with the idea that walk-through and summary have different roles in the two sets of meetings.

Communication and Coordination at Project Level

An important theme in the responses of our field interviewees was that OOD helps to focus communication–primarily on decisions affecting object interfaces and placement of methods. These answers suggest that OOD may ease coordination by helping developers work independently

---

management, walk-through, and summary–to use in the model. So, our conclusions about a statistical association include only transitions among the categories in this model; we have insufficient data to draw statistical conclusions about the others. This reduced table has 8 antecedent categories × 8 consequent categories × 2 conditions. Because we are interested in testing for the presence of an association between sequential structure (i.e., the interaction of antecedent and consequent categories) and design method, the appropriate model includes all three main effects and all three two-way interactions (see G. M. Olson et al., 1994).

6. The degrees of freedom were 47 instead of the 49 one would expect given an $8 \times 8 \times 2$ table because, despite our partitioning, there remained one 0-marginal value, which, of course, resulted in a row with both cell entries of 0 and necessitated the reduction in degrees of freedom. (See Fienberg, 1980, pp. 140–142.)

and identify what needs to be communicated. This, of course, is one of the intended effects of encapsulation in OOD.

Our interviewees unanimously endorsed OOD as a technique for avoiding coordination problems while accommodating change. Several causes of this advantage were identified by our interviewees. An obvious one is encapsulation, which was generally successful in isolating internal changes in an object or class from other objects and classes. Also, using a domain model to structure the software was perceived as an aid to quickly identifying those places where changes had to be made.

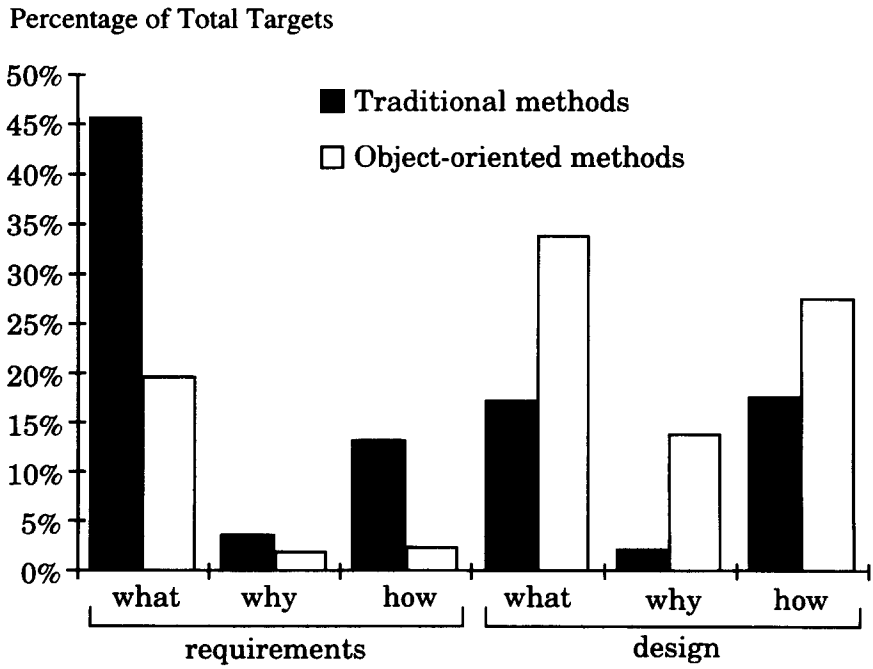
3.4. Knowledge Dissemination

In previous work (Herbsleb & Kuwana, 1993), we noted that there were many proposals for making various kinds of information available to software developers, including design rationales, information about the application domain, and user scenarios. There are few data, however, on what kinds of information developers actually need in order to do their jobs. To investigate this question, we extracted all of the questions that software developers asked one another in a sample of development meetings in Japan (traditional project E) and in the United States (traditional projects A, B, and C). We found a very surprising degree of similarity in questions from these very different sources. One of the consistent findings was that developers tend to ask few *why* questions (5% to 6%) about the requirements or about the design. The most frequently asked type of question concerned what the requirements were, but there were also many questions about user scenarios, about defining modules and their interfaces, and about the detailed design of modules.

Against the background of previous results, where we found an astonishing degree of similarity in data from very different sources, the results from the OOD meeting are in some ways similar but in several respects markedly different. The distribution of attributes differs significantly from our previous results,[7] $\chi^2(4) = 35.43$, $p < .0001$, primarily because many more *why* questions were asked in the OOD meetings. In fact, the percentage of *whys* (15%) is roughly three times the percentage in the other two data sets (5% to 6%). The distribution of stages also differs reliably, $\chi^2(2) = 59.59$, $p < .0001$. The major difference is a much smaller proportion of questions about things traditionally created in the requirements stage and a much larger proportion concerning design-stage targets. As in our previ-

7. We show three data sets in the figures in order to make the similarity of the two sources of data from traditional meetings clearly visible. The statistics reported in the text, however, are a direct comparison of OO and traditional methods, which collapse the U.S. and Japanese traditional-method data. The results are comparable (i.e., highly significant) whether or not the data are collapsed in this way.

Figure 12. **Cross-tabulation of most frequent attribute–stage combinations in targets taken from OOD and traditional meetings.**

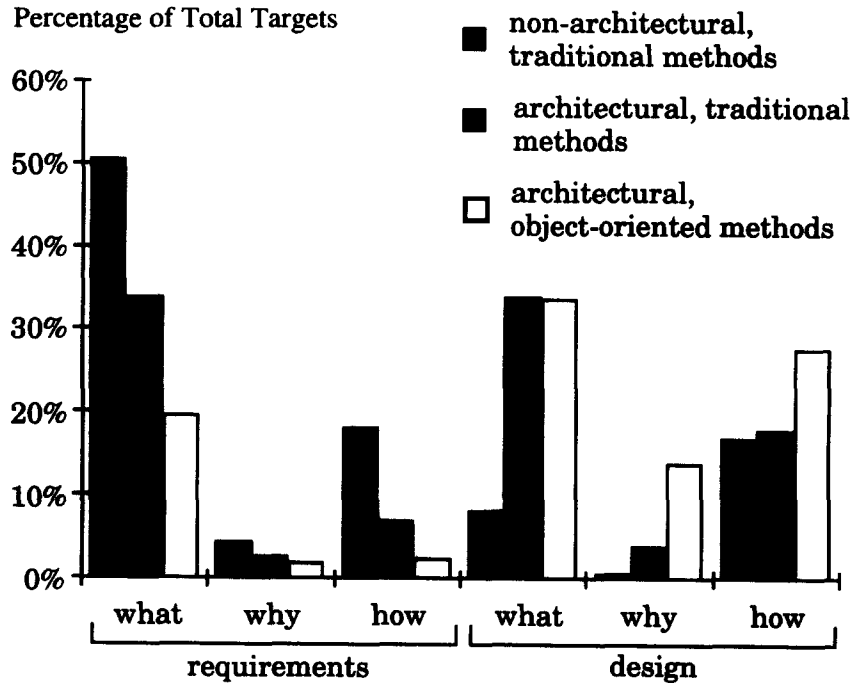


ous data, coding, maintenance, and testing are the source of very few questions during upstream development activities.

In order to explore further the most frequent kinds of targets, we constructed a cross-tabulation of the three most frequent attributes (what, how, why) and the two most frequent stages (requirements, design) and compared data from the OOD meetings to the combined data from meetings in which traditional methods were used (minutes and videotapes from projects A, B, C, and E). As Figure 12 shows, there is a smaller percentage of targets in each of the requirements categories in the OOD condition and a larger percentage in each of the design categories. Particularly noteworthy is the fact that the OOD condition has fewer *requirement-why* questions but many times more *design-why* questions. So, the approximate 3:1 ratio of *whys* in OOD development to *whys* in traditional development comes entirely from *whys* asked about design-stage targets.

In order to begin to identify the extent to which the task (developing an architecture vs. an application) and the method (traditional vs. OO) contributed to the differences in types of questions, we constructed three data sets as follows. Recall that the questions from traditional

Figure 13. **Cross-tabulation of most frequent attribute–stage combinations in targets taken from three kinds of meetings.**



meetings were drawn from several projects that were developing applications (projects B through E) and one project that was developing an architecture (project A). We culled the questions from project A to produce an application-traditional data set. We then selected another meeting from project A, extracted and analyzed all the questions from it, and combined them with the questions we already had from the other project A meeting. This gave us an architectural task, traditional methods data set consisting of 232 questions. The questions from the OOD meetings formed the architectural task, OO method data set.

The results of comparing across these three data sets are shown in Figure 13. The differences in frequencies of stage, $\chi^2(8) = 240.58$, $p < .0001$, and attribute, $\chi^2(8) = 54.07$, $p < .0001$, are statistically reliable. As Figure 13 shows, in almost every case the target frequencies for architectural design with traditional methods lie between nonarchitectural traditional and OO architectural. In general, these results suggest that some of the earlier differences we saw are due to the task but that the methods are also generating some differences.

In sum, these results indicate that the kind of information sought by designers using OOD methods differs substantially from that sought by

designers using traditional methods. The major shifts are a dramatic increase in *why* questions and a major shift toward questions about design and away from requirements.

The increase in *why* questions may indicate that the designers are reasoning more deeply about the design–seeking a more thorough understanding of the underlying issues. Our earlier result of finding very few *why* questions can be interpreted in several ways, as we pointed out in Herbsleb and Kuwana (1993). If we view the failure to ask about the reasoning behind design decisions as a shortcoming in design practice, that is as a problem to be overcome, the indications from our findings reported here are that use of OOD methods may be a substantial step in a desirable direction.

The shift away from asking about requirements and toward asking about design makes sense for architecture development. Because the goal is to support a range of imprecisely specified future applications, detailed questions about the requirements of any particular application are likely to be unanswerable. So, rather than focusing on what some application is supposed to do, the focus is on why the architecture should be designed a given way. Questions about functionality are still asked but in a more general form–as justifications for design decisions.

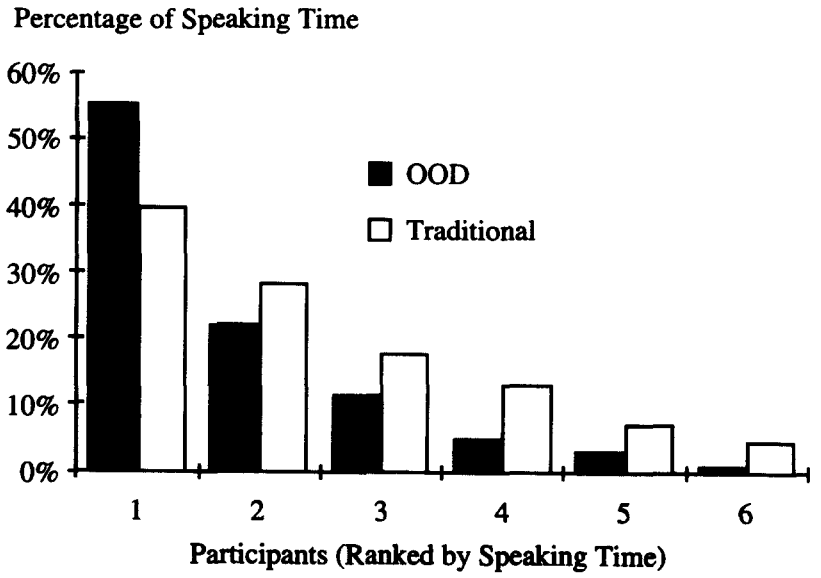
3.5. Organizational Matters

Role of Chief Architect

Many OOD methods suggest that a person or persons be specifically assigned the role of structuring the system and maintaining its integrity throughout development (e.g., Booch, 1991; Jacobson et al., 1992) In the field project we observed, this role was filled by a single chief architect. The centrality of this role is clear from the pattern of participation in development meetings we observed. There is a very consistent and robust finding in the small-group interaction literature about typical participation patterns (e.g., McGrath, 1984). As a rough generalization, the most frequent initiator of conversations will initiate 40% to 45% of all communications, the next initiator around 23%, next 17%, and so on (McGrath, 1984, p. 146). We performed a similar analysis in order to compare the patterns of participation in traditional and OOD development meetings.

For each of the 6 U S WEST OOD meetings we analyzed in detail and for each of the 10 meetings in which traditional methods were used, we added up the amount of time spent talking by each participant. We then plotted these times as percentages of total speaking time and ranked participants as the most talkative member, the next most talkative member, and so on. Figure 14 shows the comparison of the mean ranked percentage of speaking times in the OOD and the traditional meetings. Obviously, the distribution is much more uneven in the OOD meetings.

Figure 14. **Mean ranked percentage of speaking times in OOD and traditional meetings.**

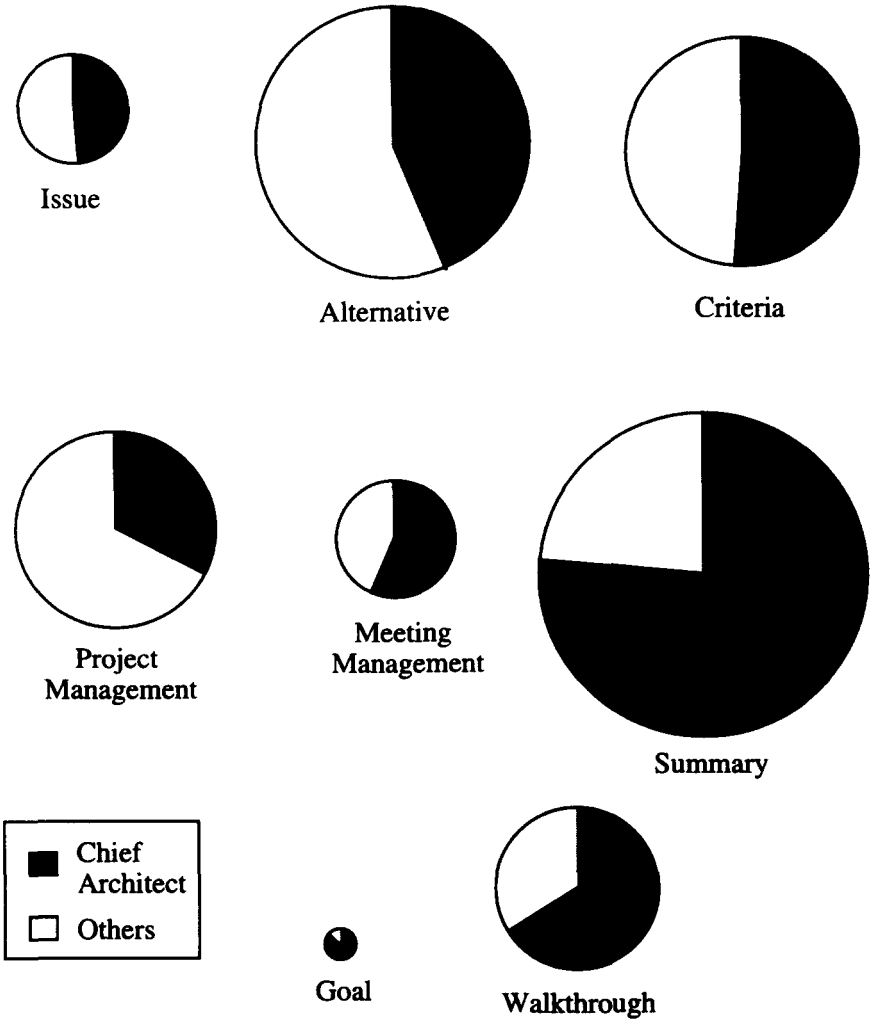


For 5 of the 6 OOD meetings, the chief architect was present; in each case, it was he who was most talkative.

In order to investigate this participation pattern further, we looked at how the most talkative person's speaking time was spent in the OOD meetings. So, we constructed a time-use chart like the ones presented earlier but consisting of only the eight most time-consuming activities. For the circle representing each of these activities, we shaded in the portion that represented the chief architect's contribution (see Figure 15). As the figure makes clear, the real dominance of the chief architect was in summary and walk-through and was not so apparent in what we had generally considered the core design activities of issue, alternative, and criteria.

There were some changes in the activity patterns suggesting that the group design work is accomplished somewhat differently by OOD teams. In particular, the chief architect developed much of the design off-line and summarized it for the rest of the team. As a result, the chief architect tended to take up a very large proportion of time at design meetings. But, the results do not suggest that the other team members had no input or felt too intimidated to offer ideas. They were very active in raising issues and suggesting alternatives. So, the image of the chief architect as dominating the process seems less accurate than the view of chief architect as one who brings previous experience and off-line problem solving to the meetings.

Figure 15. **Comparison of time spent in activities by chief architect and all others.**



More data relevant to the chief architect role are found in the concerns raised by developers in the weekly interviews. Most of the concerns, we suspect, are common to many development efforts, independent of method. The chief architect, however, is one of the two most frequently mentioned concerns (the other was resources). The real problem was that the chief architect was perceived to be a significant bottleneck in getting work done. He met frequently with other members of the team to get them

up to speed on the current state of the design and to discuss the architecture-level design decisions that guided their efforts. The chief architect also engaged in periodic "code sweeps" in which he would review the work done by himself and others and clean it up so it would be more elegant and efficient. Other members complained that these changes often caused their code to "break." Another problem seen as significant was "repeat coding," or recoding after a code sweep–the fifth most frequently mentioned concern.

These analyses of the role of chief architect help shed some light on other differences we observed between OOD and traditional meetings. In particular, the chief architect appears to be the person responsible for the large amount of meeting time spent in summaries and walk-throughs in our sample of OOD meetings. The nature of the role, the experience of the person who occupied it, and the challenge of maintaining architectural integrity may all have contributed to this pattern of interaction.

Interactions With Clients

The interactions of the design team with internal clients were complicated because of the sometimes conflicting expectations of different players. And, in fact, as is so often the case (e.g., Curtis et al., 1988; Walz et al., 1993), there was not a single "client" with a single voice and consistent set of expectations. There were constituencies that were primarily interested in a small research demo and others who wanted a particular application to go to market. The architectural group itself wanted to create a real product to be used in marketable applications. But, even factoring out these conflicting goals and expectations, it is possible to discern a few things about the quality of communication in the project we observed and the role of OOD representations in that communication.

In contrast to the picture of generally effective support for communication at the team level, our informal observations indicate that communication by the team with those outside the architecture project was much less effective. The essential problem seemed to be the "cognitive distance" (Krueger, 1992) between the very abstract classes provided in the architecture (e.g., "viewspaces" and "dataspaces") and clients' more concrete needs cast in terms of specific applications. Clients seemed to be unclear as to whether the architecture was going to provide what they needed.

As an example of this difficulty, the design team at one point resorted to metaphors to attempt to communicate what the architecture would provide. Data transported over the network were likened to a train, and a contrast was drawn between providing "flatcars" and providing "hitches." If one provided flatcars, application designers would need to place only cargo (data) on them, and the flatcars would deliver the data. On the other hand, providing hitches would serve only the function of stringing to-

gether "railroad cars" of the clients' own construction. One member of the team expressed the opinion that the clients were expecting flatcars but the team was designing hitches.

This suggests a tension between two purported benefits of OOD in support of distributed cognition. As mentioned earlier, in order to enable effective use of inheritance and reuse, one must design classes that are highly abstract. Insofar as possible, one wants to push attributes and behavior up the inheritance hierarchy so that they can be inherited rather than redesigned and recoded for each slightly different object in an application or across similar applications. On the other hand, the more abstract the classes become, the farther they are from the understanding of users and domain experts–which presumably is grounded in basic-level categories. This tension between generality and understandability has been noted before. Potts (1993), for example, put the matter quite bluntly, "Our experience is that one must choose between ease of change and ease of understanding, and that one can't have both" (p. 226).

This fairly negative view of OOD and client interaction contrasts sharply with what we heard from interviewees in seven of our nine field interviews. These interviewees reported that the flow of information between developers and users was enhanced by OOD methods. In the two interviews in which this enhancement was not reported, communication with clients was viewed as very good, but OOD was not seen as having much effect one way or the other. Those reporting the enhancement identified several ways in which it took place. By explicitly focusing on design as refinement of a domain model, developers were more motivated to acquire the vocabulary and concepts of the application domain–and the OOD primitives of objects, message passing, and so on seemed easy for users to grasp so that they could understand and contribute to design discussions.

Our interviewees also reported that the highly iterative nature of OOD contributed to effective client communication. In eight of the nine interviews, developers stressed that the iterative nature of the OOD development process was a valuable means of refining the problem domain. To paraphrase one manager, embedded within each of the major steps of analysis, design, and coding were smaller steps of analysis, design, and coding taking place iteratively. This allowed the developers to work with users to continually refine what they knew, and the involvement of users in each new iteration enabled them to see what they were getting and to identify what they really wanted.

This contrast between the troubled developer–client communication in the observed project and the generally favorable reports from our field interviews has led us to think about ways in which these projects differ. Although they obviously differ in many ways, one difference that seems particularly crucial is that, for the projects in which our field

interviewees had gained their experience, the application domain was generally well understood (e.g., billing systems, financial systems). This understanding was reflected in the existence of appropriate abstractions that had already been tested in use. On the other hand, the observed project was designing an architecture for a domain (a wide range of multimedia applications) for which a well-understood and tested set of abstractions did not yet exist. When such abstractions do not exist at the outset, there is, obviously, no vocabulary and set of concepts to be adopted by the developers that will automatically form a common ground for communication with users.

4. IMPLICATIONS

4.1. Tools

Our focus has been on the team aspects of development, so, in order to maintain this focus, we neglect many other important but essentially individual concerns that influence OOD tool design. The time-sheet analysis provides a first approximation of where in the OOD development process the ability to interact with shared representations is likely to be most important. The most relevant findings, of course, are that group work was most prevalent during upstream activities (i.e., requirements and design) as opposed to the more downstream activities of coding and testing. Design, in particular, seemed to require a very high ratio of group work to individual work. Group work was also more prevalent in the earlier periods for each type of activity. The trend was more pronounced in requirements and testing but is also visible in design and coding.

In addition, one of the major concerns encountered in this project was the difficulty developers had learning about the software design and maintaining consistency with changes the chief architect made in the code. These tasks were major components of the problems experienced in this project–problems often perceived as centering on the role of chief architect.

These findings have some fairly straightforward implications for OOD software development tools. Most such tools are designed for individual use. Yet, as research on distributed cognition shows (e.g., Hutchins, 1990), open interactions and open tools are often crucial to successfully accomplishing essential coordination functions. Tools often serve as a medium of communication as well as a means of learning about the domain and the task. This need was clearly perceived by one of the more experienced developers on the project we observed. He noted that one of the most important functions of the class hierarchy should be as a communication medium–a collaboratively developed knowledge base that represents a consensus on the bedrock concepts and vocabulary. The lack of tools to

support annotating, keeping abreast of changes, bringing novices up to speed–and collaborative use in general–is a serious drawback.

Facilitating the sharing of information about the design, in particular, is an extremely important function that should be supported by tools. We have found in laboratory studies of design (J. S. Olson et al., 1993) that a simple shared text editor used to support synchronous collaboration significantly improves the quality of the design outcome. The results reported here suggest that design is a highly collaborative activity, and yet there were no tools at all, beyond the traditional paper and whiteboard, available to the project team to facilitate collaboration by recording the design, changes to the design, decisions that had been made, rationales for decisions, open issues, and so on. The current state of research suggests there are very significant opportunities here for improving the design process with well-conceived tools.

Our findings are consonant with research on the major causes of software errors. For example, the two most frequent causes of errors in a sample of industrial applications were errors in module interfaces and misunderstanding of module functions (Nakajo & Kume, 1991). This points out the very high cost of failing to reach a consensus on design decisions and of failing to record them in a form that can be effectively shared.

4.2. Grain Size of Design Units

The difficulty of maintaining the consistency and integrity of the design may also suggest that encapsulated units larger than objects and classes are needed in order to coordinate development effectively. Individual classes and objects in large systems represent a fairly fine grain view. Larger entities, such as frameworks (Johnson & Russo, 1991), patterns (Coad, 1992), and subsystems (Wirfs-Brock et al., 1990), which could provide another, courser grain level of encapsulation, represent another possible approach to this problem. Rather than having a single person (e.g., a chief architect) manage the definitions and interfaces of a large collection of fine-grain objects and classes, system integrity issues could be addressed at the macro level.

4.3. Interactions With Clients

We had expected to observe how the development team members acquired specific knowledge about their application domain and how this knowledge was disseminated among them, but the conceptual gap between the specific needs of an application and the abstract resources to be provided by the architecture remained throughout the project. There was not much evidence that specific, detailed application domain knowledge was disseminated to any great extent among the design team members.

About 4 months into the project, the design team produced a requirements document, part of which was prepared by the main client. The team examined the client's part but primarily to make sure that the technology being designed was sufficiently general to support what the client wanted. The team gave little attention to how the architecture would support these specific needs. The same client generated an OO analysis model, but the team viewed it as not very helpful. In fact, the chief architect gave it only a very cursory examination, and those who looked at it more closely agreed that it offered little useful guidance for their purposes.

One of the observations that struck us most forcefully about this project was just how different the knowledge acquisition task facing developers was both from the traditional-methods projects we had observed and from the sort of OO domain modeling one sees in textbooks. Very little attention was paid to how any particular application would behave, and much effort was spent solving general problems that would have to be addressed for almost any distributed multimedia application. This, of course, is at least partly the result of the project priorities, as discussed earlier.

But, in addition, the struggle to understand the domain in this project was a radically different task from working in an established domain. The difference is that the requisite domain knowledge did not yet exist. Capturing existing knowledge can itself be a very difficult task, of course. But, in many well-understood domains, useful abstractions exist and have been developed and tested over a substantial period of time. For example, the field of accounting has evolved a set of abstractions that underlie financial software. They have been tested in use many times in a wide variety of applications and are by now well understood and known to be very general.

But where the goal, as here, is to span several domains, a set of tested abstractions known to have correct properties does not exist. So, the design is not primarily an exercise in capturing domain knowledge but rather the extraordinarily difficult task of discovering the deep conceptual structure of this new, broadly defined domain. This task is in many ways more akin to scientific discovery than to requirements analysis.

We think that this distinction between capturing and inventing domain knowledge has important implications for OOD development. One might wish to think of potential projects as arrayed along a continuum from pure knowledge capture to pure knowledge invention. The place occupied by a potential project on this continuum is no doubt determined by a complex set of factors, including the current state of knowledge in the project's application domain, practical concerns such as the availability and communication bandwidth with domain experts, and the extent to which project goals include creation of reusable designs and code.

We speculate that adopting an OOD approach will pay off with relatively little risk for projects toward the knowledge capture end of the continuum. Indeed, this is suggested by the favorable reports of OOD experience in our field interviews. This speculation is also based on the idea that the arguments favoring OOD and its support for distributed cognition are strongest when the task is essentially knowledge capture. By definition, domain knowledge exists in these projects and is disseminated throughout the design team, and a vocabulary and a set of concepts capable of supporting distributed cognition are also present. Toward the knowledge invention end of the scale, on the other hand, the risks will be high regardless of which method is used. Whether the arguments advanced in favor of OOD approaches have any basis at all in projects of this sort is unclear.

4.4. How to Organize for OO

There were significant problems with team coordination, and they centered on the role of chief architect. In the designers' view, as expressed in the weekly interviews, the chief architect was a major bottleneck, and this was a very significant issue. This points out the importance of exploring new models of cooperation and different types of class ownership as ways of coordinating in the face of evolving class hierarchies in OOD development (Nascimento & Dollimore, 1993). In particular, the role of chief architect–its boundaries and responsibilities–needs to be examined carefully. In this project, the chief architect was, in effect, a tutor on OOD technology for less experienced team members, a tutor on the previous version of the software on which part of the current design was based, the person with the authority to decide about the functions and interfaces for all classes, and an implementer who wrote a considerable portion of the code and modified much of the rest of it. This set of responsibilities is likely too much for one person, and ways must be found to allocate some portion of them to other team members.

5. CONCLUSIONS

In the OOD project we observed, design is the phase of software development in which the members of the development team met most often as a group. This reflects the need to bring multiple minds together to work synchronously on the hard problems of doing design. Supporting communication, coordination, and knowledge dissemination is a particularly important property of methods and tools aimed at the design phase.

Adopting OO methods for design appears to have significant effects on how teams work. Communication among members of the team is evidently quite effective, in that many fewer episodes of clarification occur than is the case among members of comparable teams using traditional

methods. Design discussions with OOD are also differently organized. Summaries and walk-throughs occur much more often and are more integrated into design activities.

The design team using OOD methods focused more attention on the *whys* of design. The analysis of comparable data on traditional design suggests that the shift in focus was at least partly due to the task (i.e., building an architecture as opposed to an application) but is probably also due to the adoption of OOD methods. To the extent that knowledge dissemination is driven by asking and answering questions, OOD seems to encourage a deeper inquiry into the reasoning underlying design decisions but less inquiry into the requirements.

The adoption of OOD methods was accompanied in our study by considerable uncertainty about how to organize the team. The use of a chief architect, a role advocated by several methods, seems fraught with risks that are likely to go up as project size increases. All of our field interviewees were much concerned about how to organize their teams, and none expressed great confidence in any particular management scheme. It is going to take careful analysis of the interplay of cognitive and organizational factors across a range of studies to determine how best to organize OOD teams. Especially interesting is how new tools to support information sharing in design might interact with organizational matters.

NOTES

Acknowledgments. We thank Lori Meiskey and Tim Miller for their help in gathering data for this work. We also thank all the development team members, who dutifully participated in interviews, reported how they spent their time, and cheerfully tolerated the constant videotaping and audiotaping. Last, we thank Michael Cohen, Charles Hymes, Robin Lampert, Max Rahder, and Lynn Streeter for their many insightful comments on drafts of this article.

Support. This work was supported by a grant from U S WEST Technologies, by National Science Foundation Grant IRI-8902930, and by the Center for Strategic Technology Research (CSTaR) at Andersen Consulting.

Authors' Present Addresses. James D. Herbsleb, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: jherbsle@sei.cmu.edu; Helen Klein, Gary M. Olson, and Judith S. Olson, Collaboratory for Research on Electronic Work, University of Michigan, 701 Tappan Street, Ann Arbor, MI 48109-1234. E-mail: helen.klein@ um.cc.umich.edu, gmo@crew.umich.edu, and jsolson@crew.umich.edu; Hans Brunner, U S WEST Technologies, 4001 Discovery Boulevard, Boulder, CO 80303. E-mail: hansi@advtech.uswest.com; Joe Harding, Harding Consulting, P.O. Box 4615, Boulder, CO 80306.

HCI Editorial Record. First manuscript received March 15, 1993. Revision received December 29, 1993. Accepted by Robert Rist. Final manuscript received October 19, 1994. – *Editor*

REFERENCES

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Booch, G. (1991). *Object oriented design with applications.* Redwood City, CA: Benjamin/Cummings.

Brooks, F. P. (1975). *The mythical man-month: Essays on software engineering.* Reading, MA: Addison-Wesley.

Brooks, F. P. (1987). No silver bullet. *IEEE Computer, 20,* 10–19.

Bruegge, B., Blythe, J., Jackson, J., & Shufelt, J. (1992). Object-oriented system modeling with OMT. *Proceedings of OOPSLA '92,* 359–376. Vancouver: ACM.

Champeaux, D. D., Anderson, A., & Feldhousen, E. (1992). Case study of object-oriented software development. *Proceedings of OOPSLA '92,* 377–391. Vancouver: ACM.

Chi, M. T. H., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science, 5,* 121–152.

Coad, P. (1992). Object-oriented patterns. *Communications of the ACM, 35,* 152–159.

Coad, P., & Yourdon, E. (1991). *Object-oriented analysis* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Constantine, L. L. (1993). Work organization: Paradigms for project management and organization. *Communications of the ACM, 36,* 35–43.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31,* 1268–1287.

Curtis, B., & Walz, D. (1990). The psychology of programming in the large: Team and organizational behavior. In J.-M. Hoc, T. R. G. Green, R. Samurçay, & D. J. Gilmore (Eds.), *Psychology of programming* (pp. 253–270). London: Academic.

Fenton, N. (1993). How effective are software engineering methods? *Journal of Systems and Software, 22,* 141–146.

Fienberg, S. E. (1980). *The analysis of cross-classified categorical data* (2d ed.). Cambridge, MA: MIT Press.

Galbraith, J. (1973). *Designing complex organizations.* Reading, MA: Addison-Wesley.

Gentner, D. (1981). Some interesting differences between verbs and nouns. *Cognition and Brain Theory, 4,* 161–178.

Gentner, D., & France, I. M. (1988). The verb mutability effect: Studies of the combinatorial semantics of nouns and verbs. In S. I. Small, G. W. Cottrell, & M. K. Tanenhaus (Eds.), *Lexical ambiguity resolution: Perspectives from psycholinguistics, neuropsychology and artificial intelligence* (pp. 343–382). San Mateo, CA: Morgan Kaufman.

Gottman, J. M., & Roy, A. K. (1990). *Sequential analysis: A guide for behavioral researchers.* New York: Cambridge University Press.

Henderson-Sellers, B., & Edwards, J. M. (1990). The object-oriented systems life cycle. *Communications of the ACM, 33,* 142–159.

Herbsleb, J. D., & Kuwana, E. (1993). Preserving knowledge in software engineering: What designers need to know. *Proceedings of INTERCHI '93,* 7–14. New York: ACM.

Hutchins, E. (1990). The technology of team navigation. In J. Galegher, R. E. Kraut, & C. Egido (Eds.), *Intellectual teamwork* (pp. 191–220). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Hutchins, E. (in press). How a cockpit remembers its speed. *Cognitive Science.*

Jacobson, I., Christerson, M., Johnsson, P., & Overgaard, G. (1992). *Object-oriented software engineering: A use case driven approach.* Reading, MA: Addison-Wesley.

Jeffries, R., Turner, A. A., Polson, P. G., & Atwood, M. (1981). The processes involved in designing software. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 255–283). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Johnson, R. E., & Russo, V. F. (1991). *Reusing object-oriented designs* (Technical Report UIUCDCS 91-1696). Champaign: University of Illinois.

Karat, J., & Bennett, J. L. (1991). Working within the design process: Supporting effective and efficient design. In J. M. Carroll (Ed.), *Designing interaction: Psychology at the human–computer interface* (pp. 269–285). New York: Cambridge University Press.

Krasner, H., Curtis, B., & Iscoe, N. (1987). Communication breakdowns and boundary spanning activities on large programming projects. In G. M. Olson, S. Sheppard, & E. Soloway (Eds.), *Empirical studies of programmers: Second workshop* (pp. 47–64). Norwood, NJ: Ablex.

Kraut, R. E., & Streeter, L. A. (in press). Coordination in large scale software development. *Communications of the ACM.*

Krueger, C. W. (1992). Software reuse. *ACM Computing Surveys, 24,* 131–183.

Kuwana, E., & Herbsleb, J. D. (1993). Representing knowledge in requirements engineering: An empirical study of what software engineers need to know. *Proceedings of the IEEE International Symposium on Requirements Engineering,* 273–276. San Diego: IEEE Computer Society Press.

Lubars, M., Potts, C., & Richter, C. (1993). A review of the state of the practice in requirements modeling. *Proceedings of the IEEE International Symposium on Requirements Engineering,* 2–14. San Diego: IEEE Computer Society Press.

Malhotra, A., Thomas, J. C., Carroll, J. M., & Miler, L. A. (1980). Cognitive processes in design. *International Journal of Man-Machine Studies, 12,* 119–140.

Martin, J., & Odell, J. J. (1992). *Object-oriented analysis and design.* Englewood Cliffs, NJ: Prentice-Hall.

McGrath J. E. (1984). *Groups: Interaction and performance.* Englewood Cliffs, NJ: Prentice-Hall.

McKeithen, K. B., Reitman, J. S., Rueter, H. H., & Hirtle, S. C. (1981). Knowledge organization and skill differences in computer programmers. *Cognitive Psychology, 13,* 307–325.

Meyer, B. (1992). The new culture of software development: Reflections on the practice of object-oriented design. In D. Mandrioli & B. Meyer (Eds.), *Advances in object-oriented software engineering* (pp. 51–64) Englewood Cliffs, NJ: Prentice-Hall.

Miller, G. A. (1991). *The science of words.* New York: Scientific American Library.

Nakajo, T., & Kume, H. (1991). A case history analysis of software error cause–effect relationships. *IEEE Transactions on Software Engineering, 17,* 830–837.

Nascimento, C., & Dollimore, J. (1993). A model for co-operative object-oriented programming. *Software Engineering Journal, 8,* 41–48.

Nierstrasz, O., Gibbs, S., & Tsichritzis, D. (1992). Component-oriented software development. *Communications of the ACM, 35,* 160–165.

Olson, G. M., Herbsleb, J. D., & Rueter, H. H. (1994). Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Human–Computer Interaction, 9,* 427–472.

Olson, G. M., Olson, J. S., Carter, M. R., & Storrøsten, M. (1992). Small group design meetings: An analysis of collaboration. *Human–Computer Interaction, 7,* 347–374.

Olson, G. M., Olson, J. S., Storrøsten, M., Carter, M., Herbsleb, J., & Rueter, H. (in press). The structure of activity during design meetings. In T. P. Moran & J. M. Carroll (Eds.), *Design rationale.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Olson, J. S., Olson, G. M., Storrøsten, M., & Carter, M. (1993). Groupwork close up: A comparison of the group design process with and without a simple group editor. *ACM Transactions on Information Systems, 11,* 321–348.

Potts, C. (1993). Panel: "I never knew my requirements were object-oriented until I talked to my analyst." *Proceedings of the IEEE International Symposium on Requirements Engineering,* 226. San Diego: IEEE Computer Society Press.

Rosch, E. (1978). Principles of categorization. In E. Rosch & B. Lloyd (Eds.), *Cognition and categorization* (pp. 28–49). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Rosch, E., Mervis, C. B., Gray, W., Johnson, D., & Boyes-Braem, P. (1975). Basic objects in natural categories. *Cognitive Psychology, 8,* 133–156.

Rosson, M. B., & Alpert, S. R. (1990). The cognitive consequences of object-oriented design. *Human–Computer Interaction, 5,* 345–379.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-oriented modeling and design.* Englewood Cliffs, NJ: Prentice-Hall.

Walz, D. B., Elam, J. J., & Curtis, B. (1993). Inside a software design team: Knowledge acquisition, sharing, and integration. *Communications of the ACM, 36,* 62–77.

Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1990). *Designing object-oriented software.* Englewood Cliffs, NJ: Prentice-Hall.