INSIDE A SOFTWARE DESIGN TEAM: KNOWLEDGE ACQUISITION, SHARING, AND INTEGRATION

Diane B. Walz, Joyce J. Elam, and Bill Curtis

More than half the cost of the development of complex computer-based information systems (IS) is attributable to decisions made in the *upstream* portion of the software development process; namely, requirements specification and design [20]. There is growing recognition that research on how teams actually go about making requirement determinations and design decisions can provide valuable insights for improving the quality and productivity of large-scale computer-based IS development efforts [9, 12, 23]. Traditional models of group dynamics, group decision making, and group development are not rich enough to thoroughly explain the real-world complexities faced by software design teams. Most of this research was performed on tasks that were shorter, less complex and did not require the extensive integration of knowledge domains that characterizes software systems design [9, 12].

Knowledge is the raw material of software design teams. For complex projects, knowledge from multiple technical and functional domains is a necessity [12]. Ideally, a software design team is staffed so that both the levels and the distribution of knowledge within the team match those required for the successful completion of the project. Because of knowledge shortfalls such as the thin spread of application domain knowledge in most organizations, however, this is seldom the case [12]. In general, individual team members do not have all of the knowledge required for the project and must acquire additional information before accomplishing productive work. The sources of this information can be relevant documentation, formal training sessions, the results of trial-and-error behavior, and other team members. Group meetings are an important environment for learning, since they allow team members to share information and learn about other domains relevant to their work.

Productive design activities need to revolve around the integration of the various knowledge domains. This integration leads to shared models of the problem under consideration and potential solutions. A software design team seldom starts its life with shared models of the system to be built. Instead, these models develop over time as team members learn from one another about the expected behavior of the application and the computational structures required to produce this behavior. This means that team members need to be *speaking the same language* (or, at least, dialects whose semantics are similar enough to facilitate communication and understanding) in order to share knowledge about the system.

Knowledge acquisition, knowledge sharing, and knowledge integration are significant, time-consuming activities that precede the development of a design document. The purpose of this article is to examine how these activities unfolded over time inside an actual software design team. Two related questions with respect to this team will be resolved:

1) *How do the group members acquire, share, and integrate project-relevant knowledge?*
2) *Do the levels of participation in these activities differ across team members?*

The findings reported here challenge some of the conventional wisdom and common practices of managing software design teams. An initial caveat is that the design team studied here worked in a research and development environment where knowledge acquisition, sharing, and integration activities are accentuated. However, to varying degrees, these activities characterize most software projects [12]. A better understanding of the role and process of knowledge acquisition, sharing, and integration in software design has very real implications for managing large software projects, particularly in the areas of planning, staffing, and training.

The Software Project: An Overview

A software project involving the development of a system to manage persistent data within an object-oriented framework, an object server, was undertaken in 1986 at Microelectronics and Computer Technology Corporation (MCC). A single team of individuals worked on the project. The team was formed specifically for the project, and, in general, the team members had not previously worked with one another. All participants were either experienced software designers, researchers, or both. No specific design techniques or disciplined development methodology was forced upon the project team. Team meetings during the design phase of the project (August

through November) were videotaped by researchers as part of MCC's Software Technology Program (STP). Team members were aware they were being videotaped. They reported that the taping was not intrusive and did not affect their behavior.

The design team met 19 times from August through November. In two of these meetings, technical information was formally presented to the team in seminar fashion by an outside expert. The remaining 17 meetings were more traditional team meetings. In early November, a formal project plan, including specific tasks and relevant stop and start dates, was developed by the project manager. Once this project plan was in place, and the project shifted from design to coding, videotaping of the team meetings ceased. By February, the project team had produced a prototype of the object server and two language interfaces, along with relevant documentation (functional specifications and users' guides). No formal measure of the quality of the team's output was available. The actual system that was developed was an exploratory prototype, and although it was executable, it was not installed for commercial use. The customers stated they were, in general, pleased with the project's outcome in supplying the organization with a valuable prototype and considerable learning in a specialized technical area.

A time line, shown in Figure 1, describes project staffing and some major events over the four months during which the 19 team meetings were videotaped. The team members are identified as eight designers (D1–D8), a project manager (PM), and one representative from the customer group (C1). A brief summary of each of the 19 meetings is given in the remainder of this section.

Meeting #1 of the design team was held on August 6. The team was given a one-page specification document that described the object server from the customers' perspective. A deadline of January 15 was given for delivery of the object server. The discussion during this meeting focused on areas that different members found to be unclear in the specification document they were given. The designers with the least experience wanted to request that the customers produce a specification that was more clear and precise. The experienced designers agreed the specification was fuzzy, but stated that this was fairly standard.

- *"I haven't seen any good ones and they always come up with exactly the same thing. This is just characteristic of them."*

Experienced designers recognized that the customers may not, themselves, understand the true nature of the requirements at the beginning of a project.

- *"The big problem is they don't know what they want. Articulating it is not always the problem, it's really knowing what you want in the first place."*

While this first meeting primarily focused on the specification document supplied by the customers, one of the team members (D1) presented his ideas about the general concept of an "information base." His presentation was very interactive and was interrupted every 2 or 3 sentences with questions from others. At the end of the meeting, team members agreed to write their questions for discussion in the next meeting.

Meeting #2 was held on August 8. During this meeting, the questions concerning requirements prepared in advance by team members were discussed. This discussion, however, was not limited to requirements. In fact, the group spent considerable time talking about various technical aspects of an object server (e.g., classes, objects, inheritance, messages, locking, concurrency). The discussion in this meeting was quite lively, with team members interrupting and disagreeing with one another as well as expanding on their own or others' comments. During this meeting, D4 stated he wanted to develop a prototype of the object server in Prolog for (personal) educational purposes. Team members were informed that the project deadline had been extended two weeks to February 1.

Meeting #3 was held August 12. The team invited two customers (C2 and C3) to attend the meeting to help clarify the requirements. D1 had a list of 23 questions he had assembled from the previous two team meetings. There were four general questions and 19 specific questions. During this meeting, the four general questions and four of the 19 specific questions were discussed. D1 recorded answers to the questions on the typed document which contained the questions, writing in the margins and between lines. One of the customers offered fairly elaborate "scenarios of use" to explain his views.

- *"Let me give you a sense of the kind of dynamics that we're talking about here . . ."*
- *"Let's put it this way, I stated it, and I think I probably stated it wrongly . . . What I would like is . . ."*

These complete scenarios were not recorded anywhere; only small fragments of them were noted next to the questions. The two customers had several disagreements about the overall approach to the task.

Meeting #4 was held the next day (August 13) with C2 and C3 to complete a discussion of the design team's questions. Again, C2 and C3 disagreed about many things, including the specific language in which to implement the object server. This disagreement can be clearly seen with respect to one of the questions being discussed:

Question: Are messages to be posed in the same language in which the server itself is written?

- *C3: "no"*
- *C2: "yes"*
- *D1 records "Y (for now)"*
- *D4: "And we can have another meeting without him (C3) where we can talk about the language issue again."*

Meeting #5 was held on August 19. D3 was present for the first time. The activities of the project (so far) were described for D3, including the disagreements between C2 and C3 on ideas about the project.

- *D1: "Basically, we sat down, we decided what it was that we wanted clarified about the spec, made up a list of questions, strapped C2 in a chair, beat him forcefully with a list, required that he at least verbalize something about what he was thinking . . . It became quite clear that the*

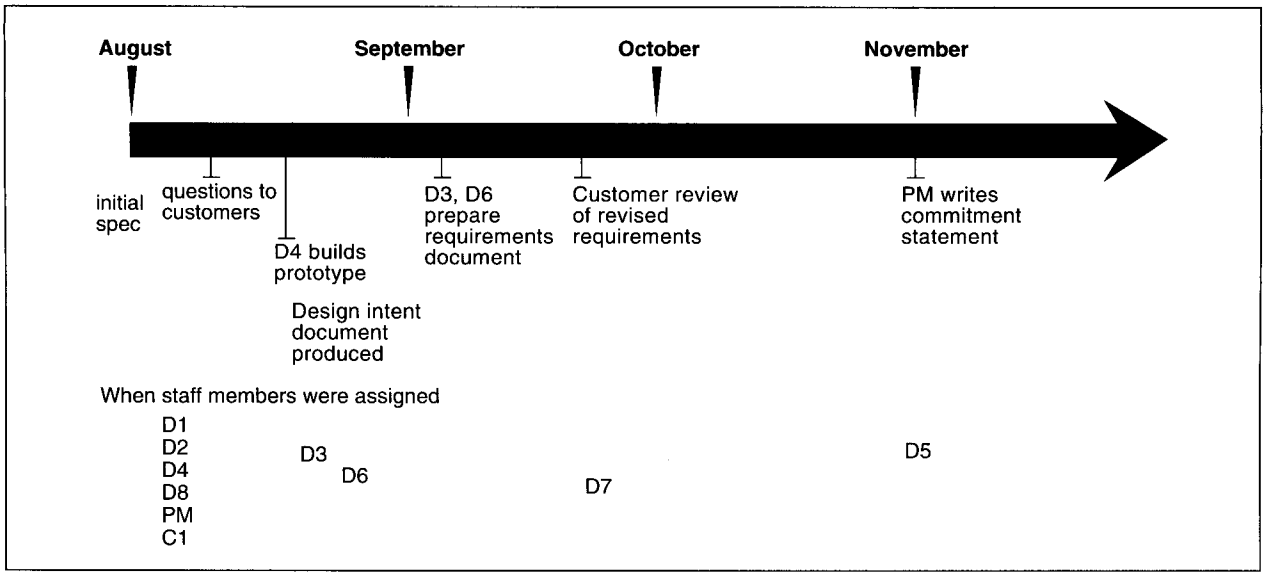


Figure 1. Time line

intersection of the ideas of C2 and C3 are very very small, at least as related to this project . . . But, it has ceased to be our problem—he's (C3) gone off . . .

- *D4: "Is it really no longer a problem? Is he no longer a customer officially?"*
- *D1: "Apparently"*
- *C1: "Well, given the wide diversions among customers, we probably should feel free to choose which customers we want, who speaks loudest to us . . ."*

During this meeting, the team members learned that D1 had prepared a preliminary design intent document and that D4 had developed a prototype of the object server in Prolog.

- *D1: ". . . I then . . . created a document expressing our preliminary design intent, in a very informal way. Sort of encapsulating the answers we've gotten from (C2), plus the discussions that I had with (D4). (D4) had been building a Prolog implementation of an object-oriented environment, partly for his own understanding, and partly to see how, if at all, it related to this."*

D1 stated that he had sent the design intent document to C2. He took C2's lack of "complaints" as an indication that he approved of the document. During the meeting, D4 spent considerable time explaining why Prolog was an appropriate environment for implementing the object server as well as describing the current state of his implementation of the object server. He requested that others read his documentation, examine the code, and work with the prototype in order to determine if an extension of the prototype could satisfy the specifications. Few of the designers were familiar with Prolog and were reluctant to commit to implementing the object server in Prolog. Thus, the issue of whether the object server should be written in Prolog or some other language remained unresolved. The meeting ended with a request for D4 to provide team members with a tutorial on Prolog.

During meeting #6 on August 22, D1 presented a plan for implementing the "master information base" in Ingres. The stated goal of the meeting was to "educate" everyone on Ingres so that the group could evaluate its potential as a tool for building the object server. A great deal of technical information on Ingres was shared. The team spent considerable time comparing how various requirements could be met in Ingres vs. Prolog. The team was beginning to consider different designs and how those designs could be implemented using both Prolog and Ingres.

- *D3: "An observation I'd like to offer at this point is that you've gone essentially from a set of requirements that aren't completely finished yet to a specific implementation without expressing things in a more generic sense . . . in the future, . . . we're going to have a bit of a problem understanding what the original intent was because we've gone directly to implementation."*

An outside expert was invited to come and talk to the group about Prolog on August 27 (Meeting #7). He described some current research on the subject in a fairly abstract manner. The group pushed strongly to bring the discussion to issues directly relevant to the system they were building. One of the issues discussed by the group concerned the paging of Prolog facts between primary and secondary storage.

- *Expert: Paging is done by the host operating system, it's not a Prolog-controlled paging.*
- *D4: You could have Prolog write its own facts out.*
- *Expert: Right, you could have communication between Prolog and some database system.*
- *D4: That's the way I imagined doing it. Do you think that's feasible?*
- *Expert: Yeah, sure. Of course, it's not going to be very efficient but that is a good first pass.*

Meeting #8 on September 3 was devoted to discussing a revised design intent document that had been prepared by two of the designers. Recall that D1 had written the initial design intent document. During the meeting, it became clear that the team members were not at all happy with this document. It seemed to the customer representative on the team that the current document had not

Experienced designers recognized that the **customers may not understand the true nature of the requirements** *at the beginning of a project.*

captured much of what the design team had agreed to earlier.

• *C1: "We had a bunch of discussions during last week and I thought we were starting to get agreement but now that I've read this I can't (see that agreement)."*

The team spent a great deal of time discussing exactly what should be prepared for the customers—should it be a requirements document, a design intent document, or a functional specification? This discussion led to another detailed discussion of requirements, which in turn led to a detailed discussion of design issues. Two important decisions emerged from the discussions: first, a decision was made to start from scratch in preparing the design intent document. Second, D1 and D4 were given the approval to continue to work on the Prolog prototype.

Meeting #9 was held on September 5. A new outline for the design intent document was distributed. As the team members went through the outline, requirements were discussed further. The team discussed the difficulties in getting feedback from customers. The extent to which the team had moved toward developing a design became very clear when the customer representative on the team indicated his intent to get feedback from a large number of customers on the design intent document being prepared by the design team.

• *C1: Incidentally, when you give this to the customers, I'm going to reexpand the customer group from the original group.*
• *D4: Isn't it a little bit late? I mean after all the preliminary design work that we have done . . . It worries me that it's going to start the whole thing all over again.*

During Meeting #10 on September 17, C1 informed the design team that the customer group had received the design intent document and was working on a response. He said the customers wanted a functional specification prepared in the form of three users manuals: one user manual for an application programmer using Lisp/Flavors to interface with the server, a second user manual for an application programmer using Prolog/Biggertalk to interface with the server, and a third user manual for someone who wanted to write the interface for another language such as Objective C. At this meeting, it was announced that the prototype being developed by D4 would be ready to show to customers as soon as D4 built an example case and D1 wrote a user manual. A good part of the meeting was taken up by D4 describing his prototype. At this meeting, two basic design approaches were articulated: Plan A and Plan W.

• *D6: Should we start on Plan A or Plan W? Because they're different design issues, very different ones.*
• *PM: I don't think that we have enough information right now to make a decision . . . we need to come up with the process cost for each plan and decide when we will reconvene and make a decision on it.*

Meeting #11 was held on September 26. Although the design team was still waiting for the next round of requirements from the customers, they proceeded to write the user manuals that C1 outlined in the previous meeting. There was much technical discussion on how to perform different tasks, such as handling conflicts surrounding objects. Some changes to the prototype were suggested.

During Meeting #12 on September 30, the design team discussed the requirements document received back from the customers. Team members were not happy—they were frustrated because the requirements still were not clear.

• *D1: D3 and I basically decided that rather than go through another iteration of make up the questions, give the questions to customer, wait a month-and-a-half for their response, we're simply going to list things and say these are now implementation-defined.*

Much of the meeting was devoted to trying to "read between the lines" to determine what the users really wanted and discussing how they could adapt the design approaches they had been pursuing to do this.

• *D3: What C2 wants is to be able for his reusability project to work in Flavors environment. He doesn't want to comingle languages; he just wants an information base server that has many objects that is accessible to his Flavors environment.*
• *PM: This (the document) does not say it.*
• *D3: It doesn't say it, no.*

Meeting #13 was a continuation of Meeting #12. During this meeting, the team went through the customer requirement document point by point attempting to clarify and reach a consensus of understanding. There were still unclear areas; there were apparent contradictions. Since the team did not get all the way through the document, they agreed to meet the next morning to continue.

During Meeting #14 on October 1, the design team continued their joint review of the customer's document. They went through the document, discussing items that were unclear, wrong, and so forth. They kept track of matters they could not resolve and which needed to be addressed by the customer group. Some of the discussion was very high-level (. . . "different language environments"); some of it was very detailed (". . . We have to explicitly follow the pointers in the application code.")

At the start of Meeting #15 on October 8, the PM distributed a draft of a "Commitment Statement for the

Information Base Server." D4 mentioned C2's "revised expectations" about sharing objects between languages. The team was not willing to commit to this new requirement. There was not a lot of controversy involved with PM's draft document. Only minor wording changes were suggested. Most of the discussion referred to fairly high-level issues concerning the requirements: multiple languages and the sharing of objects. The next steps for the project were discussed, including the need to develop a rigorously laid-out plan, with specific tasks, dates, and so forth.

Three members of the design team had been working on a part of the object server—an object base management system (OBMS). The intended format of Meeting #16 on November 5 was to let each of these members run through their presentations and save questions for the end. Only two of the designers were able to present. There was some detailed discussion of issues such as locking, notification, time stamps, serializing problems, and atomic operations.

Meeting #17 occurred later in the day on November 5. This meeting involved the discussion of a huge PERT chart that the PM had prepared. It was a plan for completing the project tasks. The PM wanted the team members to evaluate the plan, especially to see if the start and stop dates were reasonable. This was a short meeting, barely 20 minutes long. Much was still unclear and unresolved, as was evident in the longer meeting earlier in the day. However, with time running out, the project leader was obviously trying to get beyond any further discussion of requirements.

The purpose of Meeting #18 on November 11 was to discuss work that had been done concerning database issues. It was to be a continuation of the discussion of database issues from Meeting #16. At the start of the meeting, D4 requested that C2 be invited into the meeting:

- *D4: Can we get C2 here for this because something was raised during the meeting we just had with him that struck me as remarkable, and I think his new notions of what he wants should be expressed directly to the group. I just spoke to him and he sounded like he'd be willing to come and talk at this meeting about that certain topic.*

C2 commented on what he wanted:

- *C2: In the time frame between now and February 1, I want to focus my personal activities on just . . . reusability issues . . . I'm going to sit and ask questions about what kind of objects do I want to build in a Flavors environment. And I think realistically, I or anybody else, am not going to have a good idea of what requirements there are until you've gone through that, until you actually tried out in a fairly large-scale experiment with a set of objects trying to do some reusability.*
- *D4: If you tell us in a precise way what you want to do with this thing, we can build in the functionality right now.*
- *C2: That's indeed why I want to spend the next few months figuring out what that precise way is, that's what I'm saying.*

The design team discussed several alternatives for providing the functionality requested by C2. By the end of the meeting, five possible approaches were identified. After C2 left, the discussion turned more technical about "how to do it" including such issues as dangling pointers, locking objects, locking subtrees, deleting objects, garbage collection, and global backup and integrity of the object store.

Meeting #19 was held on November 21. A functional specification document that had been previously prepared by one designer was discussed. Much of the discussion was related to the goal of trying to assure the specification document was complete. The team discussed whether this project related to others at MCC, the system, and how it would work. Questions about the system were posed, relative to D4's system and the various documents describing the server and the OBMS. There was some discussion about how to proceed. D4 discussed starting work "on the languages": D3 suggested splitting the current functional specification by category (e.g., maintenance, storage, etc.) and assigning a category to each team member who would critically evaluate the document and participate in future discussions with that "bent." These assignments were made late in the meeting. After this meeting, the project shifted to implementing the specification.

Observations from the Videotapes

The observations presented in this section are based upon our analysis of the 19 design team meetings. We first reviewed the transcripts of the group meetings to qualitatively assess the nature and level of knowledge acquisition, sharing, and integration activities. Next, the transcripts were analyzed in a structured manner in order to obtain measurements which might support or deny our qualitative assessments. A description of this analytical approach is described in Appendix A; further details are provided in [24].

The design meetings were very professional in nature. Interactions were, for the most part, task-oriented with lively discussions. Participants were serious about their assignment and appeared to be trying hard to do a good job. In general, we identified three general topics of discussion: 1) background knowledge (technical and application knowledge, especially knowledge that was new to some or all team members), 2) system requirements, and 3) design approaches.

The traditional approach to software development recommends that these topics be addressed in sequence. Projects are supposed to be staffed to cover the different "knowledge domains." If necessary, early training is supplied. The design team begins its work by determining requirements, although during this time, designers may require "education" about the functional area. After requirements are determined, designers invent some reasonable design approaches to meet system requirements and evaluate these approaches, selecting one to be implemented.

In the design project we studied, we saw these three "steps." From the descriptions of the team meetings, it is clear that these steps were not addressed in sequence, they were not independent of one another, and they did not appear to have clear starting and ending points. Technical

knowledge was introduced, exchanged, and evaluated according to its ability to meet requirements in the context of one or more specific design approaches. New information about requirements was evaluated in the context of design approaches framed in terms of technical and application knowledge. Presentations about new technology were discussed in light of various design approaches and whether or not such approaches met requirements. Thus, new information was sought, filtered, and integrated in context.

Very early in the project (Meeting #5), the team began to focus on what they called a *design intent document*—a document for customers that said "this is what we understood you to mean and this is what we intend to do about it." The team also was introduced to the prototype being developed by D4, which provided a very concrete design for meeting requirements. Beginning in the middle of August, discussions related to technical knowledge, requirements, and design became closely intertwined. We did, however, see shifts over time in the team's focus with respect to these three topics (see Figure 2). In the early meetings, the team focused on learning what they needed for producing a design and identifying the requirements of the system. Discussions, however, generally related to assumed, or "trial" design approaches. This emphasis was evident through meeting #7, in late August. Around this time, the emphasis on new technical knowledge appeared to lessen and the focus of the team was one of getting a clear handle on requirements and relating these to specific design approaches. In fact, design approaches discussed in previous meetings appeared to have been solidified by the middle of September and were referred to by names (e.g., "Plan A") for the first time, starting in meeting #10.

After meeting #10, the team was still attempting to get requirements clarified. The discussion of requirements from this point, however, was rooted in the context of specific design alternatives ("Plan A vs. Plan W"). As can be seen in meeting #12, the team was close to reaching its limit on accepting additional requirements from customers. By meeting #16 (early November) the process of actively determining requirements was simply 'shut down,' even though requirements were still not entirely clear, either to the designers or the customer representative. The team's focus from this point is on the various design alternatives, discussed in the context of known requirements. It appeared that the shifting of the team's attention from requirements determination to design activities was precipitated by members' awareness of time and deadlines.

The phenomenon of 'shutting down' in other software design projects was observed by Gersick [14], who noted that it tended to occur near the halfway point between a project's starting date and its deadline. It is interesting that the shift observed here, in early November, corresponds to the midway point between project inception in early August and the February 1 delivery deadline.

We have classified our observations of this software design team along three dimensions: acquiring, sharing, and integrating the necessary knowledge for the design task (*getting up to speed*), integrating the knowledge into a shared understanding of the application and the design (*creating the team memory*), and the role of individuals in these activities (*the players*). The analysis of the transcripts from the project has yielded some interesting observations in these areas, suggesting that some of our traditional approaches to managing the software design process may need rethinking.

Getting Up to Speed

From Meeting #1 through meeting #16 in early November, the team members focused on obtaining both technical and requirements-oriented information. The junior designers were appalled at the amorphous nature of the initial requirements document and wanted to demand something with more specifics. The experienced designers recognized that customers "don't know what they want" and the fuzziness in their requirements document was common. Customers, like designers, needed to go through a learning process in order to clarify the requirements. Once this was recognized, it was not surprising that it took so long to gain closure on the requirements.

Determining requirements was also complicated because different customers had different requirements. The design team clearly wanted to avoid this complication by being responsible to a customer (or customer group) that shared the same view of the requirements. The customer they chose was C2 and much of their thinking was shaped by what C2 wanted. The design team was quite alarmed when the customer representative on the design team wanted to open the discussion of requirements to a larger customer audience once the project had become established.

On the technical side, the team actively sought information about the object-oriented paradigm and the relevant characteristics of various database environments. From the meeting discussions we can infer the kinds of activities in which they engaged outside of the project meetings: tracking down and reading documentation and research papers; consulting with experts (both internal and external to MCC), technical specialists, and vendors. Even during the first meeting, it is obvious that team members had "done homework" before assembling.

During the meetings, team members exchanged knowledge through discussions. Individuals often asked one another direct questions. Team members appeared eager to supply their own expertise where relevant. We observed numerous examples of knowledge exchange in a classic dialectic process, in which a statement of position was criticized as a catalyst for a discussion process whose outcome involved individuals accepting new knowledge or revising beliefs.

We observed a large amount of conflict in the meetings we studied. On average, about 16% of all statements were made in disagreement or challenge to another. While we did observe some cases of disagreements that appeared to be the result of incompatible goals, most of the conflict that occurred during the design team

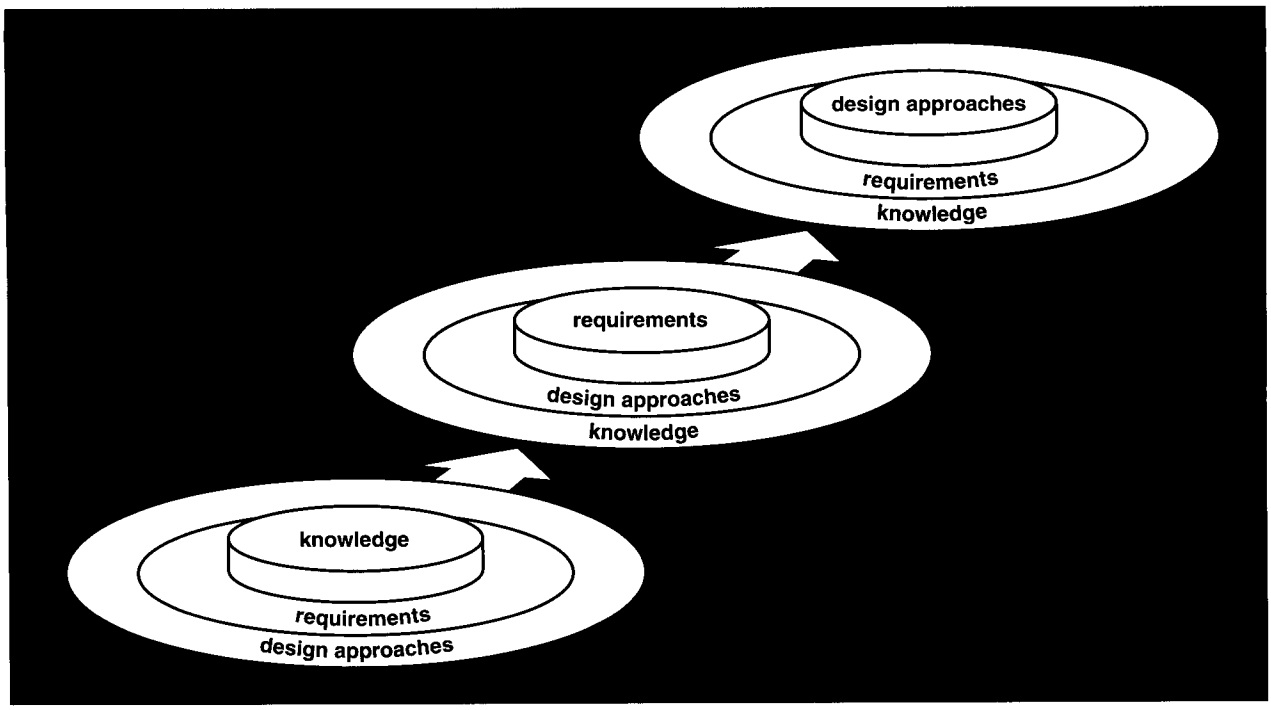


Figure 2. Shifts in emphasis over time

meetings was dialectic, or educational, in nature. This conflict was not personal, it did not appear to be hostile or antagonistic, and individuals did not appear to be disturbed by these interactions; in fact, they seemed to be learning from one another.

The design team was very deliberate in choosing the application domain and technical knowledge needed to complete the design task. While team members may have initially had their own ideas concerning the mapping between the requirements of the application and a design, a few members of the team succeeded in getting the team to focus on only a small subset of possible mappings. The group seemed to only recognize and assimilate technical knowledge seen as directly relevant to this subset. Requirements were also viewed within the boundaries of this subset as well. Around meeting #16 (early November), the group began to focus almost exclusively on design issues. After this point, the group seemed to "close its mind" to any new knowledge.

However, even before this point, we observed the reluctance of the group to significantly shift its current thinking. Two analysts were added to the project at relatively late dates: one in late September and one in early November. Both analysts were experienced professionals who were brought into the project because they possessed expertise in specific areas that was lacking on the team. On both occasions, the addition of another analyst appeared to have little effect on the direction of the project. Both analysts were initially given information intended to bring them up to speed regarding the history and status of the project. Nevertheless, work seemed to progress as before. Potential design approaches were not altered (or even considered for alteration) after the new analysts were added to the team.

Implications for Management

Knowledge acquisition, sharing, and integration are all activities that enable the software design team to learn what it needs for producing an appropriate design. Seldom are these activities explicitly accounted for in the design phase. Consequently, the time required for design is often seriously underestimated [12]. The length of time that a team spends in its learning phase depends on the breadth and depth of knowledge the team members bring to the project. It is also affected by the extent to which customers understand the requirements of the project. In the software design team we studied, there was some relevant technical knowledge but little application-domain knowledge, and customers were unclear on requirements. As a result, over 75% of the time devoted to the design phase of this project was spent in learning. Although the team had not learned everything it needed to know, time pressures forced it to move ahead with whatever knowledge it had gained. This insight can help project managers set more realistic estimates for the design phase of a project by including the required learning curve in the equation.

Also, during the learning phase, it is important to facilitate the open airing and exchange of ideas across all relevant domains of expertise. Project managers should not be too concerned during this phase if the team does not demonstrate visible progress toward developing design specifications, since it is generating the raw material necessary to move to the next phase of actually producing a design [14]. It is only if the group fails to move out of the learning phase midway through the project that overt actions should be taken.

Our observations also indicate the

Conflict was the mechanism for facilitating learning. *It was not a debilitating factor needing to be suppressed in the software design team.*

importance of including relevant team members from the beginning of the project. If new members (and their relevant expertise) are added after the group has come to closure in its learning phase, the group may be reluctant to deal with the new knowledge they bring to the team. Thus, knowledge at this point may not be incorporated easily into the group's work. If new members must be added during the project, project managers should take special care to ensure the knowledge brought by these members gets integrated into the team's current thinking.

There are implications for training as well. Conventional approaches to software design allow for training of team members in technical methods or tools, as necessary. Usually, this is done at the beginning of a project. Often, designers are physically removed from their day-to-day work environment in order to receive formal training. And typically, this training is separate and independent of the actual project activities. In our study, the software design team was involved in two formal training sessions—one involving database technology and one on Prolog—that were held on-site. These formal sessions did not seem to have much impact, primarily because the training did not focus on those things that were especially relevant for the project. When team members are immersed in a design activity, they are often unable (or unwilling) to acquire knowledge that cannot be immediately put to use. We recommend that formal training activities, when appropriate, be integrated into project activities rather than remain independent *(just in time training)*. One way to achieve this might be to have a technical trainer participate in a few design meetings so the training can be custom-tailored to the project.

Conflict was the mechanism for facilitating learning. It was not a debilitating factor needing to be suppressed in the software design team. In fact, we recommend consideration of formal techniques for managing conflict to help with knowledge acquisition, sharing, and integration. Two techniques for programming conflict into organizational decision making processes have been suggested: the devil's advocate decision program (DADP) and the dialectic method (DM) [6, 8]. In the devil's advocate method, an individual or group plays the formal role of critic in order to help a decision maker test the assumptions and the logic of the ultimate decision. The dialectic method pits a thesis against an antithesis. Most modern legal systems today are formal dialectic processes. Two sides exist, each with champions, and cases are made for each. This method is especially appropriate when a group is attempting to define problems and generate the necessary information for decision making under conditions of uncertainty, or where there is more than one way to solve a problem [7].

Formal methods for the use of dialectic techniques for strategic planning are presented by Mason [17, 18]. The strategic assumption surfacing technique (SAST) offers a method by which facilitated groups can identify and resolve underlying differences and similarities. Thus, it seems especially suited to heading off communication problems that may occur in such knowledge-intensive tasks as software design.

We recommend that at least one person within the group, perhaps the PM, serve in the capacity of a facilitator of programmed conflict. This individual would receive formal training in the DADP or the SAST as well as training in dialectic thinking and philosophy. The methods may need to be adapted somewhat to take into account the informal nature of the group interactions. We believe that formalizing these methods to the management of software design teams represents a potential area for significantly improving software design quality and productivity.

Creating the Team Memory

The team sought clarification of the requirements contained in the initial specification document by addressing a number of questions to their customer representatives. Interestingly, the customers were not willing to provide written answers, but agreed to attend meetings to be interviewed (specifically meetings #3 and #4). Thus, most of the information given to the team concerning the nature of the requirements of the system was given orally. And, interestingly, a large amount of this information was lost. A very influential customer (C2) attended three of the team's meetings, during which he spoke a great deal, usually in response to designers' questions. In fact, this customer offered many elaborate *scenarios of use* to explain his views, needs, and preferences. While the designers listened attentively, made comments, asked questions, expressed disagreement, and otherwise interacted with this customer, very little of the information contained in the interactions was recorded.

In one interchange, the designers asked C2 to prioritize three distinct approaches which seemed to be indicated by the initial specification. C2 did this within a very long discussion which included detailed and elaborate scenarios as well as modifications and clarifications of these three approaches. After this discussion, the designer taking notes wrote simply: "2-3-1." In fact, the documentation produced by the scribe designer during the two lengthy meetings with C2 (meetings #3 and #4) consisted of less than 150 words written on the

design team's copy of the question sheet. This documentation was used by the designer who took the notes to help produce a first draft of a design intent document. However, as was apparent from Meeting #8, when two other designers from the team took over the production of this document, even the small amount of information here seems to have been lost.

It was clear from our observations that the designers were learning from C2, gaining insights into his needs, and attempting to relate these to possible design alternatives. The process of interviewing C2 served to bring out information that was absent from formal requirements and was elusive, in that it was difficult to get from any direct source. However, it was also clear that the designers were not always able to integrate all of the new information they received. The designers were not just trying to ***accept*** information from C2. They were attempting to ***integrate*** this information into their own working model of the design task. In the beginning, these models were very sketchy [1]. Consequently, it was difficult to integrate requirements information into what they currently knew and understood: the information did not "stick," since they had yet to develop adequate "hooks" for it in their understanding of the problem.

The process of acquiring information and integrating this information was driven by *design bites*. The designers were only capable of integrating a design bite's worth of information into their current understanding of the design task, based on the ability of the new information to "attach" to that already integrated into the design. Therefore, a large amount of information from C2 was either lost or unnoticed. The discussions in later meetings often went back to "what C2 said," or "what C2 would say now." Some of the information provided by C2 had to be painfully (and only partially) reconstructed by the designers at later stages. Some of the information he provided in these earlier meetings was solicited again in a later meeting to which he was invited. A good example of this can be seen with respect to the issue of reusability. C2 was mainly interested in how reusability could be enhanced through the use of an object server. He made this clear in Meeting #3. Later during Meeting #12, D3 reiterated what he believed C2 wanted out of the object server. Since this requirement had not been captured in any design document, no one else on the design team seemed to pay much attention to it. In Meeting #18, D4 states "I think his (C2's) new notions of what he wants should be expressed directly to the group." In fact, these were not new notions; what he said he wanted was almost identical to what he stated in Meetings #3 and #4 and what D3 had perceived to be his desires back in Meeting #12.

We also observed cases in which design decisions became lost, or were forgotten, from one meeting to the next. Situations in which previously made decisions were questioned were fairly common. The following three episodes illustrate this phenomenon.

EPISODE 1: Meeting #5, August 19

PM relays the news that the acceptance test requested by C2 involves the ability to run a program that produces Nassi-Shneiderman diagrams. This program is written in Lisp. The design team has been less than enthusiastic up to this point of building the object server so that Lisp programs can be run. They are very reluctant to accept this as a valid test for their system.

- *C1: "A test of the product. He wants to be able to run this Nassi-Shneiderman program."*
- *D1: "See, I don't take that seriously, I really don't."*
- *D4: "Why didn't you tell us if you meant us to take it seriously. I mean why is it just mentioned in passing?"*

The team has either not remembered (or they did not take seriously) C2's comments from a specific scenario he gave them in Meeting #3 on August 12.

- *C2: ". . . So what happens is maybe you want to implement a Nassi-Shneiderman chart so (D4) goes off and he does some magic with Lisp and whatever and a week later the Nassi-Shneiderman classes get in there and they'll largely stay unchanged for a long time. . . ."*

EPISODE 2: Meeting #9, September 5

D3 has worked on a version of a requirements document. C1 points out that a major aspect of the project concerning its database functionality has been left out.

- *C1: ". . . what this is is an extension of the normal capabilities we've come to expect of databases, in terms of reliability, and failure recovery."*
- *D3: "That's what we're doing?"*
- *C1: "That was part of it."*
- *D3: "Not in any document I've ever read. I had three documents relating to . . . "*
- *C1: "That's the odd thing."*

EPISODE #3: Meeting #12, September 30

- *D4: "And if you store that, and the hierarchy changes in any way, you have to search through everything stored to reresolve those static references. I thought I won that argument a long time ago."*
- *D1: "Yes, but you did that in the background. The other problem—the problem with doing it your way is that in order to find the code here, you have to search every method of every single object all the way up, which makes . . . "*
- *D4: "That's logarithmic. Everything else is everything. If you're talking N versus log N . . . "*
- *D1: "Logarithmic?"*
- *D4: "Yes, the one path or set of paths up instead of the whole thing. That's how I won the argument five tapes ago."*
- *D1: "I don't think we ever did it on tape, I think we did it in my office."*
- *D4: "No, we did it here, and you came to a point where you sort of said 'Oh.' Even after that, I remarked, you actually agreed you were wrong and admitted you do that occasionally. I remember it quite vividly."*
- *D1: "Good for you. Well, I don't remember it, but I'll take your word for it."*

One possible explanation for design team "forgetting" is that every team member was not present at every meeting, and some designers were added to the project fairly late. For instance, D3 missed the first two weeks of the project. Because of this, he missed many relevant conversa-

tions covering significant aspects of the project which were discussed but not clearly documented. Participants in the early meetings understood these issues but D3 did not. This hampered him when he was put in charge of preparing a revised requirement document and highlights the difficulty of bringing new team members "up-to-speed."

Sometimes the design team could not "remember" some information because they considered it unimportant. Since the design team was leaning against implementing the object server in a Lisp/Flavors environment, they ignored relevant information such as the fact that the object server must be able to run a specific program in Lisp. Even when individuals remembered that a decision had been made, they often found it difficult (if not impossible) to recreate the logic, or the rationale, behind the decision (*"Why did we do it this way?"*).

Another aspect of the design meetings that contributed to a difficulty in "remembering" was the complexity and lack of structure in the discussions. Design is an intense cognitive activity and the project we observed for this study was no exception. In general, the discussions within the meetings were informal. Issues were not discussed hierarchically, but in a free-flowing, unstructured string of quasirelated episodes. A discussion of one issue seemed to trigger the discussion of new issues (see Appendix B for a detailed listing of the issues discussed in a fairly representative meeting). It was reported in [21] that design teams appear to have limited attention spans. The design team studied here did not attend to issues at great length. Many times, they never came to a decision on what to do about an issue, but were distracted from the issue, moving on to other topics. It appeared from our analysis that the group was aware of an unresolved issue only if it was raised again at some later time.

These team memory limitations were enough of a problem that the design team eventually requested access to the videotapes of their previous meetings. However, even though they were granted access to the tapes, they decided it would be too time-consuming to view them.

Implications for Management

Design teams have historically been expected to manage their collective "memory" in an *ad hoc* manner. The design team's *formal* memory is represented by its trail of formal documentation, such as functional specifications and users' manuals. An increasing number of tools exist to help design teams manage their *formal* memory: CASE tools, document preparation tools, and modeling software provide help in this area by managing the formal record of the output of the various design stages. They do little, however, to provide a record of the *process* of the design.

Prototypes provide limited help in this area since they are products of the design team's consensus model of the customers' requirements. They represent output (a model of a system), not process. A prototype can trigger a conversation which includes customers' scenarios of product use, but does not provide a means to capture *a priori* the information from these conversations.

The team's *informal* memory is much more complicated and more difficult to manage. It consists of the material scrawled by individuals in the margins of their personal copies of formal documents, the notes on the blackboard on any given day ("Do Not Erase"), and the thoughts and impressions of the individual team members themselves.

Software design teams could benefit from tools that are intended to record and capture the *process* of software design. Such tools would provide methods for capturing *design rationale* [3, 5, 19] including *scenarios of use* as supplied by customers or suggested by designers. Groupware tools that allow the capture, storage, and retrieval of the design *process* information have been suggested for software process management [11, 15, 16]. Such tools may use the design process as input. For example, on-line conversations about various issues and videotapes of design meetings, can be stored and processed so that information can be retrieved. Such tools could also keep track of key issues raised during group meetings and the position, if any, taken by the group with respect to these issues.

Relative Participation by Team Members

It was reported in [21] that team members in design groups participate unequally. In a study of 17 large projects, it was found that the early phases of software design projects were dominated by a small coalition of individuals, occasionally even a single individual [12]. Our observations are consistent with these findings. We identified 3 individuals out of the 10 members of the software design team who seemed to dominate the design process. One of these individuals was the customer representative, C1 and the others were two designers (D1 and D4) who emerged as leaders of the design effort.

C1 was a customer representative with a rich technical background and excellent communication skills who used examples and scenarios of use to convey information about the customers' needs and preferences effectively. He drew on his technical background to frame these examples and help the designers to understand the system requirements.

D1 and D4 were the only project team members who attended all of the meetings. They participated more frequently than other team members. In fact, counts of individual speech acts from the transcripts of the meetings reveal that these individuals actually spoke more often than any of the others by a factor of nearly 2 to 1 (see Figure 3). D1 and D4 worked hard, both in and out of the group meetings, and performed more tasks than they were explicitly asked to do. They routinely contacted outside experts, searched for relevant research papers, and discussed unresolved issues with the customers. They both developed plans for addressing the design (not all of which panned out) and they presented these suggestions to the team, after discussing them with the PM.

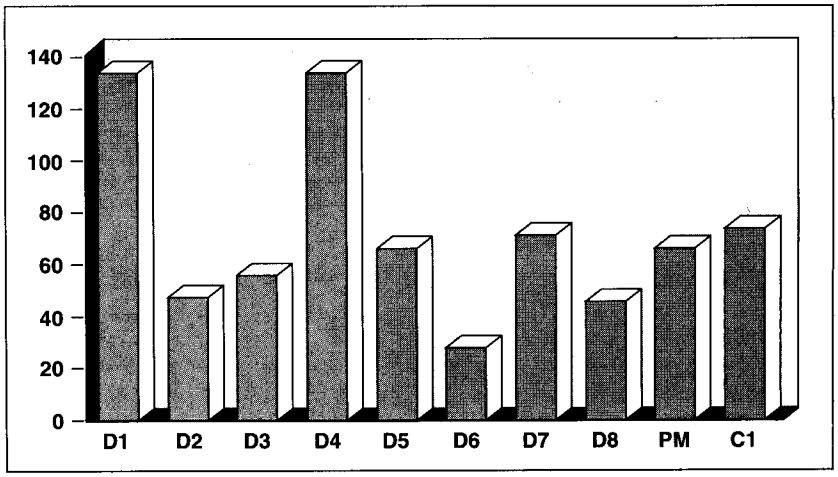
D1 impacted the team effort both technically and administratively. As described previously, he tried to seek out and integrate new technical knowledge into a framework for producing the design. He also influenced the team process by taking the initiative in the administration of team duties. He routinely volun-

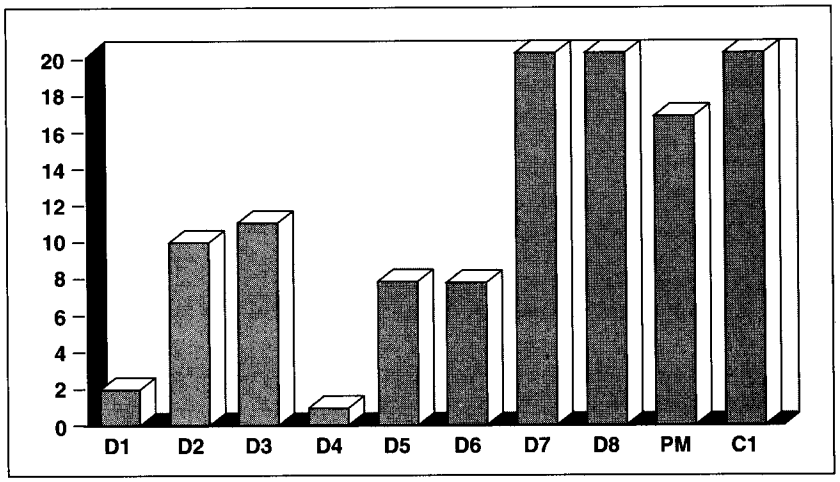
teered to coordinate group activities (assemble lists of questions, solicit input and produce documentation) and he actively led the group (in terms of both meeting and project management) for several of the meetings.

D4's influence on the project was largely technical. He exhibited considerable technical expertise and other team members regularly sought his help and deferred to his opinions. He seemed to be something of a loner, not especially interested in reaching consensus. Without being overpowering, he would do his own work and offer the results to the group—it was D4 who built the Prolog prototype in the first few weeks of the project. When D4 initially suggested building the prototype, the PM was not supportive. But D4 built the system anyway, explaining that he wanted to do this as a learning tool. Interestingly, the Prolog prototype that he built became an important piece of the finished system.

D1 and D4 formed a very influential coalition over the course of the project. They were influential not only in determining the overall design approach and its subsequent decomposition, but also in the allocation of responsibilities. D4 ended one of the meetings by reading from his notes (written on a Styrofoam coffee cup!), where he summarized the project status and suggested assignments for team members; the others assented.

We surveyed the team members to learn about their backgrounds, knowledge, and expertise. Interestingly, D1 and D4 had the fewest number of years of professional experience (2 and 1 years, respectively), although they had a number of years of programming experience (7 and 11) in a variety (5 and 20) of languages (see Figure 4). We infer that most of their experience was in an academic or personal computing environment. We also asked each team member to identify (for 12 project-related knowledge areas) those individuals on the team who they felt were knowledge resources in these areas. We studied the responses to see if there were any knowledge-related differences between the emergent leaders (D1 and D4) and the other team members. We recorded the number of times each team member was mentioned by others as being a knowledge resource. We also noted the number of knowledge areas for which each person received votes. This is a measure of the breadth of an individual's expertise. See Figure 5 for a summary of these results.





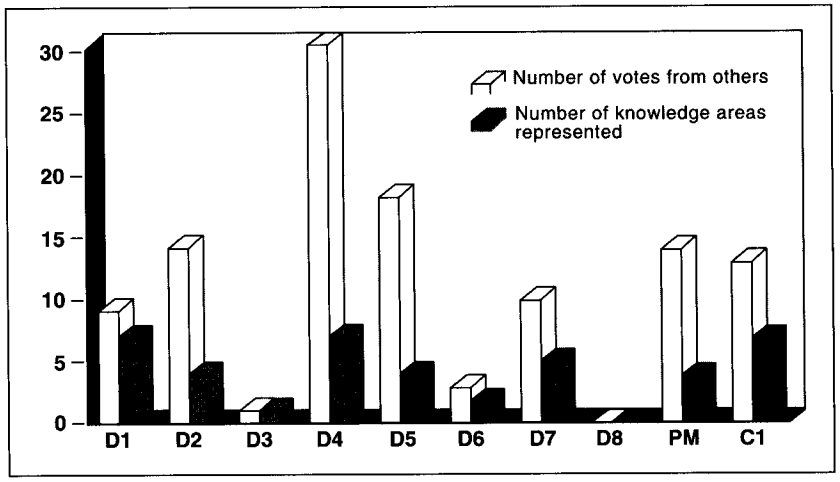


Figure 3. Number of statements per meeting

Figure 4. Years of professional service

Figure 5. Expertise as perceived by team members

We have historically valued both technical and communication skills in software designers. **We suggest organizations put programs in place for developing these skills** *in more depth.*

Interestingly, D4 was mentioned most often by teammates as a knowledge resource, receiving 30 votes. Also, D4's votes covered 7 knowledge areas, implying that he had a breadth of knowledge as well as depth and expertise. D1 and C1 (the customer representative) were also mentioned in 7 knowledge areas. No one else on the team received votes in more than 7 knowledge areas.

Implications for Management

The conventional wisdom for hiring programmers and designers values experience, where experience is often equated with knowledge. Of Boehm's [2] five basic principles of software staffing, three are especially related to knowledge and expertise. The basic premise of *The Principle of Job Matching* involves fitting the task to the skills and motivations of the available staff. Operationally, however, this usually involves matching individuals' technical experience (software environments, operating systems, databases, programming languages, application areas) with the technical requirements of the task. *The Principle of Team Balance* suggests that an appropriate mixture of knowledge, technical skills, and personality characteristics are especially important. *The Principle of Top Talent* recommends the use of fewer and better people.

In the design team we studied, the two individuals who were identified as the most knowledgeable were also the least experienced. This supports the findings of other studies that breadth of experience is a better predictor of individual performance than years of experience [13, 22]. In spite of this, years of experience is often used as a key input into staffing decisions.

We suggest that a better approach would be to develop a "knowledge profile" for each member of the software design and programming staff. The *Principle of Job Matching* could be operationalized by matching knowledge profiles of staff members to the knowledge profile of a particular project. These knowledge profiles could also be used to ensure, as much as possible, that requisite knowledge, skills, and abilities are appropriately distributed among the members of the team, in accordance with the *Principle of Team Balance*. Where this is not possible, management would need to be aware of any knowledge gaps that need to be addressed.

There were 10 members of the design team that we studied. However, three members dominated its functioning. They dominated not only because they were the most knowledgeable on the team, but also because they had the skills necessary to exchange and integrate knowledge. If the *Principle of Top Talent* had been adhered to, a design team with fewer individuals might have been adequate. It is often difficult to identify, *a priori*, who the key team members will be. Slack, in the form of extra members, is necessary in order to increase the probability that key contributors are included. Identification and management of knowledge profiles could help reduce this need for slack. An organization may, however, choose to include a few extra members on a design team in order to move them up the learning curve.

We have historically valued both technical and communication skills in software designers. We suggest that organizations put programs in place for developing these skills in more depth. On the technical side, individuals with the intelligence, talent, and desire should be exposed to a variety of knowledge areas through appropriate task assignments and formal training. On the communication side, we recommend that special attention be paid to developing team building, negotiation, and teaching skills. Due to the abundance and importance of verbal information received by team members, it is important that team members develop good listening skills and the ability to translate this verbal information into a form that can later be retrieved.

Software designers must be knowledgeable in the application domain. The software design team in this study, like many others, had designers who were knowledgeable in the techniques of computer science. They lacked some knowledge in the application domain (i.e., object servers). Consequently, significant learning costs were incurred by this team. Through this experience, however, these designers acquired knowledge that could only be obtained by going through this learning process.

Conclusion

Observing a software design team closely has allowed us to gain some important insights into the design process. We observed needs that were not met within the project life span. We were surprised to see how important context-sensitive learning was to the design process. We were surprised at how much information was presented to the team and never captured. We were surprised to see that the requirements determination did not end cleanly, but was a lengthy process that seemed to "shut down" based more on project timing than on achieving a full understanding of the requirements. And we were also surprised at the extent to which knowledge and expertise was the force behind participation and leadership of the design process.

These observations, however, are less surprising if we acknowledge the criticality of knowledge acquisition,

sharing, and integration activities. Adopting a knowledge perspective leads to some specific recommendations for managers of software design efforts. One obvious recommendation is to increase the amount of application domain knowledge across the entire software development staff. Assigning one or two individuals with deep application domain and technical knowledge to a design project can significantly reduce the learning time involved. Another recommendation is to actively promote the acquisition, sharing, and integration of knowledge within a design effort through team facilitation techniques and to formally recognize these activities by allocating time to them. Explicitly managing conflict as a way to facilitate learning has been proposed as one way of doing this. Finally, it is also important to recognize that much of the information that needs to become part of the team's memory is not captured formally, particularly in standard documentation. New computer-based tools are needed to easily and unobtrusively capture this process-based information.

The software design project examined in this study was an exploratory R&D project undertaken within a research organization. An unanswered question is the extent to which we would observe the same types and levels of activities related to knowledge acquisition, sharing, and integration along with the same patterns of participation and leadership in software design teams engaged in commercial application development. It seems liked that the frequency of behaviors we observed exists on a continuum dominated by how much is already known about a software product. For projects involving a new application area in which considerable learning is required to produce a design, it is likely that our observations and findings would be very similar. In projects building well-understood products requiring little learning, our observations might have been quite different. A broader range of empirical research on software design teams is necessary to determine how far our observations and findings generalize to projects in other organizations.

Acknowledgments

The authors wish to acknowledge the support of the Software Technology program at Microelectronics and Computer Technology Corporation in Austin, Texas. We thank Jeff Conklin and Herb Krasner for videotaping the meetings, and extend special thanks to Herb Krasner for invaluable input throughout this project.

Appendix A: Transcript Coding Methods

The transcripts of videotaped group meetings were broken into speech acts (by speaker) which were then classified according to the following predefined coding scheme:

Expository
- —offers opinion
- —offers clarification
- —agrees
- —disagrees
- —modified previous position

Acquisitive/Facilitative
- —interprets
- —asks

Other

To test the interrater reliability of the coding scheme, a subset of the transcripts was independently coded by three employees at the research site. None of these individuals had participated in the design project or in this research project, and all had experience with classification of interaction data according to coding schemes. For these subsets, the average percentage of interrater agreement was 66%. Sources of discrepancies did not appear to be systematic across coders.

Appendix B: Example Issues

The following issues were addressed (in chronological order) in meeting #8, September 3.

Agenda
Staffing
Agenda
Formalize design intent document
- Time frame for competing
- Changing, dynamic nature of requirements
- What is this document called?

Quality of draft of requirements document
- Project status
- Suggest change to agenda
- Discuss requirements

Goals/nature of requirements documentation
Project history, background
Text processing, sharing files for project management
Document history—who wanted this document?
Schedule—can we wait 2 weeks?
User's manual/reference manual/functional spec
Communication with customers using functional spec
Text processing, sharing files for project management.
Distribution of document
How much to include in document (hide anything?)
Customer's application program (Nassi-Shneiderman) for acceptance
Project overview—original intent and goals
- Possible implementation languages (Smalltalk, Flavors, Objective C)
  - Store objects and methods
  - Base classes
  - Concurrent use
  - Translators
- Object-oriented systems with respect to requirements
  - Objective C, Flavors, . . .
  - Prolog
  - Translators

Nature of requirements document
Actual requirements
- Access objects from different languages

Actual requirements
- Translators
- Multiple copies
- Support for object-oriented languages

Requirements vs. design decisions
Languages
- Character string translation

Prolog prototype
- Flavors
- Server—access, store objects
- How it works, what it does

Prototype vs. requirements document for communicating design
- Instructions for using prototype
- Functional spec

How to express requirements
Define prototype in relation to requirements
- Document prototype
- Environment: C program that uses objects
- Store and access objects
  - Convert to C
- Methods
- Relation to objective C

Project plans:
- Proceed with prototype
- Object-manager (interface to disk)

- Flavors interface
- Difficulty of doing design . . .
  - Translate into Smalltalk or Flavors
- Local objects
  - Object name . . . how does it work?
    - Flavors
    - Define? pointer?
    - Surrogate objects
  - Accessing objects
  - Object name
    - Character string, pointer

Requirement: Lisp machine—include in spec?

- Send messages
  - Character strings, surrogate
  - Object pointer
  - Object name, object id
- Send messages
  - Flavors objects
    - Same in Prolog
    - Flavors details
  - Access to objects is external access
  - Flavors with Biggertalk, e.g.,
  - Internal vs. external objects
    - Transient/permanent, e.g.,

Implementation issues, project management: who decides this?

- Flavors in specification (designers don't want it)
- Scope of specification, requirements

Server—base set of classes, objects

- Arrays, lists
- Define
- Not on server
- Methods in separate environment
  - Class of integers, class of arrays
- Base set, base classes

Server functions

- Data types

Prolog, handling integers

- Add to a string
- Tag to identify language
- Methods
  - Storage tracks
  - Speed

Portability

- Hardware
- Track size, block size
- Database access speed

Who's going to do what? Assignments for the near future

References

1. Adelson, B. and Soloway, E. The role of domain experience in Software Design. *IEEE Trans. Softw. Eng.* (Nov. 1985), 1351–1360.
2. Boehm, B.R. *Software Engineering Economics*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1981.
3. Burgess-Yakemovic, K.C. and Conklin, J. Report on a development project use of an issue-based information system. In *Proceedings of CSCW*. ACM, New York, 1990, pp. 105–118.
4. Carroll, J.M., Thomas, J.C. and Malhotra, A. Clinical-experimental analysis of design problem solving. *Design Studies 1*, 2 (1979), 84–92.
5. Conklin, J. and Begman, M. gIBIS: A tool for all reasons. *J. Am. Soc. Inf. Sci.* (1989), 200–213.
6. Cosier, R.A. Methods for improving the strategic decision: Dialectic versus the devil's advocate. *Strategic Manage. J. 16* (1982), 176–184.
7. Cosier, R.A. and Dalton, D.R. Competition and cooperation: Effects of value dissensus and predisposition to help. *Human Relations 41*, 11(Nov. 1988), 823–839.
8. Cosier, R.A. and Schwenk, C.R. Agreement and thinking alike: Ingredients for poor decisions. *Acad. Manage. Exec. 4*, 1 (1990), 69–74.
9. Curtis, B. By the way, did anybody study any real programmers? In *Empirical Studies of Programmers*, E. Soloway and S. Iyengar, Eds., Ablex, Norwood, N.J., 1986.
10. Curtis, B. Technology transfer in knowledge-intensive organizations. In *Technology Transfer in Consortia and Strategic Alliances*, R. Smilor, Ed., Roman and Littlefield, Savage, Md, 1992.
11. Curtis, B., Kellner, M.I., and Over, J. Process Modeling. *Commun. ACM 35*, 9 (Sept. 1992), 75–90.
12. Curtis, B., Krasner, H., and Iscoe, N. A field study of the software design process for large systems. *Commun. ACM 31*, 11 (1988), 1268–1287.
13. Curtis, B., Sheppard, S.B., Kruesi-Bailey, E., Bailey, J. and Boehm-Davis, D. Experimental evaluation of software specification formats. *J. Syst. Softw. 9*, 2 (1989), 167–207.
14. Gersick, C.J. Time and transition in work teams: Toward a new model of group development. *Acad. Manage. J. 31*, 1 (1988), 9–41.
15. Krasner, H., McInroy, J. and Walz, D.B. Groupware research and technology issues with application to software process management. *IEEE Trans. Syst. Man. Cybern. 21*, 4 (July/Aug. 1991), 704–712.
16. Krasner, H., Terrel, J., Linehan, A., Arnold, P. and Ett, W.H. Lessons learned from a software process modeling system. *Commun. ACM 35*, 9 (Sept. 1992), 91–100.
17. Mason, R.O. A dialectical approach to strategic planning. *Manage. Sci. 15* (1969), 403–414.
18. Mason, R.O. and Mitroff, I.I. *Challenging Strategic Planning Assumptions*. Wiley and Sons, New York, 1981.
19. Moran, T.T. and Carroll, J.M. (Eds.) *Design Rationale: Concepts, Techniques and Use*. Erlbaum, Hillsdale, N.J., to be published.
20. Myers, W. MCC: Planning the revolution in software. *IEEE Softw.* (Nov. 1985).
21. Olson, G.M., Olson, J.S., Carter, M.R. and Storrosten, M. Small group design meetings: An analysis of collaboration. *Human-Comput. Inter. 7*, 4 (1992).
22. Sheppard, S.B., Milliman, P., Curtis, B. and Love, T. Modern coding practices and programmer performance. *Computer 12*, 12 (1979), 41–49.
23. Soloway, E. What to do next: Meeting the challenge of programming-in-the-large. In *Empirical Studies of Programmers*, E. Soloway and S. Iyengar, Eds., Ablex, Norwood, N.J., 1986.
24. Walz, D.B. A longitudinal study of group design of computer systems. Ph.D. dissertation, University of Texas, Dec. 1988.

CR Categories and Subject Descriptors: D.2.9 [**Software Engineering**]: Management; D.2.10 [**Software Engineering**]: Design; K.6.1 [**Management of Computing and Information Systems**]: Project and People Management; K.7.2 [**The Computing Profession**]: Organizations

General Terms: Management

Additional Key Words and Phrases: Case study, empirical studies of software development, requirements determination, software design teams, software management

About the Authors:

DIANE B. WALZ is an assistant professor of information systems at the University of Texas at San Antonio. Current research interests include group processes in software development, problems of outsourcing, and creativity and software design. **Author's Present Address:** Division of Accounting and Information Systems, University of Texas at San Antonio, 6900 N Loop 1604 W, San Antonio, TX 78249; email: 1ISDXW@ UTSAVM1

JOYCE J. ELAM is the James L. Knight Scholar in management information systems at Florida International University. Current research interests include the competitive use of information technology to support both individual and group decision making. **Author's Present Address:** Department of Decision Sciences and Information Systems, Florida International University, University Park, Miami, FL 33199; email: elamj@servax.bitnet

BILL CURTIS is former director of the Software Process program at the Software Engineering Institute (SEI) at Carnegie Mellon University. He continues to work with the SEI and is helping to establish a software quality institute at the University of Texas at Austin. Current research interests include improving organizational capabilities for developing software, empirically based models of software design, and software measurement and design process. **Author's Present Address:** 3644 Ranch Creek, Austin, TX 78730; email: bcurtis@cs.utexas.edu

Twenty of the 34 **ACM Special Interest Groups** (SIGs) held elections last June for terms running from July 1, 1993 through June 30, 1995. The newly elected officers are listed below. For a full list of current SIG chairs, see the ACM masthead in this issue; for a complete list of SIG officers, contact SIGS@acm.org.

Newly Elected ACM SIG Officers

SIGACT
F. Thomson Leighton
Chair
Jeffrey S. Vitter
Vice Chair
Michael Luby
Secretary/Treasurer
Michael T. Goodrich
Member-at-Large
Vijaya Ramachandran
Member-at-Large

SIGAda
Hal Hart
Chair
Jerry Mungle
Vice Chair, Mtgs. & Confs.
Edward Colbert
Vice Chair for Liaison
Brad Balfour
Secretary
Russell R. Plain
Treasurer
Rudolf Landwehr
International Rep.

SIGAPL
Dick Bowman
Chair
Stuart Yarus
Vice Chair
Michael Kent
Secretary/Treasurer
Dick Holt
Member-at-Large
Christopher H. Lee
Member-at-Large
David M. Weintraub
Member-at-Large

SIGARCH
David A. Patterson
Chair
Jean-Loup Baer
Vice Chair
Alan Berenbaum
Secretary/Treasurer
Mark D. Hill
Board of Directors
Mary Jane Irwin
Board of Directors
Norman P. Jouppi
Board of Directors
Alan J. Smith
Board of Directors

SIGART
Stuart C. Shapiro
Chair
Alan M. Frisch
Vice Chair
Lewis Johnson
Secretary/Treasurer

SIGBIT
George M. Kasper
Chair
Janice C. Sipior
Vice Chair
Don Hardaway
Secretary/Treasurer

SIGCAS
C. Dianne Martin
Chair
David Bellin
Vice Chair
Deborah G. Johnson
Secretary/Treasurer

SIGCHI
James R. Miller
Chair
Michael E. Atwood
Executive Vice Chair
Gene Lynch
Vice Chair for Conferences
Vivienne Begg
Vice Chair for Operations
Beth Adelson
Vice Chair
Clare-Marie Karat
Vice Chair for Finance
Jakob Nielsen
Vice Chair for Publications

SIGCPR
Thomas W. Ferratt
Chair
Albert Lederer
Vice Chair
Catherine M. Beise
Secretary
Bruce E. Breeding
Treasurer

SIGCSE
Lillian (Boots) Cassel
Chair
G. Michael Schneider
Vice Chair
Henry M. Walker
Secretary/Treasurer
Janet Hartman
Board of Directors
J. Paul Myers
Board of Directors
Margaret Reek
Board of Directors

SIGDA
Jim Cohoon
Chair
Joanne DeGroat
Vice Chair
Robert A. Walker
Secretary/Treasurer

SIGDOC
Nina Wishbow
Chair
Stephanie Rosenbaum
Vice Chair
Barbara Mirel
Secretary
Katherine Haramundani
Treasurer

SIGGRAPH
Mary C. Whitton
Chair
Sylvie J. Rueff
Vice Chair
Steven M. Van Frank
Treasurer

SIGMETRICS
Linda S. Wright
Chair
Donald Towsley
Vice Chair
Daniel A. Reed
Secretary/Treasurer
Domenico Ferrari
Board of Directors
Mike Molloy
Board of Directors
Richard Muntz
Board of Directors
Randolph Nelson
Board of Directors

SIGMOD
Won Kim
Chair
Laura Haas
Vice Chair
Michael Carey
Treasurer

SIGNUM
John R. Gilbert
Chair
Robert S. Shreiber
Vice Chair
Christian Bischof
Secretary/Treasurer
David H. Bailey
Board of Directors
Alan Edelman
Board of Directors
David M. Gay
Board of Directors
Andreas Griewank
Board of Directors
Stephen G. Nash
Board of Directors
Maria Elizabeth Ong
Board of Directors
Lloyd N. Trefethen
Board of Directors
David W. Walker
Board of Directors

SIGPLAN
Brent T. Hailpern
Chair
Barbara Ryder
Vice Chair for Conferences
John Pugh
Vice Chair for Operations
Bernard Lang
Secretary
Mary Lou Soffa
Treasurer
Marina C. Chen
Member-at-Large
Ron K. Cytron
Member-at-Large
David W. Wall
Member-at-Large

SIGSAM
Erich Kaltofen
Chair
Stephen Watt
Vice Chair
Bruce W. Char
Secretary
Gene Cooperman
Treasurer

SIGSMALL/PC
Hossein Saiedian
Chair
Gerald P. Crow
Vice Chair
Richard McBride
Secretary/Treasurer

SIGSOFT
Lori Clarke
Chair
John Gannon
Vice Chair
David Notkin
Secretary/Treasurer
Barry Boehm
Member-at-Large
Thomas Ostrand
Member-at-Large
Mary Lou Soffa
Member-at-Large