Software Teams

Each quarter, the Practical Programmer will discuss practical ways to build better software, taken from the experience of working programmers. This month, Marc Rettig describes a successful development group organized as a "Structured Open Team."

I have often heard the phrase, "We see what we know." As technicians, we concentrate on technical ways to manage complexity: abstraction, design techniques, high-level languages, and so on. That is what we know best. But when the tale is told of a project that failed, the blame is often laid not on technical difficulties, but on management and interpersonal problems.

In the last six months, I have seen firsthand how attention to the social organization of a software team can make a big difference in the success of a development project. I work in a "Research and Development" group. "Research" means that some aspects of the project are experimental—we do not know for sure what is going to work. "Development" means we are expected to produce high-quality software for real users. So while we want to encourage creative thought, we must pay heed to the lessons of commercial software developers in quality assurance, testing, documentation, and project control.

Our all-wise project leader decided we also needed to pay heed to the lessons of sociology. In particular, we began to apply the ideas found in Larry Constantine's work on the organization of software teams. Our efforts have resulted in a team that is productive, flexible, and comfortable. I thought these qualities are unusual enough to merit a column on the subject.

The Birth of a Software Team

Before most of our team was assembled, two of the present members (the project leader and the lead designer) spent nearly a person-year working closely together. They accomplished three things:

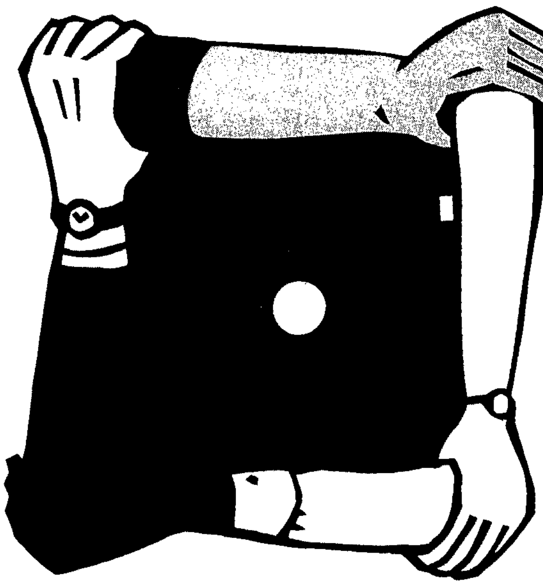
- They sold upper management on the design concept and rallied user departments to participate in the development process.
- They agreed on the "guts" of the design, and wrote a set of design documents which described the components and their relation to one another.
- They laid out a plan for the first six months of development, including personnel requirements, time estimates, and complexity estimates.

About the time they completed this, the rest of the team showed up for work: two programmer/analysts with advanced degrees, a seasoned manager with good technical knowledge, and assorted part-time secretarial support.

Our first project was to organize ourselves as a team, which we accomplished in a slightly unusual setting. We went on a two-day retreat. The whole team, including spouses and day-care staff, went to a hotel. For two days and one night we participated in team-building exercises and horseplay. This might sound a little corny, but it did a lot to make strangers feel they were part of the team, and that the team had a purpose. What did we do for two days?

- We all took a "social styles inventory" test, then plotted the team's collective profile on a big sheet of paper. This led to an insight that otherwise may have gone unspoken: almost all of us are introverts, including the team leader.

BY
Marc Rettig



Contemplating this discovery helped us understand potential communication problems. The leader had a chance to say, "so, if I don't come and talk to you, it's not because I'm ignoring you. It's just the way I am."
- We worked together on a mission statement for the team, which includes a clear description of our purpose and who we serve. This was helpful in two ways. Unlike many teams, we have all agreed on a common purpose. Further, we have a nice guideline for deciding between conflicting demands: "which alternative most clearly relates to the group's mission?"
- We learned about Constantine's notion of "structured open teams," set guidelines for meetings, and planned ways to help each aspect of the project get the attention it deserves. Very often only the project leader thinks about these issues. Sometimes no one thinks about them. Working together on our team organization not only made for increased unity, it gave us an opportunity to

ILLUSTRATION: DENNIS AHLGRIM

practice problem solving as a team.

- We played and gabbed. We played cards, rowed boats, and swapped life stories. Again, this might sound corny, but it helps team members feel comfortable with each other and builds a unity that is otherwise hard to achieve. I know of one team leader that enrolls his staff in bowling league for the same reason.

The details of our organization were worked out over the next few weeks, and are still open to adjustment. But the two days built an atmosphere of cooperation and willingness to communicate about the nontechnical aspects of our work that has lasted ever since.

A "Structured Open Team"

Most of the software teams I have been a member of or heard about were organized in a hierarchy—what Constantine calls the "closed paradigm." One person has responsibility for the project, and there are supervisors under that person, each responsible for a major component or aspect of the project. Each supervisor tells the people down the line what to do, and so on. Sometimes this works very well. Since the group's direction comes from the leader, decisions can be made quickly and everyone understands exactly what needs to be done to complete his or her part of the task—*If* the leader is good at the job. If not, the project and the team members suffer.

Structured open teams flatten hierarchical organization. One team member has the permanent role of "Technical Leader" (or Project Manager, or whatever you want to call it). This person is accountable to the rest of the company for the success of the project, and holds final veto power over the group's decisions. Other than that, the team is a group of equals, making decisions by consensus.

Further structure comes from a recognition that certain roles are essential to the success of any software team. At a minimum, these include: decision making, coordination, information management, critical feedback, application domain knowledge, and technical/analytical functions. While many teams permanently assign a role to each team member, this is "opened" by Constantine so that individuals, subgroups, or the entire group are given roles to play, and role assignments can be rotated to suit changing situations and special abilities.

Decision by Consensus

Team participation in decision making has many advantages. It provides creative diversity, combines abilities, shares the load, and makes for built-in monitoring of the development process. But there are dangers—operational overhead, communication and personality problems, and the greater inertia of a group of people rather than one person.

Structured open teams attempt to maximize the advantages and minimize the dangers, with the feeling that the benefits of consensus are worth the trouble. As Constantine puts it, "Genuine consensus on technical design is important because it means full group resources have been used and a creative integration of diverse contributions has been achieved. Moreover, consensus decision making also increases solution ownership." The phrase "solution ownership" characterizes what makes a good team so effective and such a pleasure to be part of.

We have had little trouble with consensus-building stalemates or hung juries. In any case, the team leader has final say, which avoids endless and pointless discussion. For teams with strong opposing opinions or personality conflicts, the role of facilitator or process leader, described below, becomes an important tool for consensus building.

But, back to our team. We knew we wanted to apply some of these ideas, and began to try them out in meetings. In the early stages of design, it was necessary to hold a lot of meetings. And for the first few weeks, we were deliberately self-conscious about how our work was being conducted, what was happening in our meetings, and how the idea of a structured open team was going to affect our daily lives together. Very often, meetings contained parenthetical discussions about process and organization. At the time it was a bit frustrating, but we are quite happy with the result.

Meetings

As part of the planning for the project, we broke the work into "modules," each supposedly about two weeks' work for one person. Each module has several "phases," and there are group reviews for the design and coding phase. When we are going full steam, there can be several reviews a week. On top of that, we hold a staff meeting every Friday morning, and there are occasional planning meetings. There is a real danger that we spend so much time in meetings or preparing for meetings that we have little time left for programming.

Seeing that danger, we laid some ground rules to make the meetings as productive as possible. To begin with: "No more than one meeting a day." And: "Keep meetings short." "Hah!," you say. Well, one thing we do when scheduling is to think, "we've got no business spending more than an hour on this topic. So let's schedule the meeting for 4 p.m." Works like a charm.

Since design and code reviews involve preparation for everyone (to read the documents to be discussed), we try to relieve the burden a bit. When someone distributes a document and schedules a meeting, everyone is invited. But one person is designated "principle reviewer." This person is honor-bound to study the document carefully, prepare an agenda of discussion topics for the meeting, and to give written comments to the author. Others on the team are encouraged to do these things, but do not have to live with a guilt complex if they are too busy to give every document their complete attention. The job of principle reviewer rotates among the technical staff.

For the meetings themselves (all meetings, not just review meetings), we have adopted several of Constantine's ideas about "roles." He describes roles that need to be filled on a successful team: technical leader, process leader, information manager, technical and process critic, and domain expert. Some of these roles are permanently assigned to one person (the role of technical leader, for example), while others may change. We explicitly assign some roles at the beginning of a meeting. Every meeting has a scribe and some meetings have a facilitator.

The Scribe

The scribe writes down important decisions, open questions, assignments, anything that is deemed worthy of incorporation into our "group memory." It is important to designate a scribe at the beginning of the meeting. That person becomes aware of the responsibility and listens to the discussion with a different ear, trying to pick out notable items. The others can relax and focus on the problem. Fewer things slip through the cracks.

Constantine points out that this is a demanding role, and can hamper the scribe's participation in the meeting. We have found that some meetings are easier to record than others. In any case, the scribe is responsible for producing a written record of the meeting for archival purposes, which can vary from a few lines to several pages. Mostly for this reason, nobody wants the job. So we rotate. Of course, the author of the document being reviewed cannot be the scribe, so in small meetings we wind up tossing the coin between two other participants.

The Facilitator

The facilitator is responsible for making sure that a meeting is productive—that we accomplish our purpose, that we do not get sidetracked, that everyone gets a chance to participate, and that we stop when we are supposed to. After the first few review meetings we stopped using a facilitator because we became so used to the process that we were self-facilitating. But for planning and design meetings, it is a wonderful thing to have a good facilitator.

In our case, we have one person who is very good at this, so he often gets called into meetings to facilitate even when he has little technical understanding of the subject under discussion. He will stand up by the whiteboard or sit with a notepad in his hand, quietly documenting the course of the discussion. Once in a while he will interrupt: "Let me make sure I understand. You just decided to do ⟨x⟩, right? Now, who will be responsible for that, and when should it get done?" Or, "Excuse me, but it sounds like we've wandered onto something that is important, but has nothing to do with what we're supposed to be talking about. Let me write this down as something to discuss later, then let's proceed with the subject at hand." Or, "Fred, have you been trying to say something?"

With a good facilitator in the room, much is accomplished and very little slips through the cracks. It is important that the facilitator be neutral, and clearly responsible only for the process, not the outcome of the meeting. As Constantine says, "the role is based on making the best possible use of the collective resources of the group, sustaining a process that maximizes participation, collaboration, and individual ownership of or 'buy in' to the final product . . . . The process leader involves everyone and defends everyone . . . . When the facilitator is also a participant, it is too easy to maneuver the group and manipulate the outcome, despite all the best intentions . . . ."

Finally, probably the most important and most demanding job of the facilitator is consensus building. "The effective process leader must always be alert to opportunities for consensus and must guide the group toward consensus whenever possible without force or premature closure."

You may not have the luxury of having someone outside the technical staff who can serve as a facilitator. It would work to have the facilitator role rotate like that of the scribe. In fact, I understand some teams use tokens to represent rotating roles. Maybe the scribe has a giant pencil, the critic has a devil button, and the facilitator has uh, a cattle prod? The tokens are used to make the role changes plain to everyone. For example, say the discussion turns to something the scribe is deeply involved in, which makes it impossible to give the role the attention it requires. So the giant pencil gets passed to someone else as an explicit signal: "I am no longer the scribe. You are. (Shut up and start writing)." This kind of role changing happens in most meetings without anyone noticing. If you do not do something to keep track of who is playing what role, misunderstanding and miscommunication can result.

The Critic

We are not explicitly assigning all of the roles that Constantine describes. One interesting idea is to assign someone to be "devil's advocate" in every meeting. "Critical feedback is so essential to successful teamwork that the structured open team institutionalizes and legitimizes the function . . . . It is known that opposition and dissent are absolutely vital for successful functioning of open paradigm systems. The stigma of this function is lessened by creating a formal technical/process critic role. This position does not have a default incumbent but is assumed ad hoc by anyone at any time. . . . .By critiquing decisions, playing devil's advocate, putting down pointless debate, debunking empty hyperbole, or exposing weak and wishful thinking, the person in this role provides a valuable and indispensable service to the

Programming in Black and Blue

Each quarter, the Practical Programmer relates techniques that are bringing success to working programmers. "Programming in Black and Blue" will record the experiences of the less fortunate, who have learned the hard way that it is sometimes difficult to put into practice what has been "preached."

To prepare for the first "Programming in Black and Blue," I posted a request for stories on three different networks. Judging by the responses, it seems there is a pent-up urge for programmers to purge themselves of nightmarish experiences. Some of the stories I received make for "believe-it-or-not"-style headlines:

"Project Leader Requires Programmers to Write More Bugs!"
"Designers Forbidden to Meet With Users!"
"Programmer Kills Bug—Client Orders it Revived!"

Some of these stories will be told in future installments. This month we will focus on problems related to software teams and group communication.

Absence of Group Memory Results in Free Telephone Installations

It seems a regional telephone company had established a modern data processing facility in one part of their service area, and was preparing to establish a copy of this facility to service customers in a second area. The basic plan was to make a copy of all the source code for the billing programs, databases, and procedures and build an entirely new system. As you can imagine, this is a large job, rife with small details.

The job was done by a team that involved people from many departments at many levels—from upper management to operations technicians. With hindsight, the project made several big mistakes in group communication:

- No group memory. Planning and coordination was by word of mouth.
- Lack of communication at a high level. The manager coordinating the new installation never consulted key MIS managers about his plans. Is this symptomatic of a strong hierarchical organization?
- Lack of communication at a low level. A technician found an error while testing the new installation. There was supposed to be a program to correspond with each transaction in the system. One was found to be missing. Rather than track down the missing program, the technician patched around the problem and did not report it to anyone.

Despite all this, the second facility went into operation, and customers were happy. Too happy, as it turned out. One program was not installed, and no one noticed its absence for several months. It was the program that computed installation charges when a customer changed services. The customers did not complain when they were not billed, so things went on for some time before a discrepancy was noticed. Finally someone compared monthly financial performance with previous months and the problem was corrected.

It is tempting to blame the technician, who had a clear chance to identify and fix the problem. But the person who told me this story placed the blame on the (lack of) group organization and communication. With a written plan and clear communications, the missing subsystem would probably have been successfully copied.

team." This is definitely a role that should be rotated, to preserve both the sharp edge and the social health of the critic.

Group Memory

Software development groups are very good at producing piles of paper. With planning documents, specifications, analysis and design documents, code, and correspondence we keep our printer busily spewing out warm pieces of paper. As a collection, these documents represent the memory of the group, and are very precious indeed. But how do you find anything in that pile of collected knowledge? Constantine suggests the group appoint an Information Manager, whose job is to "keep group memory a visible and accessible record."

We have been careful to listen to this wisdom, and have taken great pains to make group memory accessible to everyone who needs it. The result is a combination of a filing system, a Hypercard index, and a simple procedure, collectively known as the document archive. It took some time to implement, but it now takes relatively little time to maintain and is worth its weight in Elvis memorabilia.

The archive is divided into categories: general descriptions, applications, project management, meeting minutes, and several categories specific to our project. All documents that come out of our development process (as well as supporting documents such as journal articles) are given to the archive's curator, who assigns them a number, enters them in the Hypercard index, and files the paper copy. We plan to assemble an archive of the electronic versions of all the documents as well. When a document is revised, the revision is archived in the same way and filed with the previous version.

This has given the group a very important resource. Looking up design notes or the minutes of a discussion is usually as simple as looking in the printed version of the archive index. For harder searches, the Hypercard index supports searching, and holds abstracts and revision notes on all the documents. As major phases of the project draw to a close, we can check through the documents produced along the way to make sure that we covered all the issues raised in meetings. There is finally a way to answer to the age-old question, "I know we talked about that six months ago, but what did we decide?"

Constantine suggests a few very useful-sounding sections for the archive that we have not implemented (which we probably should). The "process record" is a condensed, running log of the discussion and decisions of the group. The "product record" contains the requirements specification or system design, always describing the current version. And two lists help keep track of loose ends: the "deferred decisions list" holds recognized but unresolved issues, and a "do list" or "bin" contains issues that came up during discussion but were set aside temporarily. These are just the things that are usually forgotten.

So What?

I have been involved with many software projects in many settings. The software this team is building is by far the most reliable I have had anything to do with. The project is still on schedule after six months, and the team members do not feel badgered, hassled, or unappreciated.

Of course, this is not all due to the social structure of the team. The staff is extremely competent to begin with, and we established thorough testing procedures and a careful development process. Any number of factors contribute to a project's success. But team organization underlies everything else. It brings different personalities and interests together, smooths the way for communication, anticipates difficulties, gives us an identity within the rest of the corporation. I am convinced that the structure we have built would withstand difficulty (personality conflicts, for example, or incompetent management) as well as or better than any other team organization I know of.

Take Time to Make Time

One last interesting point. In the first few weeks I was worried that all this was going to take too much time. One thing about a hierarchy—the managers manage and the programmers program. It felt like we were going to be spread too thin, with all these roles distributed among so few people. The burden is increased by our development plan, which involves many review meetings and involvement in one another's code.

We based our project plan on the assumption that on average, programmers would get four hours a day to do programming. The rest of the time would be taken up with other team responsibilities. This did not sound good, considering my previous experience that programmers who were supposed to work eight hours a day actually put in three or four. Given that precedent, would we be working one productive hour a day?

As it turns out, we were about right. I cannot support this scientifically, but my observation is that we actually work more productively than we would if we had responsibility only to code. After four hours of designing or coding your mind is useless, and the diversion of a review or writing meeting minutes is welcome. And after an afternoon of distractions nothing feels better than to concentrate solely on coding again. We wind up doing five to seven hours of real work in an eight-hour day!

For Further Reading

Constantine, L. Teamwork paradigms and the structured open team. In *Proceedings of Software Development '90,* Miller Freeman Publications, San Francisco, 1990. (A good introduction and summary of these ideas).

Doyle, M. and Strauss, D. *How to Make Meetings Work.* Jove, NY: 1976. (Granddaddies of many of these ideas, including facilitator, scribe, group memory.)

Thomsett, R. Effective project teams: A dilemma, a model, a solution. *Amer. Program. 3,* 7/8 July-Aug. 1990. (Discusses structured open ideas in practice, especially personalities and team roles).

Zahniser, R. A. Building software in groups. *Amer. Program. 3,* 7/8 July-Aug. 1990. (Concerned with general productive advantage of group process, e.g., JAD, system storyboarding.)

The social styles test and instructions for administering and interpreting it came from The Tracom Corporation, 200 Fillmore St., Denver, CO 80206. The two principals of Tracom, Roger Reid and David Merrill, have published a book entitled *Personal Styles and Effective Performance* (Chilton, 1981). **C**

---

*The **Practical Programmer** wants to hear your stories. What worked for you, and why? What didn't work, and what were the horrible results? Forthcoming columns will discuss testing, the development process (how's your life cycle?), and prototyping. Send your braggardly tales and autopsy reports to:*

Marc Rettig
Academic Computing
Summer Institute of Linguistics
7500 West Camp Wisdom Road
Dallas, TX 75236
Internet: marc@txsil.lonestar.org.
Compuserve: 76703,1037
The Well: mrettig

***Marc Rettig** is a member of the technical staff at the Summer Institute of Linguistics, and a freelance writer. He is Technical Editor of* Database Programming and Design *magazine, and Consulting Technical Editor of* AI-Expert.