Agile Software Development: The Cooperative Game (2nd Edition)
by Alistair Cockburn (Author)

Chapter 3: "Communicating, Cooperating Teams" [1]

This chapter considers the effect of the physical environment, communication modalities used for jumping the inevitable communication gaps, the role of amicability and conflict, and subcultures on the team. These issues highlight the fact that projects need people to notice important events and to be both willing and able to communicate to others what they notice.

"Convection Currents of Information" compares the movement of information to the dispersion of heat and gas. The comparison yields several useful associations: the energy cost of information transfer, osmotic communication, information radiators, and information drafts.

"Jumping the Communications Gap" examines people's efficiency in conveying ideas using warmer and cooler communication channels. It introduces the idea of adding "stickiness" to information and looks at how those two topics relate to transferring information across time.

"Teams as Communities" discusses amicability and conflict, the role of small team victories in team building, and the sorts of subcultures that evolve on a project. We will see that the differing cultural values are both useful to the organization and difficult for the team to deal with.

"Team Ecologies" considers a software-development team as an ecosystem in which physical structures, roles, and individuals with unique personalities all exert forces on each other. That each project produces its own, unique ecosystem makes the job of methodology design even more difficult.

1. CONVECTION CURRENTS of INFORMATION

Saying that software development is a cooperative game of communication implies that a project's rate of progress is linked to how long it takes information to get from one person's mind to another's.. If Kim knows something that Pat needs, the project's progress depends on

- How long it takes Pat to discover that Kim knows something useful
- How much energy it costs Pat and Kim together to get the knowledge transferred to Pat

Let's see how much this costs a project.

Delays and lost opportunity costs

A programmer these days costs a company about $2.10 per minute, so that adding one minute to getting a question answered adds $2.10 to the cost of the project. Standing up and walking to another table can add that minute.

Suppose that people who program in pairs ask and get answers to 100 questions per week. Adding that minute's delay costs the project $210 per programmer per week. On a 12-person team, this is about $2,500 per week for the team, which adds up to $50,000 for a 20-week project.

The project gets delayed almost a full week and costs an extra $50,000 *for each minute of delay* in getting questions answered, not assuming any other damage to the project for the questions taking longer to answer!

The delay is more on the order of five minutes if a person has to walk down the hall. If Kim is not there, it is likely that when Pat returns to his office, he has lost the train of thought he was working on and has to spend more time and energy recovering it.

[1] **http://alistair.cockburn.us/ASD+book+extract%3a+%22Communicating%2c+cooperating+teams%22**

Even worse, the next time Pat has a question, he may decide against walking upstairs, because Kim might not be there. For not asking the question, he makes an assumption. Some percentage of his assumptions will be wrong, and each wrong assumption results in Pat introducing an error into the program. Finding and fixing that error costs the project anything from multiple minutes to multiple days.

Thus, Pat's not asking his question and getting it answered represents a large *lost opportunity cost*. Over the course of the project, the lost opportunity cost is far greater than the cost of walking upstairs.
I hope you palpably feel the project's development costs rising in the following six situations:

1. Kim and Pat pair-program on the same workstation (Figure 3-1). Pat wonders a question out loud, and Kim answers. Or, Kim mentions the answer in passing as part of their ongoing conversation, and Pat recognizes it as useful information. This takes little work by each person and takes the least time to accomplish.

Figure 3-1.

2. Kim and Pat at separate workstations, but right next to each other (side-by-side programming). Using peripheral vision or the usual chit-chat that develops when sitting close together, Kim notices that Pat is looking for something on the Web and asks what the question is. Or, Pat simply asks. Kim answers, possibly without looking away from the screen. Not much work; not much time involved.

3. Kim and Pat work on opposite sides of a room, facing away from each other (Figure 3-2). Kim is not likely to notice that Pat is looking for something, but Pat can easily see whether Kim is available to answer a question. At that point, Pat asks and Kim answers.

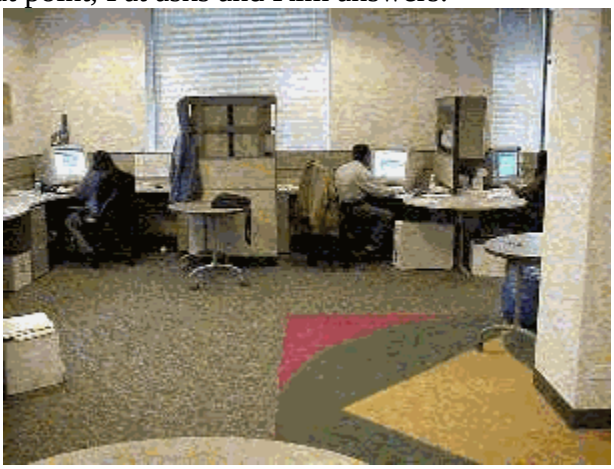


Figure 3-2.

4. Kim and Pat sit in adjacent offices, separated by a wall. Kim can't see when Pat is looking for something, and Pat can't see if Kim is available. Pat must get up, peek around the doorframe to see if Kim is in, and then ask Kim the question.

5. Kim and Pat sit on different floors or in adjacent buildings. Pat walks upstairs only to find that Kim is out! Now Pat has lost time, energy, the train of thought he was holding while he was working

downstairs, and the motivation to walk upstairs the next time he has a question. The lost opportunity cost starts to mount.

6. Kim and Pat sit in different cities, possibly with several time zones between them. In this setting, not only will they not ask each other questions as often, they also will have to use less efficient, less rich communication channels to discuss the question and its answer. They expend more energy, over a longer period of time, to achieve the same communication result.

The main question is, if you were funding this project, which working configuration would you like Kim and Pat to use?

What we see is that even minor differences have an impact on the rate of information flow.

Figure 3-3.

Pair programming and working across a partition. Between which pair of people will information discovery happen fastest?

Notice, in Figure 3-3, the two different situations occurring at the same time. The two people on the left are pair programming. It may be nice for them to have a small separation from the person on the right. However, if it happened to be the two people across the partition who needed to work together, the partition would soon become a problem. Indeed, I visited two people who were working across a partition, and it wasn't long before they removed the partition. As one of them explained, "I couldn't see his eyes."

Erg-Seconds

Comparing the flow of information with that of heat and gas is not as far-fetched as it may at first seem. With every speech act, Kim radiates both information and energy into the environment around her. That information or energy gets picked up by people within sight or hearing. Pat also radiates, with every speech act.

In his case he radiates his need for information. Sooner or later, either Kim detects Pat's information need, or Pat detects that Kim has the information. Whichever way the discovery goes, they then engage in conversation (or Pat reads Kim's document, if Kim's information is in written form).

In gas-dispersion problems, one analyzes the distance that molecules travel in a certain amount of time. The unit of measure for molecules is *moles* and that for distance is *meters;* therefore, gas dispersion is measured in *mole-meters / second* (how many moles of the gas travel how far, in how much time).

We can analyze the movement of ideas— *memes*, to borrow an appropriate term from *The Selfish Gene* (Dawkins 1990)—using similar terms. We are interested in how many useful memes flow through the project team each minute.

A meter is not the correct unit, though, because ideas travel through phone lines, e-mail notes, and documents, rather than through space.

What we care about is the amount of energy it takes to move a meme from one mind to another. The appropriate units are *erg-seconds*. An *erg* is a unit of work (such as walking up the stairs), and *a second* is a unit of time (such as time spent on the telephone); therefore, the term *erg-seconds* captures the cost in both labor and time to get a question answered.

(Bo Leuf comments that its inverse is also useful: *argh-seconds*, a measure of the pain of expending energy and *not* managing to convey the idea.)

Using this metaphor, let's look at office layouts to see the energy cost associated with detecting that someone else has some needed information.

Suppose that Kim and Pat sit in offices some distance from each other (Figure 3-4). The walls between them keep Pat from seeing or hearing Kim. Kim radiates information as she walks around on her daily travels. The people in her room detect the greatest amount of information, and the people in earshot of her movement detect the next greatest amount. Information reaches Pat either as Kim walks into his office, or indirectly, through other people.

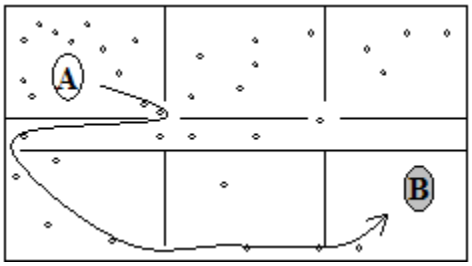


Figure 3-4. Energy and information moving through a barrier complex.

If their offices are next to each other, Kim is more likely to pop into Pat's office, or vice versa (Figure 3-5, top). Just as gas molecules or convected heat move more easily between neighboring rooms, so also does project information.

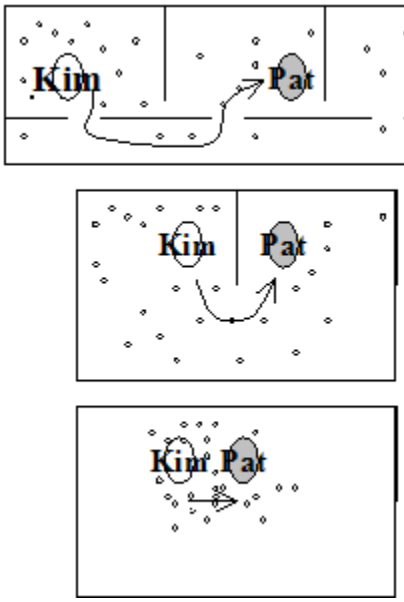


Figure 3-5. Gas canisters (or people) in three different configurations.

If Kim and Pat share an office (Figure 3-5, middle), then just as Pat will smell Kim's perfume sooner than anyone outside the office will, so will he notice if Kim radiates information that is useful to him.

The greatest rate of information movement occurs if they are sitting side by side. In the case of information, the information transmission is greater if they are working on the same task, pair programming, than if they are merely sitting side by side, working on different tasks (this has more to do with their focus of attention than the radiation).

Describing information transmission costs in *erg-seconds* captures the effect of distance and communication modality on project costs.

Assume face-to-face communications, sitting in your own office, versus walking 50 meters to a colleague's office. Walking down the hall takes work (ergs) and time (seconds). Energy and cost go up, and the information transfer rate goes down. Move people closer, to the office next door. As the distance decreases, work required to visit the colleague decreases and so do energy and project cost while the information transfer rate increases.

Similarly, describing an idea on the phone takes more time than describing it in person. In this case, the time factor increases, and so does cost to the project.

The *erg-seconds* formula accounts for these changes well.

Of course, the formula does not account for wasted energy, such as jumping up and down while talking on the phone or walking around the building the long way in getting to a colleague's office. It also does not guarantee that two people who work in the same room will ever actually understand each other (see "The Impossibility of Communication"). What it does say is that project costs increase in proportion to the time it takes for people to understand each other

Osmotic Communication

While writing, reading, typing, or talking, we pick up traces of the ongoing sounds around us, using some background listening mode even though we are not consciously paying attention.

If someone says something interesting, we may perk up and join the conversation. Otherwise, the sound goes through some background processing, either just above or just below our conscious level.

In some cases, we register enough about the conversation to be able to develop what we need directly from memory. Otherwise, we may recall a phrase that was used or perhaps only *that* a particular person was discussing a particular topic. In any case, we register enough to ask about it.

This taking in of information without directly paying attention to it is like the process of osmosis, in which one substance seeps from one system, through a separator, into another.

Osmotic communication further lowers the cost of idea transfer.

If Pat and Kim work in the same room, with Pat programming and Kim having a discussion, Pat may get just enough information to know that Kim has talked about the idea. If multiple people are working in the same room, then Pat knows that someone in the room has the answer.

We have seen three separate effects that office layout has on communication costs within a project:

1. The lost opportunity cost of not asking questions
2. The overall cost of detecting and transferring information (erg-seconds)
3. The reduction in cost when people discover information in background sounds (osmotic communication)

The three magnify the effects of distance in office seating. People who sit close by each other benefit in all three effects; people who sit in separate locations suffer in all three.

According to this theory, sponsors should think twice before sponsoring a geographically distributed project. One might think that we now have an easy answer to the riddle of how to seat people: "Obviously, put them into open and shared workspaces." Unfortunately, people are not so uniform or simple that this will work in all cases.

Three more issues affect the answer in any one particular setting:

1. The sort of information being shared
2. People's personal preferences
3. Drafts

The sort of information shared: Example; team members exchange both business and technical information.

Suppose that Chris is the business expert in the group. If Chris, Pat, and Kim sit together, Chris can answer business questions as soon as Pat or Kim encounters them. Chris might even see what Pat and Kim are doing and guide them in a different direction. The three of them can put their heads together at any instant to jointly invent something better than any one of them could do.

This sort of *radical colocation* (as it has recently been called) only works for very small teams. Among twelve programmers and four business experts, who should sit close to whom? How does one arrange seating with two-person rooms?

The most common seating arrangement I encounter consists of programmers sitting on one side of the building and business experts on the other.

This seating arrangement produces two problems. The obvious one is the cost of business communication, including the lost opportunity cost of missed early interventions.

Not so obvious is that each group forms its own community and usually complains about the other group. The chit-chat in the osmotic communication is filled with these complaints, interfering with the ability of people in each group to work with each other in an amicable way.

As is natural with osmotic communication, this emotionally loaded background noise soaks into each group's subconscious. In this case, it does not educate them but rather attacks their attitude. Going into a meeting with "those idiotic other people," they don't give full consideration to what the other people say and don't offer full information when speaking. The group's amicability suffers, with all the attendant costs just discussed.

My current preference is to find seating arrangements where one or more business experts sit close to two or more programmers. Where this is not possible, I look for other business and social mechanisms that will get the business expert in regular, meaningful collaboration with the programmers on a frequent (preferably daily) basis.

Cross-specialty teams that work together have been recommended by many authors. These teams have been given names such as *Holistic Diversity* (Cockburn 1998), CASE teams (Hammer 1994), and Feature teams (McCarthy 1995). When this can be done, the project as a whole moves faster, based on the increase in both information flow and amicability across specialties.

The *second* of the three remaining issues is the matter of people's *personal preferences*.

As I started asking people about working in shared rooms versus in private offices, several issues emerged.

Some people really value their quiet, private offices. They value them enough that they would feel offended if they had to give them up, some even to the point that they would quit the company. If that is the case, then any gain in communication is partially lost if the person stays, but feels offended, and is completely lost if the person leaves the company.

Thus, the clear theoretical argument for seating people close to the people they need to interact with is affected by personal preferences. Several people have told me, "I prefer having my own office, but considering all the projects I've been on, I would have to say that I was never so productive as when I shared an office with my project mate." I have moved out of private offices so often that I eventually noticed it as a pattern. As I noticed other experts doing it, it became a project-management strategy, which I call "Expert in Earshot" (Cockburn 2001a).

The *third* issue affecting the question of where to seat people concerns drafts.

Drafts

> Drafty Cubicles
> One day, while I was describing this peculiar notion of convection currents of information flow, one of the listeners suddenly exclaimed, "But you have to watch out for drafts!"
>
> He went on to explain that he had been working in a place where he and the other programmers had low-walled cubicles next to each other and so benefited from overhearing each other.
>
> On the other side of their bank of cubicles sat the call center people, who answered questions on the phone all day. They also benefited from overhearing each other. But, and here was the bad part, the conversation of the call center people would (in his words) "wash over the walls to the programmers' area." There was a "draft" of unwanted information coming from that area.

Drafts are the *unwanted information* in our newly extended metaphor.

Later, two programmers were talking about how their walls were too thin. They enjoyed their shared room but were bothered by their neighbors, who argued loudly with each other. Their room was *drafty*, in an information sense.

We now have a nice pair of forces to balance: We want to set up seating clusters that increase information flow among people sitting within hearing distance, and balance that against draftiness—their overhearing information that is not helpful to them. You can develop a sense of this for this yourself, as you walk around.

Osmosis across Distances

Is there anything that teams can do to improve communication if they do not sit together, for whatever reason?

Charles Herring, in Australia, describes applying technology to simulate "presence and awareness," a term used by a researcher in computer-supported collaborative work (Herring 2000). Following is a paraphrased summary of their experience:

> e-Presence and e-Awareness
> The people sat in different parts of the same building. They had microphone and Web camera on their workstations and arranged small windows on their monitors, showing the picture from the other people's cameras.

They wanted to give each person a sensation that they were sitting in a group ("presence") and an awareness of what the other people were all doing.

Pat could just glance at Kim's image to decide if Kim was in a state to be disturbed with a question. In that glance, he could detect if Kim was typing with great concentration, working in a relaxed mode, talking to someone else, or gone.

Pat could then ask Kim a question, using the microphone or chat boxes they kept on their screens. They could even drop code fragments from their programming workspaces into the chat boxes.

They reported a low distraction rate. Charles added that while programming, he could easily respond to queries and even answer programming problems without losing his main train of thought on his own work.

Pavel Curtis and others at Xerox PARC were able to simulate "whispering" (when a user would like to speak to just one person in a room) through video and audio. They also had their online chat rooms produce background sounds as people entered or left (Curtis 1995).

Because memes (ideas) don't have to travel through air but travel through the senses, primarily audio and visual, we should be able to mimic the effects of convection currents of information using high-bandwidth technology. Still missing from that technology, of course, are the tactile and kinesthetic cues that can be so important to interpersonal communication.

Information Radiators

An information radiator displays information in a place where passersby can see it. With information radiators, the passersby don't need to ask questions; the information simply hits them as they pass.

Figure 3-6. Hall with information radiators.

Two characteristics are key to a good information radiator. The first is that the information changes over time. This makes it worth a person's while to look at the display. This characteristic explains why a status display makes for a useful information radiator and a display of the company's development process does not.

The other characteristic is that it takes very little energy to view the display. Size matters when it comes to information radiators—the bigger the better.

Hallways qualify very nicely as good places for information radiators. Web pages don't. Accessing the Web page costs most people more effort than they are willing to expend, and so the information stays hidden. The following story contributed by Martin Fowler, at *Thoughtworks*, reports an exception: This team found that a particular report worked best on a Web page.

Automated Build Report
A program auto-builds the team's system every 15 minutes. After each build, it sends e-mail messages to each person whose test cases failed and posts the build statistics to a Web page.

The information about the system is updated every 15 minutes on the Web page. Martin reports that a growing number of programmers keep that Web page up on their screen at all times and periodically just hit the Refresh button to check the recent system build history.

Figure 3-7. Status display showing completion level and quality of user stories being implemented.

The first information radiators I noticed were at *Thoughtworks*, while talking with Martin Fowler about *Thoughtworks'* application of *XP* to an unusually large (40-person) project (Figure 3-6 and Figure 3-7).

> **Progress Radiators**
> Martin was describing that the testing group had been worried about the state of the system.
>
> To assuage the testers' concerns, the programmers placed this poster in the hallway (Figure 3-6) to show their progress.
>
> The chart shows the state of the user stories being worked on in the iteration, with one Post-It note per story. The programmers moved the notes on the graph to show both the completeness and the implementation quality of the user stories they were working on. They moved a note to the right as a story grew to completion and raised it higher on the poster as its quality improved. A note might stop moving to the right for a time while it moved up.
>
> The testers could see the state of the system without pestering the programmers. In this case, they saw that the work was farther along than they thought and soon became less worried about the state of the project.
>
> The best thing was that they could see the progress of the work daily, without asking the programmers a question.

Just as a heating duct blows air into a hallway or a heater radiates heat into a room, these posters radiate information into the hallway, onto people walking by. They are marvelous for passing along information quietly, with little effort, and without disturbing the people whose status is being reported.

A second use of information radiators, suited for any project using increments of a month or less, is to show the work breakdown and assignments for the next increment (Figure 3-8). The following example also comes from *Thoughtworks*.

Displaying Work Breakdown
The team created a flipchart for each user story. They put sticky notes on the flipchart for the tasks they would need to do for that story.

They would move notes '*below'* a flipchart to show tasks being taken out of scope of the current iteration in order to meet the delivery schedule.

Figure 3-8. Large information radiator wall showing the iteration plan, one flipchart per user story.

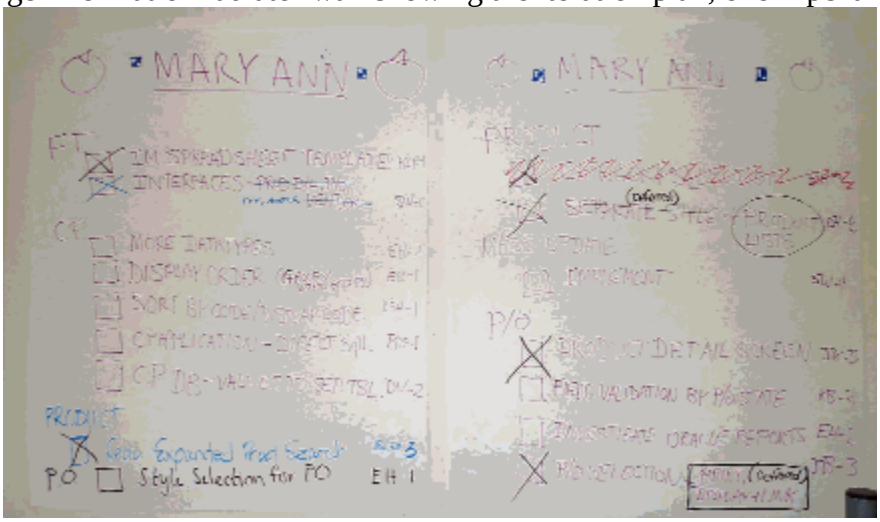


Figure 3-9. Detail of an XP task signup and status for one iteration (nicknamed "Mary Ann").

Evant's *XP* team also used whiteboards and flipcharts as information radiators. Figure 3-9 shows the tasks for iteration "Mary Ann" (each iteration was nicknamed for someone on the Gilligan TV series).

A third use of flipcharts as information radiators is to show the results of the project's periodic *reflection workshop* (Figure 3-10). During these one- to two-hour workshops, the team discusses what is going well for them and what they should do differently for the next period. They write those on a flipchart and post it in a prominent place so that people are reminded about these thoughts as they work.

The wording in the posters matters. One *XP* team had posted "Things we did wrong last increment." Another had posted, "Things to work on this increment." Imagine the difference in the projects: The first one radiated

guilt into the project room and was, not surprisingly, not referred to very much by the project team. The second one radiates promise. The people on the second team referred to their poster quite frequently when talking about their project.

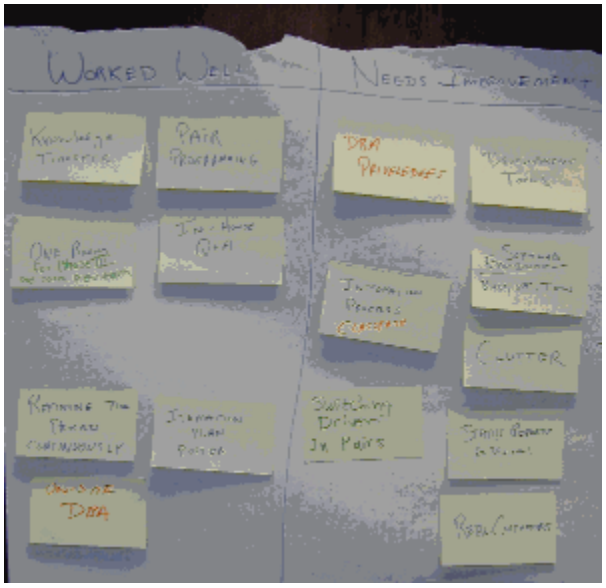


Figure 3-10. Reflection workshop output.

Periodic reflection workshops such as these are used in *Crystal Clear* and *XP* projects.

A fourth use of information radiators is to show everyone the user stories delivered or in progress, the number of acceptance tests written and met, and so on. (Figure 3-11).

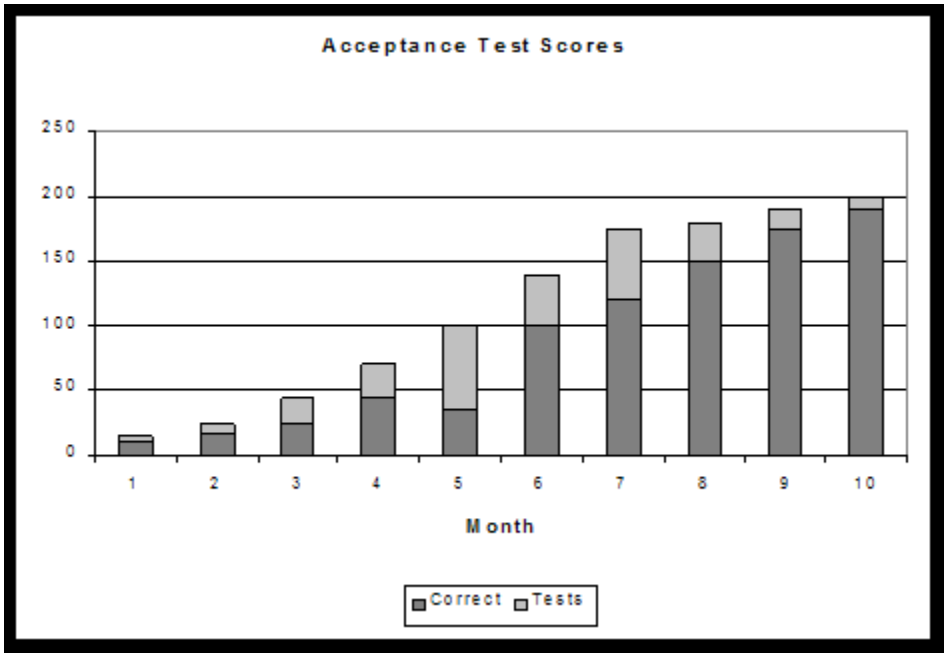


Figure 3-11. Graph showing growing completion.

The systems operations team at *eBucks.com* constructed a fifth use of information radiators, this time to keep the programmers from pestering them.

Displaying System Status

The programmers kept asking, “Is system A up? Is system B up? Is the link to the back end up?”

The maintenance team wrote the status of each system and link on the whiteboard outside their area. Each day, they updated the status. It looked rather like a ski area posting the status of lifts and runs (so skiers don't keep asking the ski resort staff).

The group at *eBucks.com* came up with a sixth use of information radiators. This time it was the programmers who created the status displays:

Displaying Work Progress
The programmers were being asked about the status of their work every hour or two, which caused them no end of frustration.

They wrote on the whiteboard outside their office their intentions for the current week. As they completed their tasks, carefully sized to be of the half-day to two-day variety, they marked the tasks complete.

After these boards had been tried by the programmers, several other groups started using them to broadcast their own priorities and progress.

Applying the theory of hot air
People have long applied the above-described "hot air theory of software development."

Gerald Weinberg discussed the damaging effect of removing a soda machine from a computer help-desk area (Weinberg 1998). Thomas Allen, of MIT's Sloan School of Management, discussed the effect of building design on R&D organizations in (Allen 19??). IBM and Hewlett Packard have incorporated such research in their R&D buildings since the late 1970s.

As a result of these and others' work, it seems natural that research and development groups have whiteboards in the hallways or near coffee machines. What we have forgotten, though, is the significance of actually being within sight and earshot of each other.

Here are several examples. The first is from a *Crystal Orange* project. The second is from a project unsuccessfully trying to apply *Crystal Clear*. Next comes a discussion of the *Caves and Commons* room design recommended by *XP*. The final example is a success story from Lockheed's Skunk Works group.

Repairing Design Discussions
On project "Winifred" (Cockburn 1998), the lead programmer announced at regular intervals that design was unnecessary and that code simply grew under his fingertips.

As a predictable result, the young programmers working in the room with him also felt it unnecessary to design. The code looked that way, too.

He eventually left and I took his place. To reverse the situation, I arranged for us to design by having conversations at the whiteboard. After some period of doing this, I started getting questions like, "Could you look at the responsibilities (or communication patterns) of these objects?"

By setting an audible tone in the room and making these design discussions legitimate and valued, the programmers started to converse about design together.

Colocation is considered a critical element in *Crystal Clear*, a light methodology for small teams (see Chapter 6). A rule of *Crystal Clear* is that the entire team must sit in the same or adjacent rooms, in order to take advantage of convection currents of information and osmotic communications.

Crystal un-Clear
"Pat" asked me to visit his *Crystal Clear* project. When I arrived, he wasn't at his desk. The secretary said he was with his teammate.
I offered to go to that office, but she said, "You can't. There is a combination lock in the hallway over to that section."
"!! …?"

Each time a team member wanted to ask a question, he had to stand, walk across the hall, punch in the lock combination, and walk to the teammate's office. Clearly, this team was not getting the benefit of osmotic communication or the low cost of information transfer. Fortunately, changing the team seating was a simple matter to arrange.

Caves and Commons
The Caves and Commons room arrangement recommended in *XP* makes use of all three information- exchange mechanisms. It is shown in action in Figure 3-12 and diagrammed in Figure 3-13.

Figure 3-12. The *RoleModel* Software team at work.

Caves and Commons is very effective, but as Tom DeMarco correctly warns, it can easily be abused to become just a programming sweatshop. Therefore, not only the room layout is described in this section but also the social *presuppositions* that accompany its use: a single project team, good team dynamics, and provision for both private and project space.

The phrase *Caves and Commons* refers to the creation of two zones in the room. The Commons area is organized to maximize osmotic communication and information transfer. For this to make sense, the people in the room must be working on the same project. It is perfect for *XP*'s single team of up to 12 people programming in pairs (Figure 3-12).

The Caves portion of the room is organized to give people a private place to do e-mail, make phone calls, and take care of their need for separation. In *RoleModel* Software's office, private workstations are set up along one wall (Figure 3-12). At *Evant*, a table came out from the walls on two sides of the room.

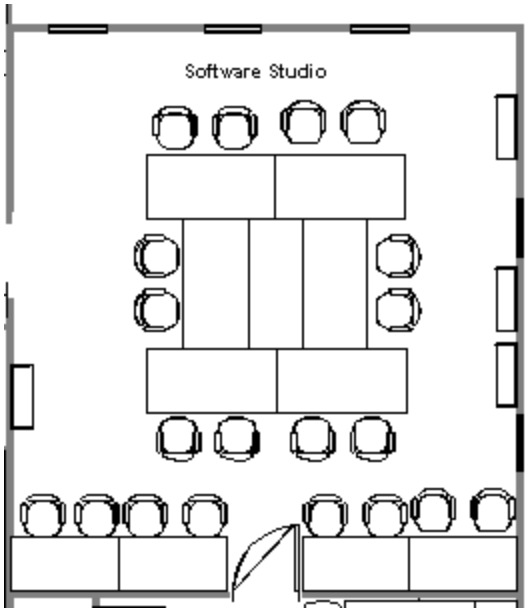


Figure 3-13. The "caves and common" room layout used at RoleModel Software.

People who have worked in *Caves and Commons*

facilities say that there needs to be ample wall space for whiteboards and posted flipcharts, and two more types of rooms for the team to use: a food-preparation room and areas for small discussions to take place.

You can see from the picture that while the caves and commons room is very efficient for transmitting information, it is also very efficient for transmitting coughs and colds. People who work in this sort of room encourage their colleagues to stay home if they don't feel well and to return after they have recovered.

You can also see that it is drafty (in an information sense): The people sitting in this configuration should really need to overhear each other.

Finally, you can see that it is very effective as long as the morale of the group is good. If the social chit-chat degenerates into negative chatter, the highly osmotic communication again magnifies its effect.

Skunk Works

It is useful to compare the above discussions against a group performing classical "engineering," one of the most effective aero-engineering groups: Lockheed's "skunk works" team. This team achieved fame for its rapid development of a series of radical new airplane designs in the second half of the 20th century, under the guidance of Jim Kelly and his successor, Ben Rich. Ben Rich wrote about their experiences in the book *Skunk Works* (Rich 1994).

Rich highlights that, among the rules of the group, Kelly insisted on people taking accountability for decisions from design through testing, and on their sitting close together. The following quotation is from that book:

> **Skunk Works** Rooms
> "Kelly kept those of us working on his airplane jammed together in one corner of our [building]... My three-man thermodynamics and propulsion group now shared space with the performance and stability-control people. Through a connecting door was the eight-man structures group. ... Henry and I could have reached through the doorway and shaken hands.

"... I was separated by a connecting doorway from the office of four structures guys, who configured the strength, loads, and weight of the airplane from preliminary design sketches. ... [t]he aerodynamics group in my office began talking through the open door to the structures bunch about calculations on the center of pressures on the fuselage, when suddenly I got the idea of unhinging the door between us, laying the door between a couple of desks, tacking onto it a long sheet of paper, and having all of us join in designing the optimum final design. ... It took us a day and a half. ..."

"All that mattered to him was our proximity to the production floor: A stone's throw was too far away; he wanted us only steps away from the shop workers, to make quick structural or parts changes or answer any of their questions."

Every project team should be on a quest to reduce the total energy cost of detecting and transferring needed ideas. That means noticing and improving the convection currents of information flow, getting the benefits of osmotic communication, watching for sources of drafts, and using information radiators. The end goal is to lower the *erg-seconds* required for team members to exchange information, whatever constraints their organization places on their seating, and with or without technology.

2. JUMPING COMMUNICATION GAPS

To make communications as effective as possible, it is essential to improve the likelihood that the receiver can jump the communication gaps that are always present. The sender needs to touch into the highest level of shared experience with the receiver. The two people should provide constant feedback to each other in this process so that they can detect the extent to which they miss their intention.

Modalities in Communication

Imagine a simple discussion at the whiteboard. How many communication mechanisms are at play? Consider these twelve:

1. **Physical proximity**. Standing about one meter from each other, the people detect minute visual cues, tiny movements of eye muscles to overall muscle tension.

   The speaker may move closer to indicate aggressiveness or enthusiasm. The listener may move closer to indicate interest, agreement, or the desire to speak; or, the listener may move away to indicate fear, disagreement, or the need to think privately for a moment. The speaker and listener manipulate their relative distance to express various emotions and stages of agreement, disagreement, aggressiveness, trust, and distrust.

   The signals vary across cultures and personalities, but the signals are both present and used.

2. **Three-dimensionality**. The people notice visual parallax, or 3D information.

   The parallax shift of the visual image is lost when the same people talk over a video link, even if they are similarly close to the camera and screen.

3. **Smell.** Smell is one of those senses that is unimportant to some people, very important to others, and important but subconscious to many. One person reported that she can often sense sublimated fear and distress, probably through sense of smell. It certainly is the case that those cues are available at the whiteboard and are lost in remote communications.

4. **Kinesthetics**. Many people use kinesthetics (sensation of movement) to help them think and remember. The speaker might use it to help construct a new explanation or to help improve the building of a question.

5. **Touch**. One person touches another on the shoulder to mean, "Don't feel threatened by this discussion" or perhaps, "This is really important" or "I have something to say." Touching is part of the overall manipulation of proximity and personal space. In some cases there are objects to touch whose feel is important to the conversation.

6. **Sound**. In the simple use of language, a speaker emphasizes points with colorful adjectives, exaggerations, metaphors, and the like. Besides that simple use of language, the speaker uses pitch, volume, and pacing to differentiate and emphasize ideas in a sentence.

7. **Visuals**. People communicate through gestures as well as words, often making a point by gesturing, raising an eyebrow, or pointing while speaking.

   The people may wave their hands to make shapes in the air or to accentuate the speaking. They may raise an eyebrow to indicate questioning or emphasis. Again, they use pacing to differentiate and emphasize ideas, for example, moving rapidly over obvious parts of a drawing and slowing down or pausing for effect at less obvious or more important parts.

   A person also draws on the whiteboard to present (particularly spatially oriented) information for the other to consider. The drawings may be standardized notations, such as class or timing diagrams. They may be loose sketches. They may even be wiggles having no particular meaning, whose sole purpose is to anchor in a public, static location the thought being discussed for later reference.

8. **Cross-modality timing**. One of the most important characteristics of two people at the whiteboard is the timed correlation of all of the above. The speaker moves facial muscles and gestures while talking, draws while talking and moving, pauses in speech for effect while drawing, and carefully announces key phrases in time, while drawing lines between shapes.

   Cross-modality emphasis helps anchor ideas in the listener's mind, enhancing the memory associations around the idea. Drawing otherwise meaningless wiggles on the board while talking gives meaning to the wiggles that the two can later refer to.

9. **Low latency**. Because the two are standing next to each other, watching and listening to each other, the round-trip time for a signal and a response is very small. This allows real-time question and answer, and interruptions:

   Real-time question-and-answer. The receiver asks questions to reveal ambiguity and missed communication in the speaker's explanation. The timing of the questions sets up a pattern of communication between the people.

   Interruptions. With the very fast round-trip times available in face-to-face communication, the listener can interrupt the speaker, asking for clarification on the spot. During the course of conversation, the speaker may be able to tune the presentation to fit the receiver's background.

   The listener can give the speaker feedback *in the middle of the expression of an idea*, perhaps through a raised eyebrow or other nonverbal modality. The speaker can then adjust the expression on the fly.

10. **Trust and learning**. Through modalities and rapid feedback, the two are likely to develop a sense of comfort and trust in communication with each other. This is comfort and trust of the form, "Oh, when he speaks in that tone of voice he is not actually angry, but just excited." The two find ways to not hurt each other in communication and to know that they will not be hurt in the communication.

They build small emotional normalizing rituals of movement and expression to indicate things like, "I'm starting to feel attacked here" and "You don't need to, because this is not an attack on you." Those rituals serve the people well over the course of the project, particularly when they can't see each other during the communication. At that juncture, touching into the shared experience of these rituals becomes crucial.

You see an example of the need for these normalizing rituals in the amount of airplane travel going on:

> Flying to Places to Be There A senior executive of a video-communications firm returned to San Jose from London. It was her second trip in ten days, each being for a single meeting.
>
> The astonishment for us around was that she obviously had access to state-of-the-art video-conferencing facilities and yet felt that she could not conduct her business over the video link. Her meetings still required the lowest latency, richest, multi-modal communication possible: "in person."
>
> We decided that it is easy to start negotiations over the phone or Internet but difficult to bring them to conclusion that way.

11. **Use of a shared, persistent information radiator**. The whiteboard holds the drawn information in place, while words dissolve in the air. The people can all see the board, draw on the board, and refer to the board just minutes later in the conversation.

The impact of removing modalities
What happens when you remove some of those mechanisms and go to other communication settings?

Remove only physical proximity. With people at opposite ends of a video link, the visual and temporal characteristics should be very much the same as being in person. Somehow, though, they aren't, as witnessed by the video-communications executive who still flew to London for single meetings.

My teammates in Lillehammer, and I, in Oslo, often found that we only made design progress when we took the train trip together. Even walking to the train station together was a more effective design environment for us than talking over our video link.

Remove the visuals (use a telephone). Removing visuals also removes cross-modality timing. You lose the drawings, the gestures, the facial expressions, sight of the muscle tone, proximity cues, and the ability to link speech with action.

Remove voice (use e-mail). With this, you lose vocal inflection, the ability to pause for effect, to check for interruptions, to speed up or slow down to make a point, to raise your tone or volume to indicate surprise, boredom, or the obviousness of the transmitted idea.

Remove the ability to ask questions (but possibly reinstate one of the above modalities). Without the questions, the sender must guess what the receiver knows, doesn't know, would like to ask, and what an appropriate answer to the guessed question might be—all without feedback. Now, the sender really doesn't know what the receiver needs to hear, where the communication gaps are too wide, or where the shared experience lies. (This, of course, applies to me, communicating with you. How many words—which words—do I need to spend on this idea?)

Finally, remove almost everything. Remove visuals, sound, timing, kinesthetics, cross-modality timing, question-and-answer, and you get … paper.

How surprising it is in retrospect that most projects require documentation in the least effective communication format possible! The person who is trying to communicate a design idea must guess at what will work for the reader, does not get to use timing, vocal, or gestural inflections, and gets no feedback along the way.

With this view in mind, it is not surprising that the busiest and best project team leaders say:

- "Put all the people into one room."
- "Don't give me more than four people, that's all I can get into one room and talking together."
- "Give me printing whiteboards, and keep all the rest of your drawing tools."
- "Make sure there are whiteboards and coffee corners all over the building."

The above are standard recommendations among successful project leaders, who count on using the highest communication mode: people, communicating face to face. The discussion of communication modalities matches the findings of researchers, such as McCarthy and Monk (1994).

Making Use of Modalities

The graph in Figure 3-14 serves to capture the above discussion visually. In the graph you see two sets of situations: those in which question and answer are available and those in which they are not.

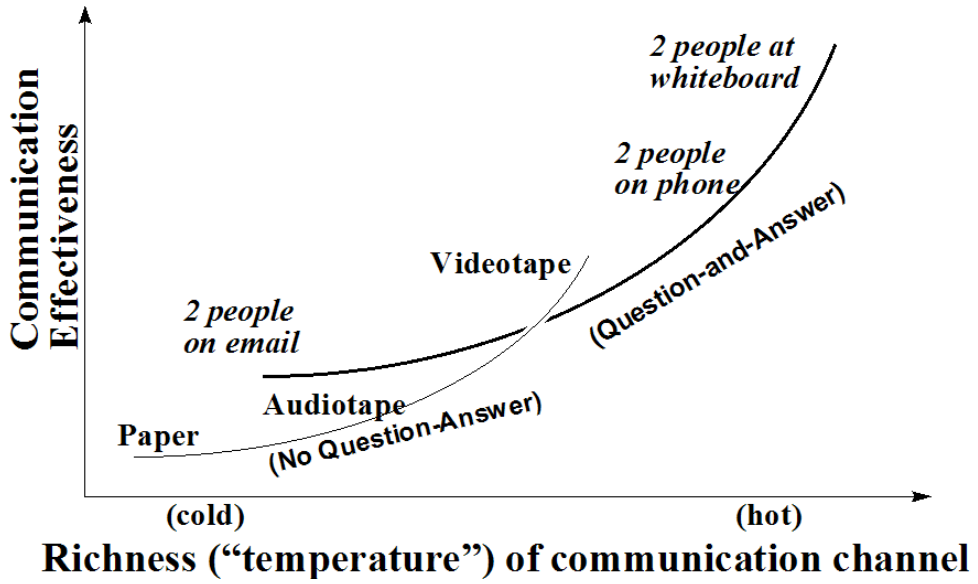


Figure 3-14. Effectiveness of different modes of communication.

The horizontal axis indicates the "temperature" of the communication channel. Warmer indicates that more emotional and informational richness gets conveyed. E-mail is cooler than audio or videotape, and two people communicating face to face is the hottest channel.

What we see in the graph is communication effectiveness rising with the richness (temperature) of the communications channel. Two people at the whiteboard are using the richest form.

The graph provides an idea about how to improve the effectiveness of archival documentation:

> **Videotaped Archival Documentation:**
> Have the designer give a short, 5– to 15-minute description of the design to one or two colleagues who are not familiar with the work. These one or two will act as ombudsmen for the viewers of the

videotape. While the designer leads the discussion, the colleagues interrupt and ask questions as they need to.
Videotape the discussion.

At the end, capture and print the examples and drawings used in the discussion, to act as mnemonic anchors of the discussion.

You might consider posting the talk online, where others can access it using hyperlinked media.

Lizette Velasquez, of Lucent Technologies, reported that not only had she already used that technique with success, but she added that I had forgotten something important:

It is also important to mark and index places where "something interesting happened."

While much of the discussion proceeds at a relatively slow pace, occasionally a question triggers a flurry of significant discussion, and the viewers will want to refer to those sections.

Several people report that they have videotaped talks on their project, but we are missing experiments telling us about this technique in actual use: how to set up the room, how long the discussion can be, what sort of person should be used for the ombudsman. Most of all, I am still waiting for someone to perform this experiment and then, six months later, reflect on whether this was a good idea and what would make it better.

If you are willing to try out this experiment, please let me know these details: what you did, what happened, and what you thought about it months later.

As a thought experiment about the utility of the graph and the experiment, consider the book *Design Patterns* (Gamma 1993). This book is excellent but difficult. I still have trouble understanding the patterns that I have not yet used. I suppose that others have similar difficulties. Imagine that instead of trying to extract the meaning of the patterns from the book, you could see one of the authors explaining the pattern in a video clip. The authors would, of course, rely on tonal inflections, gestures, and timing to get the idea across. I'm sure that most people would understand those difficult patterns much more easily.

The lesson is that we should try to move team communications up the curve as far as possible, for the situation at hand. We should *rely* on informal, face-to-face conversation, not merely tolerate it. Face-to-face communication should become a core part of your development process.

There is a second lesson to pay attention to. Sometimes a cooler communication channel works better, *because* it contains less emotional content.

Cooler Communications Needed
A project leader told me that her team deals better with her when they speak over the phone, because she is too aggressive with her emotions in person.

A married couple told me that they communicated in a more "even" and less emotional level over the phone than in person, just because the face-to-face setting flooded them with visual and emotional cues. Hovenden (1999) describes a meeting in which a senior designer ruined a meeting's original plan by standing up and taking over the whiteboard for the rest of the meeting. In this case, the *lack of anonymity* created a social ranking that interfered with the intended meeting.

Bordia and Prashant (1997) describe that brainstorming improves when social ranking information is hidden from the participants.

> McCarthy and Monk (1994) remind us that e-mail has the advantage of allowing people to reread their own messages before sending them, thereby giving them a chance to clarify the message.

Thus, warmer communications channels are more effective in transferring ideas, but cooler communications channels still have important uses.

Stickiness and Jumping Gaps Across Space

You can see, at this point, how the team of Russian programmers got low cost per idea transferred ("The Russian Programmers" story). Sitting in a room together, they got convection currents of information, osmotic communication, face-to-face communication, and real-time question and answer.

So why did they need to write use cases at all?

The answer is: To give the information some *stickiness*. Information that is recorded on paper has a sort of stickiness—or permanence—that the information in a conversation doesn't, a stickiness you sometimes want.

The person who went to Russia with the use cases wanted to make sure that he did not forget what he was supposed to cover in his conversations. He wanted to make sure that after he explained the use cases to the Russian programmers, they could subsequently read the use cases, understand them, and recall the information without having to ask him again.

The use-case writer, knowing that the use cases were only game markers to remind them of what they already knew or had discussed, could balance the time he spent writing the use cases against the time that would be spent discussing other material. He could decide how much detail should go into the writing.

Figure 3-15. Two people working at a shared, sticky information radiator.

Large, sticky, revisable shared information radiators are often used by people to achieve greater understanding and to align their common goals. Figure 3-15 and Figure 3-16 show a useful mix of whiteboards (static information radiators) and people (dynamic information radiators).

Both whiteboards and paper are particularly good static information radiators and can be written on by all parties, making them *shared, sticky information radiators*.

Until recently, archivability and portability were still problems with whiteboards. If a discussion results in really valuable information being placed on the whiteboard, no one dares erase it and the group can't archive it. This

slows the archiving of valuable information and shuts down the board for the next use. As Ron Jeffries put it, "If you never erase the whiteboards, you might as well write on the walls."

Figure 3-16. Dynamic and static information radiators at work.

A colleague, Mohammad Salim, responded to this situation by covering *all* the walls and hallways with rolls of butcher paper so that people could literally draw on the walls wherever they were. He said, "If you have to take time to walk to a workstation or find a blank whiteboard, you just lost your idea." He continued, saying that when a section of paper gets full, to just roll it up and date it. That way all discussions are archived and can be pulled out for later examination. In his description of finding rolls of paper for later examination, he made use of the fact that humans are good at looking around, as discussed in the last chapter. He also worked hard to reduce the cost of invention and communication while preserving archivability for later discussions.

A number of people report that they are using digital cameras in conjunction with software that cleans up the image ("Whiteboard Photo" at www.pixid.com is one that they refer to). Printing whiteboards continue to be very practical. Often, people start a discussion thinking the outcome will not be significant but see at the end that the whiteboard holds valuable information. With a printing whiteboard, they can simply push the Print button if they wish.

Different information radiators are suited for different sizes of discussion groups, of course. A piece of paper works for two or three people; a whiteboard works for perhaps a dozen.

Recalling these differences will serve us well when we consider methodologies for different projects, in the next chapters.

Sticking Thoughts onto the Wall

On one project, the business analysts were frustrated because their work was growing more and more interdependent. At that time they had no way of holding their thoughts in clear view, and still, while planning their joint work.

We held a discussion about cooperative games, game markers, and stickiness. The people saw that creating a large, persistent and revisable display of their mental territory would help them do their work. One of them immediately posted a picture of the domain on the corridor wall as a starting point.

They worked on it over the weeks, experimenting with representations of their concerns that would allow them to view their mutual interdependence.

There is an interesting and relevant aside to mention about this group, having to do with expectations and citizenship. For reasons I won't go into, this team of business analysts thought they were supposed to work in the *XP* style and that *XP* prohibited them from writing things down.

Notice four things about their situation:

1. They misunderstood *XP*. It does not forbid people to write things down.
2. Their citizenship was so strong that rather than be poor citizens and write down their thoughts on the domain model, they chose to be good citizens and not write down their business model at all!
3. Actually, they knew that the project wouldn't succeed if they really wrote nothing down. So they each clandestinely wrote pseudo- use cases and other notes, which they passed to the programmers. They still did not create a domain model for themselves.
4. By writing down those notes, they subverted their own (mistaken) interpretation of the official process. I find this situation particularly interesting, because they were at war with themselves about whether to be good citizens and follow the process (at the expense of the project) or to be good citizens and protect the project (by violating the process).

What was significant in the end was that they posted an information radiator on the corridor wall, on which they scribbled individually and as a group, to give their thoughts and decisions some stickiness.

Jumping Gaps across Time

Finally, let us look at communicating across time and the twist that lies in store here.

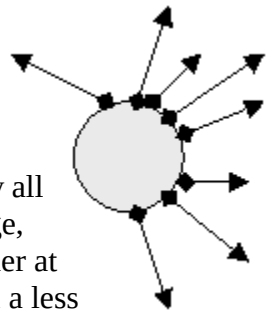
You might expect, after the preceding discussion, that to preserve information across time you would definitely drop reliance on face-to-face communication in favor of paper, audiotape, and videotape.

However, on long-running projects, it turns out to be critically important that the chief architect stay around! This person's contribution is to keep memories of key ideas alive on changing development teams. Once again, *people* are used as the archival medium!

Individual people transfer information effectively across both time and space. As an IBM Fellow put it, while talking about technology transfer, "The way to get effective technology transfer is not to transfer the technology itself but to transfer the *heads* that hold the technology!"

3. TEAMS AS COMMUNITIES

We have looked at what it takes for someone to *notice* something of value on a project and what it takes for someone to *communicate* something of value. It is time to consider whether a person *cares* to notice and communicate anything.



On an effective team, the people pull approximately in the same direction. They actually all pull in slightly different directions, according to their personal goals, personal knowledge, stubbornness, and so on (Figure 3-17). They work together at times and against each other at times. The directions are more closely aligned on a more effective team than they are on a less effective team.

Figure 3-17. An average team working to pull towards a goal on the right.

You can create a large overall effect on the project by eliciting small changes in each person's behavior. This is "micro-touch" intervention: getting people to make changes they don't mind making, in ways that get amplified by the number of people on the project. As each person pulls in a direction closer to the desired and common direction, the changes felt by any one individual are small but the composite effect is large (Figure 3-18).

The small changes come from people being given:

- Additional information about the direction in which they should pull
- Additional information about the effects of their actions so that they notice which actions pull in a different direction
- A better reason to pull in the desired direction

The result is that people start contributing to each other's work as opposed to ignoring or accidentally working against each other.

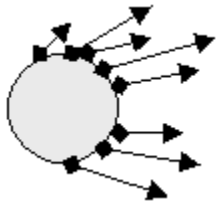


Figure 3-18. A slightly better aligned team.

With small changes like these, people see greater project output for similar amounts of energy and without having to learn major techniques or philosophies. As they notice this, they develop greater pride in their work and more trust in each other. Usually, morale improves, and the project team becomes a better community in which to live.

The Project Priority Chart

The project priority chart is one simple mechanism that every project team should use to help align team members' effort [this chart is also described in *Adaptive Software Development* (Highsmith 2000) and *Crystal Clear* (Cockburn 2002)].

At the start of the project, the executive sponsors and the developers discuss and write down the single aspect of the development that everyone should attend to. It may be time-to-market, defect reduction, response time, ease of learning to use, speed of expert usage, memory used, extensibility, or ease of maintenance. They may write a second one, such as time to market and ease of casual use.

They then select, from among all the other desirable characteristics the team should strive for, those three or four that the team should be willing to sacrifice in order to achieve the main two.

From this exercise, each person sees what sorts of trade-offs are permitted on the project and how to prioritize their actions. With a modicum of goodwill between team members, simply writing the choices down in a joint meeting and referring to it periodically gets goal alignment close enough.

The project priority contract addresses a common problem: The executive sponsor wants the software out soon (to hit a market window), but the programmers want to "design it *right*" (delaying their output to improve the design aesthetics). Or the reverse may be true: The programmers are used to working fast and sloppy to hit market windows, and the sponsors want them to take more time and make fewer mistakes. In these cases, the entire organization suffers for a simple, correctable miscommunication of the desired priorities (assuming that the reward structures in place align with the priorities being requested).

Sometimes the priorities need to change in the middle of a project. For example, a competitor may come out with a new product. At that instant, it may become important to reverse priorities, shifting from development speed to defect freedom or vice versa. Should this happen, the sponsors should get the team members together again and announce the shift in priorities.

Amicability and Conflict
Amicability is the willingness of people to hear the thoughts of another person with good will and to speak without malice.

Amicability is the weaker cousin to trust. Trust is wonderful and should be nurtured, but amicability is easier to achieve within a group and still confers advantages. I always watch the amicability level in an organization to learn to what extent information is being revealed versus concealed in conversations.

When people conceal information from their colleagues, they lower the rate of information discovery, which raises the lost opportunity cost as well as the overall cost per idea developed.

Amicability permits successful conflict to occur when the project goes through a stressful period. The people, knowing that the others are not intending to be hurtful, can look past the current disagreement toward resolving the issues.

One might think that removing all conflict from a project team would be the best, but that turns out not to be the case. People need to be able to disagree, in order to identify design problems! I was surprised to find one organization that suffered from too little conflict:

> **Not Enough Conflict**
> In a church organization I visited, each staff member was employed for as long as she wished. The group cherished virtues of humility, peacefulness, and amicability. The unsuspected negative effect that accumulated was the absence of both disagreement and initiative!
>
> Each person would think twice (or more) before criticizing someone else's idea, for fear of being seen as seeding discord or of disrupting the group. People would also think twice (or more) before taking initiative, lest they be considered glory hungry or power hungry.
>
> The net result was that projects moved very slowly.
>
> Before you start offering suggestions for this group, recall the values of the group. They will only improve their development practice when they can find ways to disagree without jeopardizing their values of humility and amicability.

Schrage (1999) describes the intentional use of small doses of conflict to get people to meet and learn to talk with each other. This is like introducing a weakened form of a virus so that the body can build ways of handling the stronger virus:

> **Deliberate Conflict**
> "... [A]ccording to some reports, engineers on the 777 design-build teams deliberately introduced conflicts with other systems into their proposed designs.
>
> "... Although Boeing officially acknowledges only that interferences naturally evolved, according to at least one mechanical engineer, some of those interferences were intentional. Why? So that engineers in one part of Boeing could use the interference to find the people in other parts of the company with whom they needed to discuss future design issues. ... [T]he software's ability to notify appropriate parties about interferences became, at least in some instances, a tool to forge interactions between various groups within Boeing.
>
> "... The resulting conversations and negotiations resolved design conflicts before more serious problems could emerge."

Citizenship within Working Hours

Good citizenship is a matter of acting in ways that benefit others. Citizenship is illustrated by people

- Getting to meetings on time
- Answering questions from other people
- Bothering to mention things that they notice
- Following group coding conventions
- Using code libraries

Citizenship permits programmers who disagree about coding styles to nonetheless create a common coding standard for themselves. As many lead programmers have said, "It's not what I would like, but I recognize that many ways work and selecting any one of them is better than not selecting any at all."

Helping other people in the company is a characteristic of citizenship. Dixon (2000) reports on the strong effect of taking time to help other people. She cites, among many examples, a woman at Tandem Computers who was asked about taking away from her work time to answer questions posted on the corporate discussion boards. The woman responded, "Answering questions like this is part of being a good company citizen."

I often find that workers show citizenship and sacrifice from the moment they start work, and management takes too much advantage of it. People join a new company and work overtime, thinking that after they contribute this extra work the company will respond in kind and give them more recognition and time off. What they don't realize is that their bosses and colleagues assume that however they work in the first month is how they will work and act forever. As a result, people regularly get poor evaluations for dropping their working hours from 65 down to a'*mere*'50!

I am afraid that managers will use the pretext of good citizenship to coerce people into working yet more overtime. Read *Death March* (Yourdon 1998) for examples of this.

Citizenship should be encouraged *within* normal working hours, not as a means of lengthening normal working hours. There are plenty of ways in which to apply citizenship within working hours.

Hostile *XP* versus Friendly *XP*

To round out this discussion, let's look at the consequences of working with and without attention to community. I choose to discuss Extreme Programming (*XP*), because although communication and community are core values within *XP*, I have seen it practiced with and without that community: "friendly" *XP* and "hostile" *XP*, as it were. The difference is profound.

The three following situations are some in which customers and programmers might magnify their differences and create a hostile *XP*:

- The customers are not quite sure what they want. The programmers insist, "Tell us what to build," so the customers say something. The programmers build exactly that and then ask, "Tell us what to build next."

  In this situation, neither group is really sure what the correct thing is to build next. The programmers escape the pressure of the situation by shifting the burden over to the customers (which they are allowed to do). The customers experience the situation as unsettling: There is little time to reflect, examine, experiment, and sort out options.

As a result, the customer's instructions over the course of succeeding iterations conflict with each other: "Build this. ... No, now build this. ... No, try building that now". Both parties become depressed about the lack of clear progress.

- The programmers do whatever the customers say, even if they are sure that the idea is silly.

  As with the story "Not Enough Conflict," a project suffers when the developers don't mention problems they notice. The project loses the creative interplay of sharp programmers offering their insights to refine the requests of the customers.

- The customers tell the programmers that a particular feature will be coming up and ask if the programmers will please design the system to handle that gracefully. The programmers cite a series of the *XP* mantras: "Keep it simple," "You aren't gonna need it," "We'll do the simplest thing that will possibly work," and they ignore any suggestion of what to build into the software.

  The consequence is that the designers run through a sequence of designs everyone knows are incorrect, until the critical requirements finally appear. By then, time has been spent redesigning the system several times. In the cases I have encountered, the programmers were happy about the exercise and the sponsors were unhappy.

In each of these cases, the programmers withheld information. Withholding their own thoughts and experience from the discussion, they abdicated responsibility toward the overall project. By doing so, they damaged the project by concealing from view superior development strategies.

In friendly *XP*, practiced with community, the three situations play out differently. In each case, the programmers actively share their views, experiences, cost estimates, and solutions.

- In the first situation, not knowing what to build next, the programmers help the customers gain experience in voicing what they want. They can do this by producing small working prototypes tailored to discovering the desired characteristics.
- In the case of the silly idea, the programmers volunteer their information through amicable dialogue: "I'm not sure you really want this thing you asked for. It will be so-and-so difficult to implement and has the following roll-on effects." The customer might still request the feature, but quite often, the person had no idea about those effects and is happy to have them mentioned. Usually, customers appreciate the insights, whether or not they change the request.
- In the story-sequencing situation, the programmers help the customers by finding those story cards that affect the decisions in question. They can then jointly consider in which order the cards should be tackled. The new order might not simply ask for more functionality along a business-value trajectory but might converge more quickly on the actual system the customers want.

Any development methodology, even one that advocates amicability and community, can be practiced without it to the detriment of the project.

Building "Team" by Winning

Team spirit was once built through singing company songs and attending company functions. (Any of you still have your IBM songbook?) When singing on the job went out of style, nothing immediate took its place.

Some companies start projects with one or several days of off-site team-building. This is good, even if it is good mostly because the people recognize the effort the company is putting forth to show that teamwork is important. Although not every team-building exercise actually builds a team, a number of successful teams have pointed to

their team-building days at the start of the project as having helped them work together more effectively. As a result, their company leaders consider the money well spent and plan on continuing the tradition.

Programmers give mixed reviews to outside-of-work team-building exercises. Several said, roughly, "I'm not interested in whether we can barbeque together or climb walls together. I'm interested in whether we can produce software together."

What *does* build teams? Luke Hohmann writes:

> "The best way to build a team is by having them be successful in producing results. Small ones, big ones. It doesn't matter. This belief has empirical support; see, for instance, Brown (1990). Fuzzy team building is (IMO) almost always a waste of time and money."

Support for this is also found in Weick's description of the importance of "small wins" (Weick 2001) as well as in interviews of successful project managers.

One successful project manager told of a key moment when the project morale and "team"-ness improved. We found the following elements in the story:

- The people, who sat in different locations, met each other face to face.
- Together, they accomplished some significant result that they could not have achieved without working together.
- At some point, they placed themselves in some social jeopardy (venturing new thoughts, or admitting ignorance) and received support from the group when they might have been attacked.

The second of those characteristics is "producing results," as Luke Hohmann mentions. The first and the third build amicability, the positive absence of fear and distrust.

Team Cultures and Subcultures

The project team itself creates a mini-culture. That mini-culture sits within the culture formed within the larger organization and also within the dominant national culture around it.

Often, the programming project ends up with its own culture, different from the national or corporate cultures in which it is embedded. People on the project find this useful, because they have a greater need to trade information about what is working and what is about to break.

Sometimes, the wider organization tolerates this different culture, and sometimes it fights back. One person who had experienced the resistance wrote, "Watch out for the organizational antibodies!"

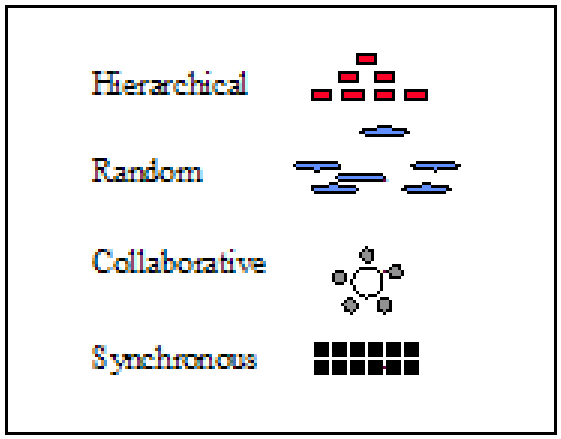


Figure 3-19. Four organizational paradigms.

Cultures and their values can be characterized in many ways. In one characterization (Constantine 1995), sociologists name four culture types by their communication, power, and decision-making habits (Figure 3-19). These four culture types are described in the following paragraphs:

Hierarchical cultures have the traditional top-down chain of command. Typically, older, larger corporations have a hierarchical culture. Many people internalize this as the dominant or natural or default corporate culture as they grow up, and they have to be trained away from it.

Random is the opposite of hierarchical. It indicates a group in which there is little or no central control. Many start-up companies work this way. Some people consider *random* a fun way to work and regret the loss of the small, informal group when the company grows. Others find it stressful, because there are no clear points of control.

Collaborative groups work by consensus. I had the opportunity to encounter a collaborative group in action at Lucent Technology:

> **Consensus Culture at Work**
> Someone in the organization decided that use cases would be a good way to capture requirements and asked me to teach a course to the people on a project.
>
> I met the team leads (who are actually called *coaches*, because in a collaborative culture they don't lead, of course, they *coach*).
> About a month later, I was called to teach it again, for more of the group.
>
> Several months after that, I was asked to lecture one last time, for the entire department. Even though the coach had decided that use cases were good, the group was not going to use them until they had all had a chance to see and understand them.
>
> The behavior of the coach in the final meeting was interesting: He programmed on his laptop while I taught. He was physically present in the room, but only just barely. Far from being insulting, I found his actions fully appropriate in light of the value systems in play around his situation. As a senior developer, he demonstrated that he was still contributing directly to the team's work. As a coach, he demonstrated support for the material being presented, which he was hearing for the third time. Thus, his behavior was a natural expression of his place in two professional societies: developer and coach.

Synchronous, or "silent," groups are the opposite of collaborative. They coordinate action without verbal communication, with people performing their roles without attempting to affect the other roles' work styles.

Constantine gives two examples of synchronous teamwork. The first comes from a scene in the movie "Witness," in which members of the Amish community raise a new barn in a single day, scarcely uttering a word. The second comes from an accident that happened inside a hospital, when a heavy table fell on a person's leg. Without speaking to each other, the people in the room took coordinated action: Two lifted the table, one held the person's hand, one went to call for an X-ray, and one went to get a gurney.

In both cases, the people involved knew the rules of the situation and the goals and the roles involved. They could simply step into a needed role. Constantine points out that in a synchronous environment, "team members are aligned with the direction established by a shared vision and common values."

It may turn out, in an odd twist, that programmers operate within a silent or synchronous culture. If this is true, it will be interesting to see how the cooperative game gets reshaped to fit that cultural pattern. Certainly, the

current wave of development methodologies, including *XP* and *Crystal*, require much more conversation than previous ones. Either the programmers will shift their culture, or the methodologies will have to adapt.

In many organizations, programmers are expected to work massive overtime. It was a great shock to me to move from one such organization to the Central Bank of Norway, where personal life was strongly valued and overtime discouraged:

> Overtime Lights at Norges Bank
> At the Central Bank of Norway, the official workday ended at 3:30 p.m.
>
> On a typical day, that is the time I suddenly waken from whatever else I am doing and ask myself what I *really* want to get done that day. As a result, I found myself wandering the halls at 3:45, trying to "really get some work completed before the end of the day" and unable to send faxes, get signatures on paper, or get questions answered. The staff really *did* go home at 3:30!
>
> Then, at 5:00, the lights automatically turned off! I learned how to turn on the "overtime lights" but got a second shock when the light turned off again 7:00 p.m. (*"You really, really ought to go home now.*")

Cultures also differ by their attitude toward frankness and politeness in speech. The Japanese are renowned for working to preserve face, while Americans are considered frank. Frankness is taken to extremes in some places, such as MIT, Stanford, and Israel. An Israeli friend was coaching me in direct speaking: When I saw him after he had to miss a review meeting I said, "We missed you at the meeting." He replied, "In Israel we would say, 'Why weren't you there?'"

In other cultures, such as the church organization described earlier, even disagreeing mildly or taking initiative are considered slightly negative behaviors, signs of a person having excessive ego.

As a result of differences around frankness in speech, people coming from different cultures sometimes have difficulty working together. The overly frank person strikes the other as rash and abrasive, while the overly polite person strikes the other as not forthcoming, not contributing.

Professional Subcultures

Each profession also builds its own culture, with its own cultural values and norms. Project managers have theirs as do experienced object-oriented developers, relational database designers, COBOL programmers, salespeople, users, and so on. Even novices in each group have their own values and norms, distinct from the experts. Here are a few:

- Project managers need an orderly attitude to sort out and predict delivery dates and costs and the complex dependencies within the project.
- OO programmers need quiet time, abstract thinking ability, and the ability to deal with the uncertainty of simultaneously evolving programming interfaces.
- Requirements analysts rely on thorough thinking, going through the requirements and the interfaces one line at a time, looking for mistakes.
- Marketing people benefit from strong imaginations and people skills and dealing with the constant surprises that the market (and the programmers) throw at them.

Let's consider programmers' "non-communicative and anti-social" behavior for a moment. Actually, as a number of them said when they wrote to me, they do like to talk … about technical things. They just don't like talking about things they consider uninteresting (baseball games and birthday parties, perhaps). What they really detest is being interrupted during their work. It turns out that there is a good reason for this.

Software consists of tying together complex threads of thought. The programmer spends a great deal of time lifting and holding together a set of ideas. She starts typing, holding in her mind this tangled construct, tracing the mental links as she types.

If she gets called to a meeting at this point, her thought structure falls to the ground and she must rebuild it after the meeting. It can take 20 minutes to build this structure and an hour to make progress. Therefore, any phone call, discussion, or meeting that distracts her for longer than a few minutes causes her to lose up to an hour of work and an immense amount of energy. It is little wonder that programmers hate meetings. Anti-social behavior, meeting-avoidance in particular, is a protective part of their profession.

Thus, the values of each group contribute to their proper functioning, and the differences are necessary for the proper functioning of the total organization, even though they clash.

It would be nice to say that all of the values and norms are constructive. Not all are, though.

An example introduced earlier is the Invent-Here-Now imperative. It is developed as a cultural value and norm all the way through college. In most organizations, however, inventing new solutions where old ones already exist is counter-productive to the aims of the organization. The ideal norm would be to scavenge existing solutions wherever possible and to invent only where it leads the organization past its competitors.

Adapting to Subcultures

Most people's initial reaction is to force one group's values on the other groups.

- Researchers in formal development techniques want more math to be taught in school.
- Managers who are uncomfortable with iterative development want their programmers to get the design right the first time.
- The programmers, frustrated with not being able to communicate with their managers, want the managers to learn object-oriented programming prior to managing a project.

There are two problems with the make- *them*- change approach:

1. The less serious problem is that it is really, really hard to get people to change their habits and approaches.
2. The more serious problem is that we don't yet understand the subcultures. To force them to change their values is a bit like prescribing medicine without understanding the defense mechanisms of the body.

In the face of this situation, there are things that the industry can do, things that a few individuals can do, and things that everyone can do.

As an industry, we can

- Encourage more ethnographic studies of software development groups, as Hovenden (2000) has done
- Identify and understand the norms in play, showing the contribution of each to the organization
- Experiment with cultural changes

Every consulting company can benefit from employing a social anthropologist or ethnographer. That person will help the consulting team understand the social forces in play on their projects, which will enhance the team's effectiveness.

People who are fluent in several specialties, such as programming and database design, programming and project management, or teaching and designing, can act as translators. These people help by converting statements phrased in one normative value set into sentences meaningful within a different value set. A number of people who perform this function have written to me to describe the difficulty and necessity of this role.

Finally, everyone can practice patience and goodwill in listening. Pretend that the other person's sentences, however crazy they may sound to you, make sense in the other culture's value system. Listen that way first, and *then* decide if you still need to disagree.

4. TEAMS ECOLOGIES

A software project sets up a small ecosystem made up of personalities from diverse cultures. We have seen some elements of the ecosystem, including

- Walls acting as barriers and open spaces acting as conduits
- People in their professional specialties acting as interacting subspecies
- Individuals with strong personalities changing the way in which the ecosystem works

Everything affects everything: the chairs, the seating, the shape of the building, whether people share a native language, even the air conditioning.

> **Lizards and Penguins**
> At one company, moving from our old building to a new one nearly caused fights.
>
> In the old building, we each had a private office, and each office had its own thermostat. In the new building, we would still have private offices, but there was only going to be one thermostat for every two offices. Each adjacent office pair had to use the same temperature setting.
>
> Suddenly, the workforce polarized into those who liked warm offices (the "lizards") and those who liked cold offices (the "penguins"). People were jockeying for positions so they could share the thermostat with someone of similar temperature preferences.

In some work situations, it is hard for people to change companies. In other situations, people change jobs every few months. The two situations create different attitudes and behaviors in the workforce.

Every job role and every person affects every other. Key individuals play a more significant role in shaping the ecosystem than others. They focus or, more frequently, block conversations. When they leave, the entire network of relationships changes.

Each project's ecosystem is unique. In principle, it should be impossible to say anything concrete and substantive about all teams' ecosystems. It is.

Only the people on the team can deduce and decide what will work in that particular environment and tune the environment to support them.

If the people on the team understand some key characteristics of humans and of methodologies, they can look around, introspect about what they observe, and construct a best first guess as to what conventions and policies might work well for them, suiting their own strengths and weaknesses.

The people on the teams will naturally reexamine and adjust their conventions over time, periodically or whenever a major event changes their ecosystem (as when a particularly influential individual joins or leaves the organization).

The set of conventions and policies I refer to as the team's *methodology*. As we will see in the next chapter, a methodology is a personal thing—"a social construction," to quote Ralph Hodgson of IBM.

Considering the methodology as the team's own social construction is useful. It highlights the idea that no methodology will work "straight out of the box." The team members will have to adapt both themselves and the methodology to work together to create their own, local, effective ecosystem.

Ecosystems and methodologies have this interesting characteristic in common: If the team members construct many complicated rules for themselves, they tie themselves to a narrow ecological niche.

However, narrow ecological niches are notoriously fragile, and the market has a nasty habit of changing the terrain around a company. The many rules that ensure effective behavior in one ecological setting are ill suited for use in another.

In biology, we use the word "become extinct." In business, the phrase is "go out of business."

If, on the other hand, the team creates and periodically updates a few well-placed guidelines, it can draw on the intelligence, pride-in-contribution, communication, and spontaneity of its members. The people will adapt those guidelines to the situation at hand, achieving robust behavior in the face of technological, social, and market surprises. Dee Hock, designer of the highly decentralized VISA system in the 1960s and 1970s, wrote (Hock 1999):

"Simple, clear purpose and principles give rise to complex, intelligent behavior. Complex rules and regulations give rise to simple, stupid behavior."

What Should I Do Tomorrow?

Walk around your place of work.

Notice

- The convection currents of information
- The drafts
- The information radiators
- The separate communities of practice
- The background conversation complimenting or denigrating other groups in the organization

See

- How you can improve the flow of information and reduce the *erg-seconds* required to detect and transmit critical information
- If you can collocate your team
- What it takes to partition the project so that teams are located around their communication needs

Try

- Removing partitions between people
- Pair programming
- Arranging for daily visits between programmers and business experts
- Micro-touch intervention (people making small changes that they don't mind making but that result in their pulling more in the same direction)
- Listening to the words of someone in a different professional specialty according to *her* cultural norms, not your own
- Translating between two subcultures in their own cultural terms

Observe the interaction between your methodology's rules and your project's ecosystem. Note the fits and the misfits and the influence of a few key individuals.

Consider what conventions or policies might improve the way in which your group gets things done. They may be conventions about seating, tools, working hours, process, lighting, meetings: anything.

Do this, and you are halfway to tailoring your methodology to fit your organization.