12

Teamwork

Contents

Related Topics

Team structure: Chapter 13
Peopleware: Section 2.2
Motivation: Chapter 11
Signing up: Chapter 34

THE MOVIE *WITNESS* CAPTURES THE MARVEL of an Amish barn raising. Shortly after dawn, several dozen farmers and their families arrive at the site of a newlywed Amish couple's farm to put up a barn. The farmers raise the supporting members of the frame in the early morning, and by noon they complete the framing from the ground up to the rafters. After lunch, the farmers nail on the sides and the roofing. They work quietly, happily, and hard. And they do their work without the aid of electricity or power tools.

There are jobs for everyone from the youngest girls and boys to the oldest men and women. The kids carry water, tools, and supplies to the adults. The oldest men and women direct others' activities. Two men who might be viewed as rivals put aside their differences and cooperate toward their common goal. Several dozen people contribute their skills in ways that best support the common cause. The result of all this is an incredible feat: By the time the sun goes down, a team of farmers (and their families) has built an entire barn in a single day.

Several things are notably absent from this remarkable scene. The Amish use traditional tools and traditional building methods to erect the barn. No one delays the project to argue the merits of adopting more modern approaches. No one gripes because they can't use power tools. None of the farmers takes time out to talk to his stockbroker on his cellular phone. No one embroils the project in a debate about whether the newlywed couple really needs a new barn, or whether they should open a bed and breakfast instead. No one leaves the Amish community at the end of the day because the pressure of the project has been too intense.

The farmers are united by a clear vision of the barn they will build and by the seemingly impossible challenge of building it in about 15 hours. When they're done, they feel a strong sense of accomplishment—both individually and collectively—and they feel even stronger ties to their neighbors than they felt before.

If ever there was a model of a perfect team, this is it.

Case Study 12-1. You Call This a Team?

The Giga-Quote 2.0 project team had five team members: Joe, Carl, Angela, Tomas, and Tina. The project was organized with Tomas as the chief programmer on a chief-programmer team. In seeming contradiction to the chief-programmer-team structure, Tina served as the informal project manager and Angela as the QA manager. Decisions were made primarily by consensus and enforced by peer pressure. None of the other group members seemed to recognize Tomas's authority as chief programmer.

Early in the project, the group experienced problems with group dynamics. The group had several strong-willed individuals with differing opinions, and four of the group members described their group as having "all chiefs and no Indians." Tomas, the nominal chief programmer, denied that there were any personality problems. The group spent considerable energy in heated discussions about technical issues and project direction. These discussions often failed to reach a resolution. For example, the group decided to skip risk analysis, not because they had technical reasons to skip it but because they thought it would be too controversial.

The group (except for Tomas) recognized from the beginning that they had a group-dynamics problem, but they actually seemed to enjoy conflict. They felt that strong disagreements had resulted in spirited participation by all group members and thorough examinations of major technical issues.

In fact, the poor group dynamics had serious damaging effects. The group originally planned to develop Giga-Quote 2.0 under a classical waterfall lifecycle model, but they did not follow their plan. Different group members described their current lifecycle model as anything from an "overlapping waterfall" model to "chaos."

(continued)

Case Study 12-1. You Call This a Team? *continued*

The group also did not follow recommended development practices. For example, the group had planned to use code inspections and reviews, but these were voluntary and they quit using them after they caused too many arguments. The group also could not agree on acceptance criteria for the end of the architecture, design, and unit-construction phases. After a few contentious initial attempts at design, they decided to proceed with coding and work out the details as they went along.

When the Giga-Quote group reached integration, they experienced a big bang. They had allowed for 1 week to integrate 9 months of work. Group members had been confident about integration because they had defined the interfaces at the "data structure level." But they had defined only the syntax; their poor group dynamics had prevented them from defining interface semantics.

As the deadline approached, personality conflicts came to a head. Carl folded his arms and protested, "I've said all along that this design approach would never work. That's why we're having all of these integration problems. We should have used my design." Angela, Joe, and Tina decided that they'd had enough and left for other positions within the company. That left Tomas and Carl as the only surviving team members. Tomas disliked Carl so much that he committed to work virtually nonstop until he completed the project if management would remove Carl. He finally completed the project 7 months after its original delivery date.

12.1 Software Uses of Teamwork

It takes more than just a group of people who happen to work together to constitute a team. In their book *The Wisdom of Teams*, Katzenbach and Smith (1993) define a team as "a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable."

Teamwork can come into play on software projects on any number of specific tasks:

- Developing and reviewing the project's requirements
- Developing the project's architecture and the design guidelines that will be used by the whole project
- Defining aspects of the technical environment that will be used on the project (including the programming languages, compilers, source-code libraries, code generators, editors, and version-control tools)
- Developing coding standards that will be used by the whole project
- Coordinating work on related pieces of a project (including defining interfaces between subsystems, modules, and classes)

- Designing difficult parts of the system
- Reviewing individual developers' designs and code
- Debugging difficult parts of the system
- Testing of requirements, design, and code
- Auditing a project's progress
- Maintaining software once it has been built (including responding to maintenance requests and making emergency fixes)

Although any of these tasks could be done by a single person, they can all benefit from the involvement of two or more brains, which requires interaction among project members. If the brains are working in cooperation, the whole can sometimes be greater than the sum of its parts. If the brains are on a collision path, it can be less. A "team" exists whenever two heads together are better than two heads individually.

Groups and Teams

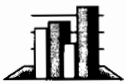
Not all groups are teams. Some projects can be done well enough by a group of cooperative people who don't form into a team. Some projects don't call for the level of commitment that teamwork entails.

12.2 Teamwork's Importance to Rapid Development

CROSS-REFERENCE
ır general reasons that
eopleware focus is key
to success on a rapid-
velopment project, see
People" in Section 2.2.

Small projects can get away with not addressing teamwork issues, but they will benefit from addressing them. Large projects are group efforts, and characteristics of the groups play an important role in those projects' success.

Variations in Team Productivity



HARD DATA

Researchers have found differences in individual productivity on the order of 10 to 1. Researchers have also identified dramatic differences in the productivity levels of entire teams. After analyzing 69 projects at TRW and other companies, Barry Boehm concluded that the best teams were at least 4 times as productive as the worst (Boehm 1981). DeMarco and Lister identified productivity differences of 5.6 to 1 in a study of 166 professional programmers from 18 organizations (DeMarco and Lister 1985). An earlier study of programming teams observed a 2.6 to 1 variation in the times required for teams to complete the same project (Weinberg and Schulman 1974).

This difference holds even among groups of developers with similar levels of experience. In one study of seven identical projects, the developers were all professional programmers with several years of experience who were

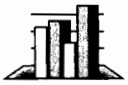
enrolled in a computer-science graduate program. The products still ranged in effort by a factor of 3.4 to 1 (Boehm, Gray, and Seewaldt 1984). Similarly, Valett and McGarry reported 2-to-1 and 3-to-1 differences in productivity between different projects at NASA's Software Engineering Laboratory (Valett and McGarry 1989).

CROSS-REFERENCE
For details on individual differences in productivity, see "People" in Section 2.2.

When you cut across this set of studies, the bottom line is that among groups with different backgrounds and different levels of experience, there is about a 5-to-1 difference in productivity. Among groups with similar backgrounds and similar levels of experience, there is about a 2.5-to-1 difference in productivity.

Cohesiveness and Performance

If your experience has been like mine, you'll agree that members of cohesive groups work hard, enjoy their work, and spend a great percentage of their time focused on the project goals. As Case Study 12-1 illustrates, participants in projects with poor team dynamics are frequently unfocused and demoralized, and they spend a great deal of their time working at cross purposes.



HARD DATA

In a study published in 1993, B. Lakhanpal reported on how group cohesiveness, individual capabilities, and experience were related to overall project performance on 31 software projects (Lakhanpal 1993). The projects ranged in duration from 6 to 14 months and in size from 4 to 8 developers. Lakhanpal found that group cohesiveness contributed more to productivity than project members' individual capabilities or experience did. (Individual capabilities were a close second.)

Lakhanpal points out that managers commonly assign project members based on level of experience and individual capabilities. The study of 31 projects suggests that managers who are concerned about rapid development would be better off to assign developers based on their abilities to contribute to a cohesive team first and only then based on their individual capabilities.

Case Study 12-2. A High-Performance Team

One illustration of a productive team is the group of Amish farmers at the beginning of the chapter. That might not have much to do with software teams, but, then again, it might.

The most productive team I ever worked on shared many characteristics with the Amish barn raisers. After I graduated from college, I worked for a startup company in the actuarial consulting business. The owner was cheap, so he

(continued)

Case Study 12-2. A High-Performance Team, *continued*

hired fresh-out-of-college graduates to minimize his labor costs rather than pay more for experienced developers. Our group members soon found we had a lot in common since our jobs at this company were the first professional, adult jobs any of us had had.

Because it was a startup company, we broke a lot of new ground for the company, and we worked to a lot of challenging deadlines. We set up friendly rivalries, such as buying each other donuts for finding bugs in our code. Because we were recent college graduates, we had more responsibility than we had ever had before.

We had all been hired with the same job title, so it wasn't too long before we started referring to ourselves as "Analysts Incorporated." Like a lot of other teams, we had a set of in-jokes and rituals that people outside of the team had a hard time understanding. We spent our programming time in a programming bull pen, and since we thought our boss pushed us way too hard, we sometimes put up a sign that read "Closed Door Analyst Meeting," closed the door, and sang spirituals while we programmed.

One day I took a new hire to get a joint assignment from our supervisor. The supervisor was an honorary member of Analysts Incorporated, and he and I made jokes and traded barbs about previous projects while we went over the new project. After a few minutes, the supervisor needed to attend a meeting, so he excused himself and said he would check back after the meeting to see how our work was going. After he had left, the new hire said, "It doesn't seem like we got much accomplished. All you two did was make jokes and insult each other, and we don't really know what we're supposed to do, do we? When are we going to get our assignment?" As I explained the assignment we had just received point-by-point to the new hire, I realized how far our group had gone toward establishing its own mode of communication and unique identity.

12.3 Creating a High-Performance Team

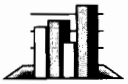
Productive teams are sometimes characterized as teams that have jelled or as teams that are highly cohesive. What characteristics does a high-performance, jelled, cohesive team have? The team has:

- A shared, elevating vision or goal
- A sense of team identity
- A results-driven structure
- Competent team members
- A commitment to the team

- Mutual trust
- Interdependence among team members
- Effective communication
- A sense of autonomy
- A sense of empowerment
- Small team size
- A high level of enjoyment

In 1989, Larson and LaFasto published a study that found unusual consistency among the attributes of highly effective teams. This was true for teams as diverse as the McDonald's Chicken McNugget team, the space-shuttle Challenger investigation team, cardiac-surgery teams, mountain-climbing teams, the 1966 Notre Dame championship football team, and White House cabinets (Larson and LaFasto 1989). The following sections explain how each of the attributes apply to software teams.

Shared, Elevating Vision or Goal



HARD DATA

Before the project gets really rolling, a team needs to "buy in" to a common vision or common goals. The Amish farmers shared a common vision of the barn they were going to raise, why they were raising it, how they would raise it, and how long it would take. Without such a shared vision, high-performance teamwork cannot take place. Larson and LaFasto's study of 75 teams found that in every case in which an effectively functioning team was identified, the team had a clear understanding of its objective.

Sharing a vision is useful to rapid development on several levels. Having agreement on the project vision helps to streamline decision making on the smaller issues. Small issues stay small because the big vision keeps them in perspective, and everyone agrees on the big vision. The team is able to make decisions and then execute them without squabbling and without revisiting issues that have already been decided. A common vision builds trust among the team members because they know that they are all working toward the same objective. It also helps to keep the team focused and avoid time-wasting side trips. An effective team builds a level of trust and cooperation that allows them to outperform a collection of individuals with similar skills.

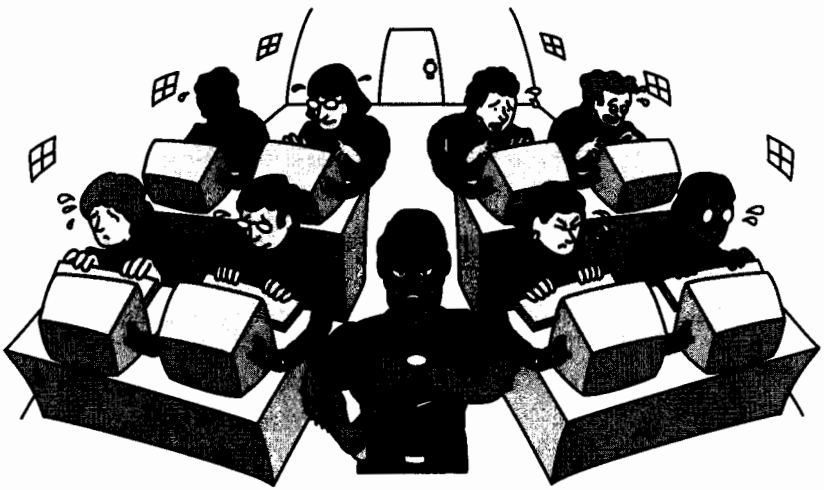
Occasionally a highly cohesive team will lock onto a shared vision that is at odds with the organization's objectives. In this case, the team might get a lot of work done, but not the kind of work the organization needs. To be productive, a cohesive group needs to have a focus compatible with the organization they are a part of.

Challenging Work

The shared vision can be of something important—such as putting a man on the moon by 1970—or it can be of something relatively trivial—getting the latest update of the billing system out 3 weeks faster than last time. The vision can be virtually arbitrary, but as long as the whole team shares it, it will serve the same purpose of helping to bring the team together.

To have a motivating effect, the vision also needs to be elevating. The team needs to be presented with a challenge, a mission. The Amish farmers responded to the seemingly impossible challenge of building an entire barn in one day. High-performance teams don't form around ho-hum goals. "We'd like to create the third-best database product and deliver it in an average amount of time with below average quality." Ho hum. Yawn. No team is going to rally around that. (See Figure 12-1.)

But the response to challenge is an emotional reaction, and it is influenced as much by the way the work is assigned or described as by the work itself. Here's a restatement of the ho-hum goal: "We're going to create a database product that will take us from zero market share to 25-percent market share in 18 months. Marketing and production need absolutely reliable schedule estimates to pull this off, so we're going to make it our goal to set internal milestones and a final ship date that we can meet with 100-percent assurance." A team just might form around the vision of 100-percent scheduling accuracy.



"Aren't you a team? Why aren't you getting more done? Work Harder."

Figure 12-1. *The way you present the project will determine whether your team sees its job as a mission or a hardship.*

A real team needs a mission, and how you frame the project has a lot to do with whether the team sees a mission.

Sense of Team Identity

As team members work together toward their common vision, they begin to feel a sense of team identity. Teams name themselves. "The Black Team." "Analysts Incorporated." "The Camobap Boys." "Seance." Some teams have a team motto. Others, like IBM's famous Black Team, adopt a team dress code. Their senses of humor begin to gravitate toward one another's, finding humor in things that other people don't understand. They look for common characteristics that differentiate them from the rank and file. IBM's Black Team continued even after all the original team members had gone. That's a strong sense of identity. Smart companies reinforce the team's sense of identity by providing team T-shirts, notepads, mugs, and other paraphernalia that validate the team as a legitimate entity.

Team members allow their sense of team identity to overshadow their identities as individuals. They derive satisfaction from the team's accomplishments. They see the common goal as more important than their personal agendas. They have the opportunity to achieve something with the team that they couldn't achieve individually. For example, from 1957 to 1969, the Boston Celtics won the NBA championship 11 times without ever having a player among the top three scorers in the league. The team came first. Team members talk more about what "we" did than what "I" did, and they seem to take more pride in what "we" did than in what "I" did.

Along with the feeling of identity, high-performance teams often develop a sense of eliteness. Team members get to be team members by going through some kind of trial by fire—a rigorous interview and audition process, successfully completing an especially challenging first assignment, or being recruited on the basis of exceptional past performance.

One project I know of used up all the money in its morale budget and later made team shirts available for $30 apiece to new team members. That is a good example of how *not* to create team spirit. (The message the team hears from the company is that the team ranks so low in the company that it has to buy its own team shirts.) The same project was identified as responsible for one of the company's "low-revenue products." Characterizing a team that way is a mistake because that isn't the sort of identity that most teams will rally around.

Results-Driven Structure

CROSS-REFERENCE
For more on team structures, see Chapter 13, "Team Structure."

You can structure teams for optimal output, or you can structure them in such a way that it is almost impossible for them to produce anything at all.

For rapid development, you need to structure the team with maximum development speed in mind. You don't put John in charge just because he's the owner's cousin, and you don't use a chief-programmer team structure on a three-person project when the three people have roughly equal skills.

Here are four essential characteristics of a results-driven team structure:

- Roles must be clear, and everyone must be accountable for their work at all times. Accountability is critical to effective decision making and to rapid execution after the decisions have been made.
- The team must have an effective communication system that supports the free flow of information among team members. Communication must flow freely both from and to the team's management.
- The team must have some means of monitoring individual performance and providing feedback. The team should know whom to reward, who needs individual development, and who can assume more responsibilities in the future.
- Decisions must be made based on facts rather than on subjective opinions whenever possible. The team needs to be sure that the facts are interpreted without biases that undercut the functioning of the team.

There are practically an infinite number of team structures that can satisfy these essential characteristics. It is amazing that so many team structures do not.

Competent Team Members

Just as team structures are chosen for the wrong reasons, team members are often chosen for the wrong reasons: for example, they are often chosen because they have an interest in the project, because they are cheap, or most often simply because they are available. They are not chosen with rapid development in mind. Case Study 12-3 describes the way in which team members are typically selected.

Case Study 12-3. Typical Team-Member Selection

Bill had a new application to build, and he needed to put a team together fast. The project was supposed to take about 6 months and was going to involve a lot of custom graphics, and the team would have to work closely with the

(continued)

Case Study 12-3. Typical Team-Member Selection, *continued*

customer. It should take about four developers. Ideally, Bill thought, he'd like to get Juan, who had worked on GUI custom graphics on that platform before, and Sue, who was a database guru and great with customers. But they were both busy on other projects for the next 2 or 3 weeks.

At the manager's meeting Bill found out that Tomas, Jennifer, Carl, and Angela would be available at the end of the week. "They'll do OK," he said. "That will let us get started right away."

He planned the project this way: "Tomas can work on the graphics. He hasn't worked on this platform before, but he's done some graphics work. Jennifer would be good for the database side. She said she was tired of working on databases, but she agreed to work on them again if we really needed her to. Carl's done some work on this platform before, so he could lend Tomas a hand with the graphics. And Angela is an expert in the programming language. Carl, Angela, and Tomas have had a few problems working together before, but I think they've put their differences behind them. None of them are particularly strong in working with customers, but I can fill in that gap myself."

Case Study 12-3 describes a team that's selected based on who's available at exactly the right time without much concern for the long-term performance consequences. It's almost certain that the team would do better on its 6-month project if it waited the 3 weeks until Juan and Sue were available.

For rapid development, team members need to be chosen based on who has the competencies that are currently needed. Three kinds of competencies are important:

- Specific technical skills—application area, platform, methodologies, and programming language
- A strong desire to contribute
- Specific collaboration skills required to work effectively with others

Mix of Roles

On an effective team, the team members have a mix of skills and they play several different roles. It obviously doesn't make sense to have a team of seven people who are all experts in assembly language if your project is in C++. Likewise, it doesn't make sense to have seven people who are all experts in C++ if no one of them knows the applications area. Less obviously, you need team members who have a blend of technical, business, management, and interpersonal skills. In rapid development, you need interpersonal leaders as much as technical leaders.

FURTHER READING
These labels are not
elbin's but are taken from
nstantine on Peopleware
(Constantine 1995a).

Dr. Meredith Belbin identified the following leadership roles:

- *Driver*—Controls team direction at a detailed, tactical level. Defines things, steers and shapes group discussions and activities.
- *Coordinator*—Controls team direction at the highest, strategic level. Moves the problem-solving forward by recognizing strengths and weaknesses and making the best use of human and other resources.
- *Originator*—Provides leadership in ideas, innovating and inventing ideas and strategies, especially on major issues.
- *Monitor*—Analyzes problems from a practical point of view and evaluates ideas and suggestions so that the team can make balanced decisions.
- *Implementer*—Converts concepts and plans into work procedures and carries out group plans efficiently and as agreed.
- *Supporter*—Builds on team members' strengths and underpins their shortcomings. Provides emotional leadership and fosters team spirit. Improves communications among team members.
- *Investigator*—Explores and reports on ideas, developments, and resources outside the group. Creates external contacts that may be useful to the group.
- *Finisher*—Ensures that all necessary work is completed in all details. Seeks work that needs greater than average attention to detail, and maintains the group's focus and sense of urgency.

Even on a rapid-development project, it's best not to staff a project with nothing but high-performance individuals. You also need people who will look out for the organization's larger interests, people who will keep the high-performance individuals from clashing, people who will provide technical vision, and people who will do all the detail work necessary to carry out the vision.

One symptom of a team that isn't working is that people are rigid about the roles they will and won't play. One person will do database programming only and won't work on report formatting. Or another person will program only in C++ and won't have anything to do with Visual Basic.

On a well-oiled team, different people will be willing to play different roles at different times, depending on what the team needs. A person who normally concentrates on user-interface work might switch to database work if there are two other user-interface experts on the team. Or a person who usually plays a technical-lead role may volunteer to play a participant role if there are too many leaders on a particular project.

Commitment to the Team

CROSS-REFERENCE
more on commitment to
project, see Chapter 34,
"Signing Up."

The characteristics of vision, challenge, and team identity coalesce in the area of commitment. On an effective team, team members commit to the team. They make personal sacrifices for the team that they would not make for the larger organization. In some instances, they may make sacrifices to the team to spite the larger organization, to prove that they know something that the larger organization doesn't. In any case, the minimum requirement for team success is that the team members contribute their time and energy—their effort—and that calls for commitment.

When team members commit, there must be something for them to commit to. You can't commit to unstated goals. You can't commit at any deep level to "doing whatever management wants." Vision, challenge, and team identity provide the things to which team members commit.

Getting project members to commit to a project is not as hard as it might sound. IBM found that many developers were eager for the opportunity to do something extraordinary in their work. They found that simply by asking and giving people the option to accept or decline, they got project members to make extraordinary commitments (Scherr 1989).

Mutual Trust

Larson and LaFasto found that trust consisted of four components:

- Honesty
- Openness
- Consistency
- Respect

If any one of these elements is breached, even once, trust is broken.

Trust is less a cause than an effect of an effective team. You can't force the members of a team to trust each other. You can't set a goal of "Trust your teammates." But once project members commit to a common vision and start to identify with the team, they learn to be accountable and to hold each other accountable. When team members see that other team members truly have the team's interests at heart—and realize that they have a track record of being honest, open, consistent, and respectful with each other—trust will arise from that.

Interdependence Among Members

Team members rely on each other's individual strengths, and they all do what's best for the team. Everybody feels that they have a chance to contribute and that their contributions matter. Everybody participates in decisions. In short, the team members become interdependent. Members of healthy teams sometimes look for ways they can become dependent on other team members. "I could do this myself, but Joe is especially good at debugging assembly language code. I'll wait until he comes back from lunch and then ask him for help."

On the most effective one-project teams that I've been on, the beginning of the project is characterized by an unusual measure of tentativeness. Team members might feel that they have specific strengths to offer the team, but they are not pushy about asserting their rights to occupy specific roles. Through a series of tacit negotiations, team members gradually take on roles that are not just best for them individually but that are best for the team as a whole. In this way, everyone gravitates toward productive positions, and no one feels left out.

Effective Communication

CROSS-REFERENCE
For more on the role of communication in teamwork, see "Effective communication" in Section 13.1.

Members of cohesive teams stay in touch with each other constantly. They are careful to see that everyone understands when they speak, and their communication is aided by the fact that they share a common vision and sense of identity. Amish barn raisers communicate efficiently during a barn raising because they live in a tight-knit community and nearly all of them have been through barn raisings before. They are able to communicate precise meanings with a few words or gestures because they have already established a baseline of mutual understanding.

Team members express what they are truly feeling, even when it's uncomfortable. Sometimes team members have to present bad news. "My part of the project is going to take 2 weeks longer than I originally estimated." In an environment characterized by interdependence and trust, project members can broach uncomfortable subjects when they first notice them, when there's still time to take effective corrective action. The alternative is covering up mistakes until they become too serious to overlook, which is deadly to a rapid-development effort.

Sense of Autonomy

Effective teams have a sense that they are free to do whatever is necessary to make the project succeed. One reason that skunkworks projects work as well as they do is that they give team members a chance to do what's right without worrying about doing what appears to be right. They can work

without interference. The team might make a few mistakes—but the motivational benefit will more than offset the mistakes.

This sense of autonomy is related to the level of trust they feel from their manager. It is imperative that the manager trust the team. That means not micromanaging the team, second-guessing it, or overriding it on tough decisions. Any manager will support a team when the team is clearly right—but that's not trust. When a manager supports the team when it looks like it might be wrong—*that's trust.*

Sense of Empowerment

An effective team needs to feel empowered to take whatever actions are needed to succeed. The organization doesn't merely allow them to do what they think is right, it supports them in doing it. An empowered team knows that it can, as they say at Apple Computer, *push back* against the organization when it feels the organization is asking for something unreasonable or is headed in the wrong direction.

One common way that teams are denied empowerment is in the purchase of minor items they need to be effective. I worked on an aerospace project in which it took 6 months to get the approval to buy two scientific hand-held calculators. This was on a project whose mission was to analyze scientific data!

As Robert Townsend says, "Don't underestimate the morale value of letting your people 'waste' some money" (Townsend 1970). The most extreme example I know of was an episode during the development of Windows 95. To ensure that Windows 95 worked well with every program, the project manager and the rest of the team headed over to the local software store and loaded up a pickup truck with one of every kind of program available. The total tab was about $15,000, but the project manager said that the benefit to morale was unbelievable. (The benefit to morale at the software store wasn't bad, either.)

Small Team Size

Some experts say that you must have fewer than 8 to 10 people for a team to jell (Emery and Emery 1975, Bayer and Highsmith 1994). If you can keep the group to that size, do so. If your project requires you to have more than 10 project members, try to break the project into multiple teams, each of which has 10 or fewer members.

The 10-person limitation applies mainly to single-project teams. If you can keep a team together across several projects, you can expand the size of the team as long as the team shares a deep-rooted culture. The Amish farmers formed a cohesive team of several dozen people, but they had been together for generations.

On the other end of the scale, it is possible for a group to be too small to form a team. Emery and Emery point out that with less than four members, a group has a hard time forming a group identity, and the group will be dominated by interpersonal relationships rather than a sense of group responsibility (Emery and Emery 1975).

High Level of Enjoyment

CROSS-REFERENCE
For more on what motivates developers, see Section 11.1, "Typical Developer Motivations."

Not every enjoyable team is productive, but most productive teams are enjoyable. There are several reasons for this. First, developers like to be productive. If their team supports their desire to be productive, they enjoy that. Second, people naturally spend more time doing things that they enjoy than doing things that they don't enjoy, and if they spend more time at it, they'll get more done. Third, part of what makes a team jell is adopting a group sense of humor. DeMarco and Lister describe a jelled group in which all the members thought that chickens and lips were funny (DeMarco and Lister 1987). Chickens with lips were especially funny. The group actually rejected a well-qualified candidate because they didn't think he would find chickens with lips amusing. I don't happen to think that chickens with lips are funny, but I know what DeMarco and Lister are talking about. One group I was a part of thought that cream soda was hilarious and another thought that grape Lifesavers were a riot. There's nothing intrinsically comical about cream soda or grape Lifesavers, but those jokes were part of what gave those teams their identities. I haven't personally seen a cohesive team that didn't have a keen sense of humor. That might just be a quirk of my specific experience, but I don't think so.

How to Manage a High-Performance Team

CROSS-REFERENCE
For the difference between managers and team leaders, see Section 13.3, "Managers and Technical Leads."

A cohesive team creates an "us" and the manager is in the sticky position of being not completely "us" and not completely "them." Some managers find that kind of team unity threatening. Other managers find it exhilarating. By taking on a great deal of autonomy and responsibility, a high-performance team can relieve a manager of many of the usual management duties.

FURTHER READING
For excellent discussions of each of these points, see *Quality Software Management, Volume 3: Congruent Action* (Weinberg 1994).

Here are some keys to success in managing a cohesive team:

- Establish a vision. The vision is all-important, and it is up to the manager and team leader to put it into play.
- Create change. The manager recognizes that there is a difference between the way things should be and the way they are now. Realize that the vision requires change, and make the change happen.
- Manage the team as a team. Make the team responsible for its actions rather than making individuals on the team responsible for their individual actions. Team members often set higher standards for themselves than their leaders do (Larson and LaFasto 1989).

- Delegate tasks to the team in a way that is challenging, clear, and supportive. Unleash the energy and talents of the team members.
- Leave details of how to do the task to the team, possibly including the assignment of individual work responsibilities.
- When a team isn't functioning well, think about the MOI model, which states that most team problems arise from Motivation, Organization, or Information. Try to remove roadblocks related to these three factors.

12.4 Why Teams Fail

FURTHER READING
For an excellent discussion of team failure, see Chapter 20, "Teamicide," in *Peopleware* (DeMarco and Lister 1987).

The cohesiveness of a group depends on the total field of forces that act on that group. As with other aspects of rapid development, you have to do a lot of things right to succeed, but you only have to do one thing wrong to fail. Teams don't need to have all the characteristics described in the previous section, but they do need to have most of them.

Teams can fail for any of the reasons listed in Section 11.4, "Morale Killers." Those morale killers can keep a team from jelling just as easily as they can undercut individual morale.

Here are some other reasons that teams fail.

Lack of common vision. Teams rarely form without a common vision. Organizations sometimes prevent teams from forming by undercutting their visions. A team might form around the vision of producing "the best word processor in the world." That vision takes a beating if the organization later decides that the word processor doesn't have to be world class, but it does have to be completed within the next 3 months. When the vision takes a beating, the team takes a beating too.

Lack of identity. Teams can fail because they don't establish a team identity. The team members might be willing, but no one plays the role of Supporter, and without anyone to look after the team, the team doesn't form. This risk is particularly strong on rapid-development projects because of pressure not to "waste time" on "nonproductive" activities such as developing a team logo or shared sense of humor. Each team needs to have someone who will take responsibility for maintaining the health of the team.

Teams can also lack identity because one or more members would rather work alone than be part of a team. Some people aren't joiners, and some people think the whole idea of teams is silly. Sometimes a group is composed of 9-to-5ers who don't want to make the commitment to their jobs that team membership entails. There are lots of appropriate places for people who work like this, but their presence can be deadly to team formation.

Lack of recognition. Sometimes project members have been part of a project team that gave its heart and soul—only to find that its efforts weren't appreciated. One young woman I know worked practically nonstop for 3 months to meet a deadline. When her product shipped, the manager thanked her in a fatherly way and gave her a stuffed animal. She thought the gesture was patronizing, and she was livid. I wouldn't blame her for not signing up for another all-out team project. If an organization wants to create a high-performance team more than once, it should be sure to recognize the extraordinary efforts of the first team appropriately. If group members' previous experience has conditioned them to ask, "What's in it for me?" you'll have an uphill battle in getting a high-performance team to form.

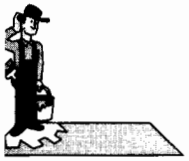
Productivity roadblocks. Sometimes teams fail because they feel that they can't be productive. People can't survive if the environment doesn't contain enough oxygen, and teams can't survive if they're prevented from getting their work done. Some experts say that the primary function of a software-project manager is to remove barriers to productivity so that the naturally self-motivated developers can be productive (DeMarco and Lister 1987).

Ineffective communication. Teams won't form if they can't communicate regularly. Common barriers to communication include lack of voicemail, lack of email, insufficient number of conference rooms, and separation of the team members into geographically dispersed sites. Bill Gates has pointed out that doing all of Microsoft's new-product development on one site is a major advantage because whenever interdependencies exist, you can talk about them face to face (Cusumano and Selby 1995).

Lack of trust. Lack of trust can kill a team's morale as quickly as any other factor. One reason that teams usually don't form within bureaucratic organizations is that the organizations (to varying extents) are based on lack of trust. You've probably heard something similar to this: "We caught someone buying an extra pack of 3-by-5 cards in August of 1952, so now all purchases have to go through central purchasing." The lack of trust for employees is often institutionalized.

Managers who pay more attention to how their teams go about administrative details than to the results they achieve are demonstrating a lack of trust. Managers who micromanage their team's activities, who don't allow them to meet with their customers, or who give them phony deadlines are giving a clear signal that they don't trust them.

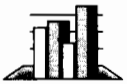
Instead of micromanaging a project team, set up a high-level project charter. Let the team run within that charter. Set it up so that management can't overrule the team unless they've gone against their charter.



CLASSIC MISTAKE

Problem personnel. The software field is littered with stories of developers who are uncooperative in legendary proportions. I worked with one belligerent developer who said things like, "OK, Mr. Smarty Pants Programmer, if you're so great, how come I just found a bug in your code?" Some programmers browbeat their co-workers into using their design approaches. Their nonconfrontational co-workers would rather acquiesce to their design demands than prolong their interactions with them. I know of one developer who was so difficult to work with that the human resources department had to be brought in to resolve module-design disputes.

If you tolerate even one developer whom the other developers think is a problem, you'll hurt the morale of the good developers. You are implying that not only do you expect your team members to give their all; you expect them to do it when their co-workers are working against them.



HARD DATA

In a review of 32 management teams, Larson and LaFasto found that the most consistent and intense complaint from team members was that their team leaders were unwilling to confront and resolve problems associated with poor performance by individual team members (Larson and LaFasto 1989). They report that, "[m]ore than any other single aspect of team leadership, members are disturbed by leaders who are unwilling to deal directly and effectively with self-serving or noncontributing team members." They go on to say that this is a significant management blind spot because managers nearly always think their teams are running more smoothly than their team members do.

Problem personnel are easy to identify if you know what to look for:

- They cover up their ignorance rather than trying to learn from their teammates. "I don't know how to explain my design; I just know that it works" or "My code is too complicated to test." (These are both actual quotes.)
- They have an excessive desire for privacy. "I don't need anyone to review my code."
- They are territorial. "No one else can fix the bugs in my code. I'm too busy to fix them now, but I'll get to them next week."
- They grumble about team decisions and continue to revisit old discussions after the team has moved on. "I still think we ought to go back and change the design we were talking about last month. The one we picked isn't going to work."
- Other team members all make wisecracks or complain about the same person. Software developers often won't complain directly, so you have to ask if there's a problem when you hear many wisecracks.

- They don't pitch in on team activities. On one project I worked on, 2 days before our first major deadline a developer asked for the next day off. The reason? He wanted to spend the day at a men's-clothing sale in a nearby city—a clear sign that he hadn't integrated with the team.

Coaching the problem person on how to work as part of a team sometimes works, but it's usually better to leave the coaching to the team than to try to do it as the team leader or manager. You might have to coach the team on how to coach the problem team member.

If coaching doesn't produce results quickly, don't be afraid to fire a person who doesn't have the best interests of the team at heart. Here are three solid reasons:

- It's rare to see a major problem caused by lack of skill. It's nearly always attitude, and attitudes are hard to change.
- The longer you keep a disruptive person around, the more legitimacy that person will gain through casual contacts with other groups and managers, a growing base of code that person has to maintain, and so on.
- Some managers say that they have never regretted firing anyone. They've only regretted not doing it sooner.

You might worry about losing ground if you replace a team member, but on a project of almost any size, you'll more than make up for the lost ground by eliminating a person who's working against the rest of the team. Cut your losses, and improve the rest of your team's morale.

12.5 Long-Term Teambuilding

The team of Amish farmers is a good model of the perfect, jelled team. But that team didn't form overnight. Those farmers had been together for years, and their families had been together for years before that. You can't expect performance as dramatic as raising a barn in a single day from a temporary team. That kind of productivity comes only from permanent teams.

Here are some reasons to keep teams together permanently.

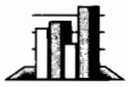
Higher productivity. With a permanent-team strategy, you keep a group together if it jells into a team, and you disband it if it doesn't. Rather than breaking up every team and rolling the dice on every new project to see whether new teams jell or not, you roll the dice only after you've lost. You stockpile your winnings by keeping the productive teams together. The net effect is an "averaging up" of the level of performance in your organization.

Lower startup costs. The startup costs for building a team are unavoidable, so why not try to reuse the team and avoid additional startup costs? By keeping the effective teams together, you preserve some of the vision, team identity, communication, trust, and reservoir of good will built up from completing an enjoyable project together. You're also likely to preserve specific technical practices and knowledge of specific tools within a group.

Lower risk of personnel problems. Personnel issues arising from people who work poorly together cost your projects time and money. You can avoid these problems altogether by keeping teams together when they jell.

HARD DATA

Less turnover. The current annual turnover rate is about 35 percent among computer people (Thomsett 1990). DeMarco and Lister estimate that 20 percent of the average company's total labor expense is turnover cost (DeMarco and Lister 1987). An internal Australian Bureau of Statistics estimate placed the average time lost by a project team member's resignation at 6 weeks (Thomsett 1990). Studies by M. Cherlin and by the Butler Cox Foundation estimate the cost of replacing an experienced computer person at anywhere from $20,000 to $100,000 (Thomsett 1990).



HARD DATA

Costs are not limited simply to the loss of the employee. Productivity suffers generally. A study of 41 projects at Dupont found that projects with low turnover had 65 percent higher productivity than projects with high turnover (Martin 1991).

Not surprisingly, people who have formed into cohesive teams are less likely to leave a company than people who have not (Lakhanpal 1993). Why should they leave? They have found an environment they enjoy and can feel productive in.

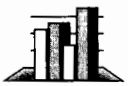
The idleness question. Organizations are sometimes leery of keeping teams together because they might have to pay a team to sit idle until a project comes along that's appropriate for them to work on. That's a valid objection, but I think that in most organizations it isn't ultimately a strong objection.

Organizations that look exclusively at the cost of idle time overlook the costs of rebuilding teams for each new project. The cost of building a new team includes the cost of assembling the team and of training the team to work together.

Organizations tend to overlook how much they lose by breaking up a high-performance team. They take a chance that individuals who could be working as part of a high-performance team will instead become part of an average team or a poor one.

Some organizations worry that if they keep teams together, they won't be able to get any teams to work on certain projects. But others have found that if you give people the chance to work with other people they like, they'll work on just about any project (DeMarco and Lister 1987).

Finally, I have yet to see a software organization that has long idle periods. To the contrary, every project that I've ever worked on has started late because personnel weren't available to staff it until their previous projects were completed.



HARD DATA

Peopleware issues tend to lose out in the bean-counter calculations because in the past they have lacked the quantitative support that staff-days-spent-idle has. But the situation has changed. Australian software consultant Rob Thomsett has shown that there is a tremendous return on investment from teambuilding—for example, it is an order of magnitude better than for CASE tools (Constantine 1995a). We now know that among groups of people with equivalent skills, the most productive teams are 2 to 3 times as productive as the least productive teams. They're 1½ to 2 times as productive as the average teams. If you have a team that you know is on the top end of that range, you would be smart to allow them to sit idle for up to one-third or even one-half of their work lives just to avoid the risk of breaking them up and substituting a merely average team in their place.

12.6 Summary of Teamwork Guidelines

Larson and LaFasto distilled the results of their research into a set of practical guidelines for team leaders and team members. If your team wants to adopt a set of rules, the guidelines in Table 12-1 on the facing page are a good place to start.

Case Study 12-4. A Second High-Performance Team

Frank O'Grady captured the intense efficiency that a jelled team can have:

"I would sit in on design meetings, amazed at what I was seeing. When they were on a roll, it was as if they were all in some kind of high-energy trance during which they could see in their mind's eye how the program would unfold through time. They spoke in rapid-fire shorthand, often accompanied by vivid hand gestures when they wanted to emphasize a point. After 15 minutes or so, a consensus was reached as to what had to be done. Everyone knew which programs had to be changed and recompiled. The meeting adjourned." (O'Grady 1990)

Table 12-1. Practical Guidelines for Team Members and Leaders

Team Leader	Team Members
As a team leader, I will:	As a team member, I will:
1. Avoid compromising the team's objective with political issues.	1. Demonstrate a realistic understanding of my role and accountabilities.
2. Exhibit personal commitment to the team's goal.	2. Demonstrate objective and fact-based judgments.
3. Not dilute the team's efforts with too many priorities.	3. Collaborate effectively with other team members.
4. Be fair and impartial toward all team members.	4. Make the team goal a higher priority than any personal objective.
5. Be willing to confront and resolve issues associated with inadequate performance by team members.	5. Demonstrate a willingness to devote whatever effort is necessary to achieve team success.
6. Be open to new ideas and information from team members.	6. Be willing to share information, perceptions, and feedback appropriately.
	7. Provide help to other team members when needed and appropriate.
	8. Demonstrate high standards of excellence.
	9. Stand behind and support team decisions.
	10. Demonstrate courage of conviction by directly confronting important issues.
	11. Demonstrate leadership in ways that contribute to the team's success.
	12. Respond constructively to feedback from others.

Source: Adapted from *TeamWork* (Larson and LaFasto 1989).

Further Reading

Here are three books and articles about teambuilding in software:

DeMarco, Tom, and Timothy Lister. *Peopleware: Productive Projects and Teams*. New York: Dorset House, 1987. Part IV of this book focuses on growing productive software teams. It's entertaining reading, and it provides memorable stories about teams that worked and teams that didn't.

Weinberg, Gerald M. *Quality Software Management, Volume 3: Congruent Action*. New York: Dorset House, 1994. Part IV of this book is on managing software teams. Weinberg's treatment of the topic is a little more systematic, a little more thorough, and just as entertaining as *Peopleware*'s. Parts I through III of his book lay the foundation for managing yourself and people who work in teams.

Constantine, Larry L. *Constantine on Peopleware*. Englewood Cliffs, N.J.: Yourdon Press, 1995. Constantine brings his expertise in software development and family counseling to bear on topics related to effective software teams.

Here are some sources of information on teambuilding in general:

Larson, Carl E., and Frank M. J. LaFasto. *Teamwork: What Must Go Right; What Can Go Wrong*. Newbury Park, Calif: Sage, 1989. This remarkable book describes what makes effective teams work. The authors conducted a 3-year study of 75 effective teams and distilled the results into eight principles, each of which is described in its own chapter. At 140 pages, this book is short, practical, easy to read, and informative.

Katzenbach, Jon, and Douglas Smith. *The Wisdom of Teams*. Boston: Harvard Business School Press, 1993. This is a full-scale treatment of teams in general rather than just software teams. It's a good alternative to Larson and LaFasto's book.

Dyer, William G. *Teambuilding*. Reading, Mass: Addison-Wesley, 1987. This book describes more of the nuts and bolts of teambuilding than Larson and LaFasto's book does. Whereas Larson and LaFasto's intended audience seems to be the leader of the team, this book's intended audience seems to be the leader of a teambuilding workshop. It makes a nice complement to either Larson and LaFasto's book or Katzenbach and Smith's.

Witness. Paramount Pictures. Produced by Edward S. Feldman and directed by Peter Weir, 1985. The Amish barn-raising scene is about 70 minutes into this love-story/thriller, which received Oscars for best original screenplay and best editing and was nominated for best picture, best direction, best actor, best cinematography, best art direction, and best original score.