,ch16.19202 Page 199 Thursday, February 12, 2009 3:09 PM

CHAPTER SIXTEEN

Steve McConnell on Better Practices

Steve McConnell has influenced both of us for as long as we've been working in software. He was one of the first people either of us read who was able to write about complex software engineering and quality topics in an easy-to-read, conversational tone. More than anyone else, Steve has been a huge influence on our own writing, both in our topics and in the way we choose to express ourselves. In some ways, it's Steve who showed us that it's possible to write a serious book that sounds like you're just talking to someone casually.

Scott Berkun was our other great influence, since if it weren't for him we wouldn't be writing for O'Reilly today. Scott invited Andrew to be a technical reviewer for his first (and excellent) book, Making Things Happen, which is where our relationship with O'Reilly started. (He was an enormously valuable reviewer for our own Applied Software Project Management.) So when Scott offered to interview Steve for Beautiful Teams, we jumped at the chance to have these two people whose work we admire sit down and talk about teams.

Scott: So the first thing I want to ask you is does the phrase "a beautiful team" mean anything to you? Is that a phrase you ever used or heard used before, and in relation to software products you've worked on or teams you've been on or managed?

Steve: I tend to be pretty literal, and so the idea of a beautiful team doesn't resonate all that well with me. When I hear "beautiful team" I assume people are talking about something different, and maybe they're talking about a high-performance team or a high-quality team or something like that.

Scott: Or a team of supermodels.

Steve: *(Laughter.)* Which is going to be hard to come by in software. So yeah, the term itself doesn't automatically make me think of anything in particular. I guess it makes me think of a team that's highly functional, has highly cooperative, collaborative, working relationships where everybody on the team feels like they are part of something elite, where the team works on something that ends up being successful.

I don't think it's good enough for the team to just feel like it's doing something special. I think whatever business context they're in it actually has to result in something that the business or the market agrees is special before I really say that that was a beautiful team.

Scott: Whenever I think of high-performance teams, beautiful teams, whatever adjectives you want to use, the core question, the first thing I think about, is whether we're talking about the means or the ends. Is it something that you can observe only while the team is working, or is it something you can evaluate only after they're done?

Steve: Well, for some reason, my mind's kind of going back to the notion of quality, and in particular, Deming's definition of quality, which is conformance to requirements. And if I'm trying to put some shape to the idea of a beautiful team, I naturally am thinking, "Well, OK, a beautiful team maybe is a team that conforms to requirements."

And the requirements can vary an awful lot from one context to another for a team. Some contexts you might really need a high degree of innovation. Another context, you might need a really high degree of operational efficiency. Other contexts, you might just need an extraordinary level of focus.

In many contexts, I think one of the implied requirements that get short-changed is sustainability, where a team does something great one time but they're too burnt out to do something great the next time. And I think whether that's good or bad just depends on what the organizational requirements are.

There certainly are cases where the organization needs to get a great product out in a short amount of time, and if the team is burnt out, that's just the cost of succeeding at that time, but there are other organizations that are in more of a sustaining mode, and in those organizations, I think, if they're burning out teams, then they're really jeopardizing their long-term viability.

Scott: What's the best software team that you ever worked on, and why?

Steve: One of the best teams I ever worked on wasn't even really a team. It was just more of a workgroup, and I say it wasn't a team because we weren't really all working toward any common objective. We all worked for the same company, but we were working on different projects for different clients.

But it was a very fun working environment. What made that a fun team—I don't know if it was a beautiful team—was that we were all young, we were all in the same life circumstance. We all had similar senses of humor. None of us had any real significant outside attachments.

So we were able to spend time together outside of work as well as in work. And we all had a common enemy, namely our boss, and I think that we felt like we were getting an awful lot done. I mean, we were getting a lot done, we felt, in spite of our boss, maybe not because of him. And was that a team? I don't know. It was a cohesive workgroup, but we were really aiming at the same thing.

Scott: So you weren't actually working on the same project at the same time?

Steve: We were all doing different projects for different clients, but we happened to all be in the same room. Our work overlapped a lot. We would switch off who was working for which client and who was working for which project, and so there was a pretty high degree of collaboration across the small projects that we were working on. So I don't know; it's hard to have a conversation like this without getting all nostalgic about the good old days.

Scott: Nostalgia's fine. I mean, that's part of the story here. There are very few people who will read this book who are aware at the moment that they are currently on the best team that they'll ever work on. We always think of the best as something in the past, and even if the team we're on right now is the best, we won't fully realize it until after we've moved on.

Steve: I agree, and that actually matches the experience I was just describing quite well. Much of the time I was in that circumstance I was longing for the group of people that I had hung out with in high school. It wasn't until I had been in that circumstance for maybe a year, where one day I woke up and realized, hey, you know, this is actually pretty cool, too.

And I'm not sure what it is about the way my mind or emotions work that would cause me to not pay as much attention to what I'm working on now as to what I was working on before, but that seemed to be the way it was in that case.

I worked on a graduate school project that was also interesting.

Scott: What was the goal of the project?

Steve: Well, one thing that made it a good team was that we were quite explicit about what the graduate school's goal was versus what our goal was. The graduate school's stated goal was to deliver some software to a customer. Our real goal was to graduate on time.

And we were always very clear within our team about the distinction between what the school thought the goal was versus what we, as team members, were very clear our goal was.

Scott: Interesting; the beautiful and subversive team.

Steve: We actively managed our project to try to achieve our real goal, which was to graduate on time, and the thing that was nice about that team was that we started out with most of us not knowing most of the other people on the team at the beginning of the year.

And we went through an unusually long period of just kind of feeling each other out to see who was good at what, who wanted to do what, who was sensitive about what, and so on.

At the time, I was impatient about it, but once the project was underway, I recognized it had been very healthy. There was enough flexibility from each person on the team that we were all willing to give where we needed.

And so everybody worked on stuff they thought was interesting, and didn't get over-committed. And as a result, I think the team was quite resilient. I mean, we had one guy whose wife was pregnant, and they actually lost the baby. We had another team member who went through a divorce. We had a couple team members who were traveling often, and even with some pretty significant trauma going on, the team held together fine. Everybody pitched in where needed.

All the more remarkable, I think, was that this wasn't our main job.

We had real jobs that were demanding our attention, too. So that definitely stands out in my mind as a team that functioned at a very high level as a team. The whole was greater than the sum of the parts in many ways.

Scott: This is an unfair question, because I'm going to ask you to give a number to something that's abstract, but I'm asking anyway. What percentage of teams do you think are high-performing teams?

Steve: Oh, I think true high-performing teams are fairly rare.

Scott: Is that 10%? Five? One?

Steve: Based on the hundreds of companies we've worked with over the years, I would say it has to be less than 10%. I mean, some of it is by definition. When you say high-performance team, you are implicitly saying this is a team that stands out from an average-performing team. It becomes a question of how many teams do we see that really stand out as being markedly different from just the rank and file teams that we work with. And every once in awhile we see a team that stands out in some way or other, but I'd say 10% or less.

Scott: OK, 10%. Do you think that there are unique attributes about those teams that are difficult to replicate, or is there some way to copy them? Let's say I run a company of 100 and have 10 teams within that 100. And I recognize what high-performing teams are and believe I have one. The question is: how can I help the other nine teams to be more like the high-performing team? Or is it impossible, since what makes a great team is tightly wrapped up in the unique relationships between the people involved?

Steve: Well, this may be controversial, but I don't think that there's anything mysterious about creating a high-performance team. And I realize that can be interpreted as maybe an overly confident statement, but I think I have a basis for saying that.

My company has received awards the last two years for being the best small company in Washington to work for. And we do that because of having really high staff morale, which has been created through the kinds of practices that I'm here to talk about.

If you have people who are selfish and they only want to do what they want to do, it will be very tough to establish a high-performance team. You may have an assemblage of individuals, but that's all you're going to have. So there has to be goodwill and flexibility. Maybe not even a super high amount, but some.

If you have some goodwill and flexibility, then I think there's a kind of template that you can follow. First, the team leader needs to establish an elevating vision or goal, and I think that this is truly a leadership function, not just management. It goes beyond the management function. An awful lot of thought should go in on the part of the team leaders into what is the team's goal and what is it about it that is not just trying to trick the team into thinking they have a goal that's elevating or inspiring, but to come up with one that truly is.

A classic example is if you're out digging ditches, that's not very elevating or inspiring. But if you're digging ditches to protect your town that's about to be attacked by an enemy, well, that's more inspiring, even though it's the same activity. And so the leader's job, really, is to try to determine or frame the activity in such a way that people can understand what the value is.

So some amount of goodwill, an inspiring, elevating vision or goal. The next step is lots and lots of communication in all directions—between the team members and the leader and between the leader and the team members. And I think facilitating communication within the team in all possible directions is good, too.

Next, I think it's useful to establish a sense of team identity, something I've seen Microsoft, in particular, do really, really well over the years with team members getting team T-shirts and going on team trips and getting team awards and so on, just to foster that sense of team identity. Whether it's mugs, posters, T-shirts, jackets, beach towels—it doesn't matter. If you do this you get two benefits. One is team members feel appreciated, which is just good for morale, in general. Two, if you do it right, they can actually develop a sense of team identity, which is very positive, too.

And then there are group activities. I mentioned fostering informal interaction, but I would also focus on group bonding activities. I have seen teams where they joke about doing bonding activities, but they still enjoy them. And I think everybody knows what you're trying to do, but the fact is a lot of companies refuse to spend the minimal amount of money it takes to actually do these activities.

So maybe to put it a different way: putting some work into trying to create the team is, in itself, helpful. So those are a handful of things that come to mind.

In the book *Peopleware*, Tom DeMarco and Tim Lister explicitly say they don't know how to create a team, but they know how to kill one. They talk about teamicide. And I've seen all the same things they describe—ways to keep teams from forming, or if they do form, how to disband them quickly, and companies do that all the time. They'll take a team that works really well together, and then they'll disband it and lose all the benefits.

Scott: Why do you think that happens so often? I think you'd agree that teams are critical. That the reason why that exceptional work happens is because of the team structure around people that enables really good work to happen. One complaint I always had during my years at Microsoft was that the rewards structure is entirely individually based. And so the performance structure, the way you're actually given bonuses and raises and promotions, is based on your individual performance.

Yet there is this genuine interest in building team identity, which is true. There are team events, team clothing, which can create bonding effects. But I never worked on a team or saw a team that said if we make our dates, everyone gets a 2% bonus, as a team. My gripe, my persistent complaint there, was that the reward model never matched the rhetoric model.

There was always the rhetoric of teams, but then when it's performance review time, oh, you were the best programmer, so you get the stock grant or whatever. Why do you think there's always a disconnect between the two?

Steve: Well, there's not always a big disconnect. We've worked with a handful of companies that actually do rewards at the team level, and they don't break it down further than that. Whether that actually works, I think, is an open question.

But I've seen the same thing that you described. And you know what you're talking about is what Gerald Weinberg would describe as a lack of congruence.

Scott: Yes.

Steve: The words say one thing, the actions say something else. And I think in general I agree with the idea that the more congruent you can be in your behavior, the better things will go.

Scott: Here's another idea. If these high-performing teams are rare, do you believe there are other things you can do to create them? They're not impossible, but they are uncommon. So once you have a high-performing team, what most software companies do, once that project has been completed, is disband the team. Those resources now need to be distributed to other projects. But the alternative model says the high-performing team is really the most valuable asset you have, suggesting you want to preserve that team, find a way to keep them together on whatever the new project is. And you sacrifice domain knowledge or optimal resourcing to protect the team. Have you ever seen anyone who put the team first in their management philosophy, as opposed to last, which is generally what I've seen? And aside from whether you've seen it or not, do you think it's a ridiculous idea?

Steve: It's funny, because I would say, contradictory to what I said a minute ago, is most companies value the domain expertise a lot. And one of the side effects of valuing the domain expertise so highly is they're reluctant to move people out of an area once a person has acquired experience in that area.

And so while they may reconfigure how people are working together, over time, once people start working in one area, we see the opposite problem more often. That is to say we see the problem of staff getting bored in an area and wanting new opportunities, and their company is reluctant to move them when they've made an investment in that particular area.

We see that more often than we see the problem of the company just scattering staff to the four corners, or to the four winds once a project is done, and losing all of that staff expertise. I guess part of what I'm thinking is that we don't necessarily see really high-performing teams working together all that often. I'd have to get a larger sample of high-performing teams before I could make any statements about what companies usually do with them.

Scott: All right. It's an unfair question. However, I think many people would agree, who've worked on good teams, that there's not enough protection for those teams. There's not enough reward for those teams to stay together and they're disbanded too often. I'm just thinking, what if they went to the other extreme: bet that an intact team that does well is the most important asset and make things work around them.

Steve: I think DeMarco and Lister have talked about this. Someone proposed the idea of allowing intact teams to bid on work within the company. And so basically, the notion was the team will say, "If you leave us together, we'll commit to get this project done in four months." And some other team might say, "Well, we're committed to get it done in five months." The idea is you take a nominal estimate for a non-intact team, and if you can get a better deal from a team by keeping it intact, and the team says, hey, it's so important to us to stay together that we're willing to work extra to come up to speed in the area or whatnot, then that could be an attractive model for a company.

To my mind, the Achilles' heel in that argument is that the team can say whatever it wants, but that doesn't mean they're necessarily going to deliver.

Scott: And you're also creating an environment where those teams are competing with each other for projects.

Steve: It has the potential. So I mean, I'm purely going on analysis here, because I can't point to any examples where I've seen this actually done, but it seems like it would have the potential to lead to a sweatshop model over time, and ultimately, you can kind of imagine a scenario in which a team works together once, has a great experience, commits to a meat grinder project on their second project, and then voluntarily disbands after that because they would prefer not to sign up for another meat grinder project.

Scott: Do you think it's possible for a high-performing team or a great team to make horrible software, or for a horrible team to make great software?

Steve: Well, so much of that just comes back to how you define a great team and a poor team. I think that you can have teams where everybody on the team is highly motivated and they're having a great time, but they're completely out of touch with business reality, and the people on the team might feel that that was a great team but the business does not feel that way.

Scott: This is sort of like your graduate student project in a way, right? That you can have a team that for their own goals feels like they've done a great job, but for the goals of their boss or their company, they've failed.

Steve: In one of the first consulting engagements we had, we were brought in by the president of the company who was a non-technical guy, and he said, "I acquired this company, and I've taken over active management of the company and I've got a release date. My technical guys keep telling me, 'Yeah, yeah, we're gonna make the release date,' but I've lost all confidence in their ability. And frankly, I think these guys are just entranced by how cool the technology is and don't care whether we ever ship at all."

And my reaction at the time was here's another classic business guy who has no sensitivity to his staff. I'm sure his staff is working really hard and he's just not being sensitive to how hard they're working. So we went in and conducted extensive interviews with the staff, and to a person, he was right. The stuff they were working on was special effects software for the movie industry. It was incredibly cool, and truly the staff didn't care whether they ever shipped anything or not. They had no sense of business urgency or of getting revenue-generating software out the door, and they were having an awesome time doing it. I think that is a great example of a case where they might have thought they were a great team, but the president clearly didn't think they were a great team, and that supposedly great team would eventually have run that company out of business.

Scott: So how about the opposite: can an underperforming team or horrible team—a team where everyone is unhappy and doesn't like each other—is it possible for them to make a great piece of software?

Steve: I think so. But it's because even a blind squirrel finds a nut every once in awhile. Morale and motivation usually go the same direction, but not always. One of the distinctions that was made clearly to me a long time ago was the distinction between motivation and morale. A lot of times people can conflate those two ideas, but you can have high motivation with low morale. And you can have high morale with low or misguided motivation.

You can look at sports analogies for the ins and outs of what teams work and what teams don't. One of the things that you can find lots of examples of in sports teams is that you can have high-performing teams where the team members love each other, and every once in awhile you see a high-performing team where the team members hate each other. It isn't common, but it does happen.

You can have high-performance teams where you've got really strong, standout individuals. You can have disastrous teams where you have some strong standout individuals. You can also have great teams where you've got no real standout individuals, but as a team, they work really, really well together.

And then you can have teams that are just plain mediocre. The individuals don't stand out and the team is mediocre. I think all those configurations can be found in software teams also.

Scott: So you studied computer science, right?

Steve: Yes.

Scott: How much of the curriculum—the projects you did, the assignments you had—had anything to do with learning about teams, or experiencing teams?

Steve: Well, so my undergraduate degree is in philosophy, and I had my—

Scott: There's not much teamwork in philosophy.

Steve: *(Laughs.)*

Scott: Unfortunately.

Steve: There is probably more than there was in computer science.

Scott: Oh, boy.

Steve: I got a minor in computer science and I don't know that I had any multiperson assignments. When I went to graduate school, one of my explicit goals was to get more experience and to get better at working in teams. And one of the things that was attractive about Seattle U's Master in Software Engineering program was that many of the assignments were team assignments. It wasn't just the final-year project, but many of the assignments, especially the bigger assignments, throughout most of the classes were done in teams.

Scott: And was that useful in helping you to understand how teams work?

Steve: It was, and now that you mention it, I think it would have been helpful if the program had actually spent some time talking about how to succeed on team assignments versus as individuals, but we stumbled on some good insights anyway. But I think it would have been possible for other groups of students to not stumble on those insights.

Scott: Your experience fits the experience I had a Carnegie Mellon University. There were projects when you reached junior level, junior year, that were team-based, but by and large, everyone I knew in the class formed teams with people they were already friends with. And there was no guidance provided on even things as simple as the five most likely kinds of conflicts you're going to have. It would take a half hour, a short class discussion on the five most common kinds of conflict and how to resolve them. But there was nothing. And the reason why I asked the question is there is definitely a gap here between how people are taught. We are taught you are an individual programmer and your goal is to be a master at the craft of programming yourself, yet once in the workforce they are put into environments that are exclusively team-based.

Steve: I think there's always a gap between the craft and the things you care about in your craft versus what you get taught academically in school. I agree that the example you gave would have been very helpful in my context. Five things that you can do to have a smooth team project; that would have been really helpful. Five ways you can mess it up, five things you can do to help it go smoothly. That would have been helpful. And I don't think that's unique to software. I was involved in a legal dispute a few years ago, and I went in and the attorney for the other side made some decisions that I thought were just incredible blunders. And even though they benefited me, I thought it was not doing their clients much of a service. And so I talked to a friend of mine who had gone to Harvard Law School, and said, what do you think about this? Is this uncommon that my attorney would pay attention and the other party's attorney wouldn't? And my friend's response was surprising.

He said, "Well, one of the things I would do before I went before any judge or commissioner, if I hadn't been in front of that judge before, I would call people and find out what that judge was like, and I would learn everything I could about that judge. I would learn what their hot buttons are. I would learn how they tended to rule in the past. I would look up their opinions that they've written on similar issues. So that when I go into the courtroom, I know how to expect that judge to behave." And I said, "Did you learn any of that when you were at Harvard Law School?" And he just laughed and said, "No, we didn't talk about anything this practical."

And yet, that ends up, as far as I could tell, being somewhat decisive in whether you're going to win or lose. I think any academic program has to strike a balance between theory and practice, and there is some level where we're probably just getting a little bit too practical. It's more like a trade school than an education if you get overly practical.

At the same time, you'd like to think that there are labs or studios or something where you could actually get some practical experience and maybe what we're saying is there's more of a need for some kind of guided studios in software than we often find.

Scott: I'm convinced there's an obligation colleges have. If you are putting students in an environment where they're experiencing something for the first time, and we know this replicates an experience they will have the rest of their careers, there's an obligation to at least say, "If you're having problems on your team, go to this website, go read this book or come to this workshop." In my experience—and again, my experience was almost 15 *years ago—there was nothing. There was not even a conversation about it. You could go to your TA (teaching assistant), but your TA was not there to listen to what they would see as your social problems. There was a machismo marine suck-it-up attitude about anything not related to code. It reinforced the idea that to be a good programmer was based exclusively on what you could convince a compiler to do, rather than what you could convince a person or team to do.*

I admit it's a separate skill set. Being good at programming and good at teamwork are unrelated: you can be great at one and suck at the other. But when I think of beautiful teams, that's where I go. I ask people questions like, what was the first team you were on, how did you learn, how did you have any idea what a good team is like. They have to at least have an idea of how it is supposed to be, and some notions of what behavior are more likely to make the team work well. The fact that most CS graduates are highly ignorant of these things is a travesty. It's irresponsible, given the nature of how most professional programming is done.

OK. I feel better now. This does all lead to a more practical question. Do you think that certain software development methods promote greater teamwork than others?

Steve: Well, definitely. There are certain practices I think contribute to high-performing or effective teams.

In Scrum, for example, I think that communication you get with the daily standup meeting is helpful. There are an awful lot of teams where people do not see each other face to face every day. In addition, the sprint planning, which is typically done at the whole team level, is a chance for the team to get together in an in-depth way, in a task-focused way, approximately once a month. These activities can be used on essentially any kind of project, but they get done on Scrum more often because they're built into that particular approach.

I think there's a point of diminishing returns with in-person communication in programming. In Extreme Programming, there was a heavy emphasis on pair programming. We see very few companies that have retained any emphasis on pair programming. We see organizations that make selective use of pairing, but if anything, it's just more communication and more interaction than most programmers want.

One of the common developments we've seen is an almost universal presence of multisite teams. I think that there are various things you can do to make the best of teams that are distributed across different time zones, but that's exactly what it is. It's making the best of it. We almost never see a case where the multiple sites end up providing any kind of a net advantage. If it's done really, really well, then it's not that much of a net loss.

Scott: So were any of the high-performing teams that you've seen virtual teams? And by virtual I mean that the majority of the team does not see each other face to face.

Steve: I can't think of any. And then the other question is what if the team is split across two or three different sites or two or three different time zones? And I think that you can do reasonably well with two sites. But once you get above two sites, it gets much more difficult to do well. You risk having extremely low efficiency and lots and lots of communication mistakes and tons of rework that arise from communication problems.

Scott: I asked you before about the best team experience that you had. What was the worst team experience you've ever had? I'm hoping for a horror story to balance out the good ones.

Steve: I think I've been lucky in my career. I can't think of any teams where I thought the team was a bad team. I can think of lots where I felt the team was not an exceptional team, but I can't think of any where I thought the whole team was just bad. I've had a few experiences where there were one or two individuals on a team that really detracted from my personal enjoyment from working on the team, but does that make it a bad team?

Well, maybe the interesting question is, can one really bad team member ruin a team? And in my experience, I would say yes. It depends how bad the person is, but if the person is obnoxious enough, then I think it can be pretty bad. I worked in a group earlier in my career, in which there was one team member who was extremely territorial about the team documentation.

And so if you went and got the design notebook that had the design for a particular section of code you were working on, even if you were scheduled to be working on that code for two weeks, she would come at lunchtime and grab the notebook off your desk and put it back in her area. *(Laughter.)* And so then you'd have to go ask her for it again.

She wasn't using it. She just wanted to be really sure that it didn't stay in your area longer than it needed to. And when you're working on something for a couple of weeks, and you have to go get the notebook 20 times when really you should have had to go get it only once, that sort of thing can get irritating. And of course, that's just the tip of the iceberg.

Scott: Well, here's a more specific situation. If you're on a team that's functioning OK, but there's one person that you believe is causing the most problems, they are creating the most turmoil, are the most difficult to work with, the most sensitive to criticism, but the challenge is they happen to be the most talented engineer you have on the team—what do you do?

Steve: Well, Gerald Weinberg, in his *Psychology of Computer Programming* book, made the blanket statement that if you have a programmer that's indispensable, you should fire him.

And I read that first in 1989, and when I first read that, I thought that was too extreme. But as years have gone by, I have come to see the wisdom in that advice. In some organizations, firing may be too harsh. But in some way get them off the team, or at least get them off the critical path. And my reason for that is the longer you let them work in that mode, the bigger the risk becomes of them leaving.

It is just bad risk management to have a single point of failure on a project, and the lowest point that your risk is ever going to be with somebody like that is *today*. It doesn't get lower by waiting. It just gets higher. And so the sooner you can take some kind of action to bring that risk down, the better off you'll be.

I've also usually found that that person who seems indispensable usually isn't. Usually there's somebody who's quietly doing their job who's actually a much stronger contributor than the difficult, visible, uncooperative person. And I'm sure that there are exceptions, but I think the first-order approximation is that that person is probably not as good as you think they are. So number one, they represent a super-high risk to your project. Number two, they're probably not even that good.

Number three, they're a drain on everybody else on the team, and so the manager in me says, try to find some way to coach them to be better, and if that doesn't work, try to find a way to coach them out of the organization. The business owner in me says get rid of them as soon as you can, because they're just costing you.

There was an interesting insight in Larson and Lafasto's study on high-performance teams where they described a survey of team members and of managers, where managers said that they felt that one of their strengths was dealing with problem team members, whereas team members said they felt that one of their managers' greatest weaknesses tended to be dealing with problem team members too slowly.

I think what this says is it says that there's a pretty big gap in perception between how effective managers think they are in dealing with these issues and how effective they really are, and I think in general, when the manager finally takes action on a problem team member, the team's reaction is typically not "Oh, that's so unfair." The team's reaction is typically "What took you so long?"

Scott: How can managers avoid falling into this gap between perception and reality?

Steve: Working at Construx for the last 12 and a half years, we have really explicit company values, and two of our values are openness and accountability, and we have pushed those principles a lot further than I have seen at any other company. And after living those values for the last 12 and a half years here I simply do not understand how you can run a company or how you can run a team without having very strong support for both openness and accountability.

And the reason I mention those as a pair is I don't think you can have one without the other. You can't have real accountability if you don't have openness about who's doing what and how they're performing, and whether they're hitting their dates, and whether their quality level is good. If that's all sort of kept secret, then it's just too easy to not hold people accountable.

You have to throw it out in the open. Otherwise, it's too subject to gaming and you don't get input on whether you've really got the right information, all that kind of stuff. Likewise, if you have just the openness but you don't have the accountability, then people think you're incredibly inept, or that you're just asleep at the switch, because you're seeing that some people are doing well and some people keep missing their dates, and you're not doing anything about it. So if you don't have accountability there, then the openness actually becomes a problem rather than an asset.

And just in terms of being able to manage workflow and address personnel issues in a timely way, and address all the zillions of little teamwork issues that come up, if you're constantly saying, well, should we expose this item or should we tell so-and-so about this, it just becomes this incredible drag on your efficiency as you're spending half your time talking about who should know what and how are we going to protect so-and-so's feelings and that kind of thing.

Whereas if you just throw everything out on the table, it makes it really easy, and I know that's not the only possible way to run an effective organization, but for my personality, I just cannot imagine going through all the overhead of doing it any other way. I don't know how you can have an effective organization without really strong accountability. And I think most organizations give lip service to the accountability word, and don't have anything that would look to me like real accountability.

Scott: What you're saying is terrifying to most people, especially to people in power. Whoever's in charge has the greatest to fear in being open and accountable, because they have the largest number of people who can challenge their authority.

Very few people, in my experience, who seek out power, are simultaneously interested in being as open and as accountable to the people that they want power over. It's rare. It's not impossible. There have been managers and executives I've worked for that I feel embody the same thing you're talking about, but it's rare. I think it's probably rarer than high-performing teams.

Steve: Yeah, I just don't know how else you could do it. The interesting thing about a software organization is you're basically talking about directing how people are thinking. That is what we pay people to do. We pay them to think, and you cannot stand over somebody with a whip and tell them what to think.

You have to make them want to think the things that they're paid to think. And there's a very real sense in which you give over your soul as a programmer in ways that you don't give over your soul if you were operating a piece of heavy machinery or standing behind a counter, talking to people. Because when you're doing those operations, you can think whatever you want to.

Your mind is your own, but as a programmer, you are actually renting out space in your brain, and hopefully, a significant amount of space in your brain, to somebody else for commercial purposes. And so the idea that when you're paying people to think that you can somehow keep things a secret and deceive them into thinking what you want them to think, I just don't think that makes any sense. I think maybe you can get away with it in the short run, but I don't think you'll ever get away with it in the long run.

Scott: I agree with you, but I can't explain the practices, the actual behavior that I see at a lot of the companies that I visit and speak to. I agree with you in principle, yet I'm convinced the majority of work environments don't live up to anywhere near the *standard that you're describing. And part of me wants to say that it has something to do with the size of an organization. That the more layers there are, the harder it is to maintain in the whole openness and accountability. You have pockets.*

Steve: Yeah.

Scott: And a manager might say "my unit, my team, my group" is open. There are often pockets within the group where there is openness and accountability. But across the company, across a division, there isn't. There's a tribalness to it. Beyond the size of a small tribe, it's extremely hard to maintain.

Steve: The bigger you get, the harder it is.

Scott: Right.

Steve: And we do have an asset of being small, and having said that, I have worked for several companies as small as or smaller than we are now that have been highly political. So it's not a foregone conclusion that when you're small, you are automatically open.

Scott: OK, here's your last question. So let's say I am an individual contributor. I'm a programmer on a team that is not functioning well. It's a bad team. Your list of criteria for high-performing teams, we embody the opposite for all of them—not all of them; we are semifunctioning. We're a dysfunctional team, but it's not a good team, and before I decide to leave, what's my list of things I should be thinking about doing to try to help the team environment for the team that I'm on? Is there anything I can do?

Steve: Yeah, I think so. You can lead from the front as well as leading from the rear, and I think most of the things that I've described as being critical to team formation can be done by an individual contributor.

You can define an elevating vision. You can figure out why what you're doing matters and communicate that to the rest of your team. You can organize team activities, both at work or outside of work. The first team that I mentioned was, I think, very much a case of that, where the team formed in spite of the boss, rather than because of him, and, in fact, a big part of the team identity was that we all thought the boss was a greedy jerk.

And so I do think that it does require that somebody exercises some leadership, and by leadership, I mean they figure out where they want to go and then they try to lead other people to wherever that is. And where you want to go is part of it and how you want to get there is another part of it.

One of the things that you see in technical organizations is the distinction between the official management hierarchy and the unofficial expertise hierarchy. And a lot of times the unofficial expertise hierarchy affects day-to-day operations a lot more than the official management hierarchy. It's interesting, because there's some guy who doesn't show up on the org chart who is the guy who everybody goes to to get their question answered, or who everybody defers to when it comes down to key technical strategy questions. And most orgs eventually identify those guys and they show up somehow or other on some corner of the org chart, but that's a form of leadership where there's no budget attached.

There's no title necessarily attached to it, but that person is demonstrating some leadership, and I think that person is in a position to make the team experience better or worse for everybody else.

Scott: We've talked for over an hour now. What key idea do you think most people haven't heard about regarding teams?

Steve: We see far too seldom any effort or thought put into even trying to create teams. The kinds of practices that I described earlier might not be guaranteed to create a beautiful team, but not doing them is almost guaranteed to prevent a beautiful team. And I don't think companies have to spend a lot of money. I don't think they have to have a great deal of expertise in trying to make their teams work better together. I think even a little bit really can go a long way.

I would say that less than a third of the companies we work with have any sort of explicit morale budget or team event budget, and that's just shocking. The companies we work with that do have those budgets, the budgets aren't large. They're a few hundred or a couple thousand dollars a year, maybe, where you're looking at payroll cost for a team is probably a million dollars. I mean, for a small team, the payroll cost is a million dollars, and we're talking about a couple thousand dollars for team building? It's a tiny percentage, but very, very high leverage.