

## Chapter 12. Management Strategy

*We will manage the overall project using business basics—phased delivery, quick and concrete feedback, clear articulation of the business needs of the system, and specialists for special tasks.*

The management dilemma: On the one hand, you would like the manager to make all the decisions. There is no communication overhead, because there is only one person. There is one person to be responsible to upper management. There is one person to have the vision. No one else needs to know about it, because all the decisions come from one person.

We know this strategy doesn't work, because no one person knows enough to do a good job of making all the decisions. Management strategies that are balanced toward centralized control are also difficult to execute, because they require lots of overhead on the part of those being managed.

On the other hand, the opposite strategy doesn't work. You can't just let everyone go off and do what they want without any oversight. People inevitably get off on tangents. Someone needs to have a bigger view of the project, and to be able to influence the project when it gets off course.

Once again, we can fall back on the principles to help us navigate between these two extremes:

- Accepted responsibility—suggests that it is the manager's job to highlight what needs to be done, not to assign work.
- Quality work—suggests that the relationship between managers and programmers needs to be based on trust, because the programmers want to do a good job. On the other hand, this doesn't mean the manager does nothing. However, there is a big difference between "I am trying to get these guys to do a decent job" and "I get to help these guys do an even better job."
- Incremental change—suggests that the manager provides guidance all along, not a big policy manual at the beginning.
- Local adaptation—suggests that the manager needs to take the lead in adapting XP to local conditions, to be aware of how the XP culture clashes with the company culture and to find a way to resolve the misfit.
- Travel light—suggests that the manager doesn't impose a lot of overhead—long all-hands meetings, lengthy status reports. Whatever the manager requires of the programmers shouldn't take much time to fulfill.
- Honest measurement—suggests that whatever metrics the manager gathers should be at realistic levels of accuracy. Don't try to account for every second if your watch only has a minute hand.

The strategy that emerges from this evaluation is more like decentralized decision making than centralized control. The manager's job is to run the Planning Game, to collect metrics, to make sure the metrics are seen by those whose work is being measured, and occasionally to intervene in situations that can't be resolved in a distributed way.

### Metrics

The basic XP management tool is the metric. For example, the ratio between estimated development time and calendar time is the basic measure for running the Planning Game. It lets the team set the Project Velocity. If the ratio rises (less calendar time for a given estimated amount of development), it can mean that the team process is working well. Or, it can mean that the team isn't doing enough besides fulfilling requirements (like refactoring and pairing), and that a price will be paid long-term.

The medium of the metric is the Big Visible Chart. Rather than send e-mail to everyone, which they learn to ignore, the manager periodically (no less than weekly) updates a prominent chart. This is often all the intervention that is needed. You think there aren't enough tests being written? Put a chart of the number of tests up, and update it every day.

Don't have too many metrics, and be prepared to retire metrics that have served their purpose. Three or four measures are typically all a team can stand at one time.

Metrics tend to go stale over time. In particular, any metric that is approaching 100% is likely to be useless. For unit test scores, which must be 100%, this advice doesn't apply, but then the unit test score is more like an assumption than a metric.

You can't count on 97% functional test scores to mean that you have 3% of the effort remaining, however. If a metric gets close to 100%, replace it with another that starts comfortably down in the single digits.

This is not to suggest that you can manage an XP project "by the numbers." Instead, the numbers are a way of gently and noncoercively communicating the need for change. The XP manager's most sensitive barometer of the need for change is awareness of his or her own feelings, physical and emotional. If your stomach knots when you get in the car in the morning, something is wrong with your project and it's your job to effect the change.

## Coaching

What most folks think of as management is divided into two roles in XP: the coach and the tracker (these may or may not be filled by the same person). Coaching is primarily concerned with the technical execution (and evolution) of the process. The ideal coach is a good communicator, not easily panicked, technically skilled (although this is not an absolute requirement), and confident. Often, as coach you want to use the person who on other teams would have been the lead programmer or system architect. However, the coach role in XP is very different.

The phrases "lead programmer" and "system architect" conjure up visions of isolated geniuses making the important decisions on the project. The coach is just the opposite. The measure of a coach is how few technical decisions he or she makes:

The job is to get everybody else making good decisions.

The coach doesn't take responsibility for many development tasks. Rather, the job duties are as follows:

- Be available as a development partner, particularly for new programmers beginning to take responsibility or for difficult technical tasks.
- See long-term refactoring goals, and encourage small-scale refactorings to address parts of these goals.
- Help programmers with individual technical skills, like testing, formatting, and refactoring.
- Explain the process to upper-level managers.

But perhaps the most important job for the coach is the acquisition of toys and food. XP projects seem to attract toys. Lots are of the ordinary brain-teaser type recommended by lateral thinking consultants everywhere. But every once in a while the coach will have the opportunity to profoundly influence development by buying just the right toy, and staying alive to this possibility is one of the coach's greatest responsibilities. For example, on the Chrysler C3 project, design meetings were going on for hours without resolution. So I bought an ordinary kitchen timer and decreed that no design meeting could be longer than 10 minutes. I don't believe the timer was ever used, but its visible presence reminded everyone to be aware of when a discussion had ceased being useful and had turned into a process for avoiding going and writing some code to get the answer for sure.

Food, also, is a hallmark of XP projects. There is something powerful about breaking bread with someone. You have an entirely different discussion with them if you are chewing at the same time. So XP projects always have food lying around. (I particularly recommend Frigor Noir chocolate bars if you can find them, but some projects seem to survive on Twizzler licorice sticks. You're welcome to develop your own local menu.)

Rob Mee writes:

*You know, these test suites are quite insidious. On my team, we reward ourselves with food and beverage. At 2:45: "If we're back at 100% by 3:00 we can have tea and a snack." Of course, we have the snack anyway, even if it takes us until 3:15. However, we almost never get the snack until the tests do run—having the accomplishment behind us makes the break a little party. (Source: e-mail.)*

## Tracking

Tracking is the other major component of management in XP. You can make all the estimates you want, but if you don't measure what really happens against what you predicted would happen, you won't ever learn.

It is the job of the tracker to gather whatever metrics are being tracked at the moment and make sure the team is aware of what was actually measured (and reminded of what was predicted or desired).

Running the Planning Game is a part of tracking. The tracker needs to know the rules of the game cold and be prepared to enforce them even in emotional situations (planning is always emotional).

Tracking needs to happen without lots of overhead. If the person gathering actual development time is asking programmers for their status twice a day, the programmers will soon run away rather than face the interruption. Instead, the tracker should experiment with just how little measurement they can do and still be effective. Gathering real development data twice a week is plenty. More measurement probably won't give you better results.