

# What Managers Can Learn from Extreme Programming: Grassroots learning from developers

Presented at East Bay IT Group meeting, April 12, 2006  
By John Levy 415 663-1818 [info@johnlevyconsulting.com](mailto:info@johnlevyconsulting.com)

## 1.1 Introduction

Many theories and methodologies of development have come down from academic studies at business schools; others have come from project management methods that were developed in non-software disciplines like building construction. Extreme Programming (XP) and agile methodologies are different. They have come up from teams doing the real work of software development. The reality of doing the work keeps XP principles fresh and flexible. The bottom line is always, "does it work?"

Managing software development teams has also suffered from trying to import methods from other disciplines.

Doing management is not like doing software development, but anything that stimulates our thinking can be worthwhile. What if we tried to apply what we have learned from XP to management *above* the project team level?

## 1.2 Principles

### **1.2.1 Planning game**

Planning is an important management activity. In the XP model, planning is something that everyone does at every iteration. The goal is to project the effort it will take to accomplish the implementation of a set of features. Collectively, the team projects the amount of effort it will take to complete a list of features or user functions so we can prune the list of features committed for the next iteration to those that we think we can finish in the fixed time allotted.

We start out making a rough estimate of the time or effort required for each function, rolling them up to get the set of functions that will probably be completed in the fixed iteration time. Normally, we have additional functions to add if we have extra time, and we will drop off functions that are not completed in the allotted iteration time. Then we use what we learned in making estimates in the next iteration.

The projection itself is not as important as the learning that occurs from iteration to iteration. The idea is to get good at estimating effort, at least in the aggregate. A second-level manager has to make estimates in two different domains. He or she is estimating his or her own time on a daily and weekly basis, allocating time and effort for activities that tend to recur. Are these activities similar enough to programming that iterative learning can occur?

Yes, in the sense that management work is repetitive on an activity level. The number of people you can spend time with in a day can be estimated. So are the number of pages of reporting you can write. So a manager can use the planning game model for projecting his or her own time.

The second domain for a manager's estimation is the work of the groups reporting to him or her. Can a manager learn to estimate other people's work?

Yes, again. But it is a little more complex than the XP planning game, because the substance of different teams' work and different departments' work may differ widely.

Let's take a software development program in which there may be three or more teams working in parallel (whether they are XP-oriented teams or not). Let's say that the teams have estimated their work in terms of feature sets to be delivered in a fixed iteration. The manager may choose to apply a scale factor to the teams' estimates, coming up with a projected completion % for the scheduled features. He or she may or may not decide to disclose the projection to the teams – after all, you don't want to discourage aggressive teams' estimates if they can in fact deliver. But the manager has a confidence factor for each team based on the track record of the team. Now, weighting each team's projection by the manager's private confidence factor and summing gives the manager a projected aggregate feature completion rate (even if it doesn't tell us exactly which features will in fact be completed). It is not all that important for a second-level manager to predict the percent completion of each iteration. The upper management is more interested in the probability that 90% or 100% of the proposed features will be included at the end of the projected number of iterations. Using the confidence factor weighting system, the manager can develop a stable projection of the completion probability, particularly when the teams' estimates get better and better over time.

There is another useful way to project completion of a fixed set of features. As the teams plan the specific sub-parts of each major feature, the list of particular needed functions gets longer and longer over time (the "what's remaining to be done" list). But as long as the additions to the list do not overwhelm the original projection, a chart of the incrementally estimated "total set of functions" vs. the "completed functions" should converge. The convergence point is a very good estimate of the completion date (iteration number). [**see sample estimation chart**]

Is this a better way of planning than using PERT charts and Gantt charts? I think so. Anything that is based on iterative feedback from the working team's estimates is better than a pile of numbers collected only once at the beginning of a project.

### **1.2.2 Daily stand-up meetings**

Every management team could benefit from daily check-ins, even if remote, as long as they are short and to the point

How would this work for second-level managers? If you regard your direct reports and staff members as a team, then you know that it is important for them all to know what's going on across your department. What better way than to expose them all to the daily flow of events and activities?

*I managed a department of more than 25 people, with three managers, for some time at Quantum. About a year after I hired the third manager, I decided to change the way I managed into a team approach. In this approach, instead of evaluating what was happening in other departments and outside partners and then making all the decisions myself, I asked my team to participate in all the decisions. The team included the three managers, a staff project manager, and a secretary (called an 'admin' at Quantum). We had weekly meetings,*

*and in these meetings I reviewed all the incoming information and the key activities that were coming up. We would then parcel out responsibility for the activities. Things got a lot more efficient after I arranged for a day-long offsite team-building session. The team learned to support each other while taking on responsibilities.*

How could that process be adapted to daily stand-up meetings?

In a fast-paced development environment, project managers could have a quick view into what each of the other projects were doing *today*, and also hear about the latest customer- or management-dictated changes.

They also would be reminded, daily, of the mutual commitments they have made; and have an opportunity to ask for support for critical activities that are happening *today*.

### **1.2.3 User stories with acceptance criteria**

What would happen if we managed departments the way we manage XP development projects?

[customer is part of the team] We would start out by listening to our “customers” ideas. We would keep them involved through the next steps of the process, too.

[user stories] Then we would transcribe them into a set of “user stories” that represent what the customer wants to do (with our output, or with something that we are helping them to deal with).

[test-driven development] Finally, we would create a set of tests for each story that prove that what we have provided (or will provide) actually works.

Imagine using this process in a department that develops in-house tools for developers. After listening to the users’ problems (say in testing their firmware or their hardware performance), we would write some user stories:  
(1) individual developers perform testing at the subsystem level by bringing to their desk a “test cart” on which is sufficient hardware and software to run all the known regression and performance tests. The tests run and show exactly which tests passed, which failed, and what the key performance measures are.

Now, how about testing the test cart? First, we can easily find out if we have all the hardware needed: look at what’s being used in the QA and other labs to test the subsystem. Second, collect and try out all of the regression tests and performance tests on the selected hardware. Third, develop the run-time control and displays, checking them with the users, to see if they adequately show what the users want.

But what if the product is not so easily defined? For example, some kind of analysis, or a service? Here, it may be necessary to propose a kind of intermediary person who represents the knowledgeable user (customer) and can pass judgment on whether the proposed output meets the needs of the users.

Ultimately, it may not be much different from writing objectives, in the manner of MBO. The key is on clarity, testability, and definiteness of the objectives. And also, how relevant they are to what the user really wants.

Downsides could occur if we can’t define who the users are. And sometimes, we can’t agree on what’s “acceptable” in a service or an analysis. But wouldn’t management be easier – or better – if we *could* define those parameters, and test them?

### **1.2.4 Test first**

Follow on to the User stories, what about test-driven development?

Suppose we are developing processes or equipment for others to use. As soon as we have a rough prototype, we can begin doing the testing. For example, in the case of the firmware test cart, we can start running the (provided) tests as soon as we have some of them available. But how to define in advance the “all tests available”? And how to test against that specification?

This shows a weakness in our definition of the test cart. We should actually enumerate all of the known tests – or at least all of the known *sources* for tests. Then build testers for the tests, which verify that a class of tests, or even a long list of known tests, can run on the device.

Ultimately, as in XP, the user’s opinion of the adequacy of the device and its functionality is what matters. So we would keep going back to a knowledgeable user to verify that we’re developing what they want.

Ultimately, this approach, as the others above, depend a lot on getting an appropriate user – or a surrogate – to participate in the development cycle.

At a higher management level, what is the analog of test-first? It is probably something like setting specific, measurable goals for the management activity. For example, if we’re trying to get more productivity or less turnover in a development organization, we should consider measuring things like schedule slippages, manpower loading, and hiring rates. The more we think carefully about what we measure, the better. Often, we get stuck with lousy measures because a VP thought they were good enough. Or because they are available measures, when other measures look difficult to obtain. Thinking hard about what to measure and why is a good thing to do. Often a top manager does well by repeating a simple question over and over, like “what are customers complaining about this week?” and then asking the “six why’s” associated with certain management and QA methods.

### **1.2.5 Product / business owners actively involved**

Much as I decided to involve my direct reports in the week-to-week management of my department, it is good to involve stakeholders from outside your own department in the process of managing it.

*Quantum was also a company that knew how to use cross-functional teams. There was a formal team structure for managing every product or product line. The team included not only functional managers from key areas, but also an executive sponsor and a process coach. The team was responsible for the product from concept to delivery and was rewarded for the results. There was peer review within the team and bonuses for the team based on results.*

Cross-functional teams can be powerful, but they can also become a bottleneck when they are not run well or rewarded for the right things. It takes a lot of commitment from top management to make team-based organizations work. Not only do they have to provide enough resources (such as process coaches), they have to abstain from pre-empting the teams’ decisions.

If the number of stakeholders in a process is large (say, more than 6 to 8), then it may not work to involve them all in the process. They then need to agree to allow one or a small number of the stakeholders to represent the whole group.

### **1.2.6 Pair programming**

Apparently, one of the hardest things to convince people to do is to try pair programming. However, once a team has tried it for a full development cycle, they can be counted on to love it – or to hate it.

I'm not sure that the XP community has been able to identify exactly what makes paired work effective in some cases and not in others, other than the general resistance to changing one's work style.

Let's assume, then, that we are convinced that paired work is effective: at reducing bug rates early in the cycle, therefore reducing development time and cost; at cross-training team members in code that they might not have looked at if not paired; and at spreading across the team effective coding practices, sometimes as discovered along the way.

How can we adapt this pairing to management?

First, we can imagine taking a new manager into a company or a department and providing on-the-job training by pairing him or her with an experienced manager/coach. The coach may be the previous manager in that position, or it may be a manager who volunteers to step into the paired-manager/coach role on a temporary basis.

There are successful companies, such as General Electric, who deliberately move people around during their first few years to get them exposed to a variety of operating divisions, managers, and roles. Others, like IBM, continually move rising managers between line (managing) roles and staff (assisting others) roles, with lots of training along the way.

But what about the nature of the pair-programming experience? Can we determine an effective way of managing by studying two-at-a-console methods?

Maybe. If the "coach" is able and willing to occasionally change roles and take on the operational decisions while the "trainee" watches and comments, this would be close to some of the paired work we know.

Can you think of a way to use two managers to operate a department on a shared basis as peers? There are companies that encourage their CEOs to bring in COOs who actually run whole businesses. But that's not the same as co-managing. Perhaps another analogy is the commanding officer who has an executive officer. But in each of these cases, there is clear responsibility with one person. I haven't seen an upper level management job that has been shared equally, for this reason: someone has to be responsible.

*I co-managed a firmware group for 7 months once at Quantum. The manager was not experienced, and the team had a chief architect who was smart but not a manager. The two together did not add up to one capable manager for this team. Did putting me in the same office as the manager make things work better? I don't think so, in retrospect, because the next level up manager, who was also new, wouldn't implement the decisions (hard decisions, like removing a couple of people from the team) that I recommended.*

### **1.2.7 Refactoring**

Refactoring is an attractive aspect of XP. We typically create object-oriented modules and interfaces. We continuously improve the code inside the modules to make them more efficient, smaller, and easier to maintain. And we implement only as much functionality as is needed to deliver the next iteration's user stories [YAGNI principle].

To make all this work well, we have to be sure that the interfaces are opaque. That is, that no one can tell by looking at the interface how the functions are implemented inside the module. This is the essence of object-oriented-ness: encapsulation and information-hiding.

Could we apply this idea to management?

Consider business processes, such as customer support, HR functions, and supplier qualification. These days, a lot of these processes get outsourced because there are vendors who can implement them well without being inside the walls of the company. What makes it possible for them to succeed? One aspect is that the process is well-defined. But does that mean that we don't care how they implement each step of the process?

Yes and no. Yes, in that we don't care how a person who is answering the phone gets trained to speak clear English, or which word-processor is used to create a job-application form. So the provider can choose among the available methods without telling us. No, in the sense that we must insist that the processes we are outsourcing are performed with proper concern for all of the stakeholders, people and groups who interface with those processes. This means that a well-functioning outsourced process needs extremely well-defined interaction descriptions, with criteria for how their success is to be measured.

For internal process, there are many departments whose inputs and outputs could well be described without specifying how to do the work in between. The thought process for this is often the subject of Business Process Reengineering, a buzzword and activity that was popular in the '80s & early '90s.

If you are a manager who owns processes that produce things for other groups and departments, it sounds like a good idea to define the inputs and outputs independent of the internal processes. Not many people do this, but I look to it as a promising area.

### **1.2.8 Collective code ownership**

Collective code ownership suggests broader responsibility. In a development team this means asking all the developers (or at least, all the senior developers) to take responsibility for what the product does.

Can we translate this into the management domain? Suppose I ask all of the managers reporting to me to take ownership of the department's processes and outputs. If I don't designate who is responsible for what, I assume that the first thing the "team" will do is to divide up primary responsibility for the work and results. But then, I still want the team to be aware of and participate in all the

strategic decisions that affect how things will turn out. This is precisely what a management team does.

I guess that not many departments – or senior managers – want to spread ownership across their managers this way. But I think it is an intriguing idea.

*About ten years ago, I was part of a strategic planning team of 8 people. We were asked to come up with a vision of the world in ten years, a set of businesses that our company could be in in that world, and then a set of steps that would take the company from where it is now to that place in the future. We worked very well together as a team, and when the people who chartered us asked us to designate a leader to represent the team, we refused. We had spread responsibility for the team's work so thoroughly through the team that we didn't want any one to be the default speaker for the group. We were confident, in fact, that any one of us could represent the team at any time. That is a good sign of collective ownership!*

In a pure management situation, collective ownership means cross-training with other managers at the same level. This is something that is done in the military and in some companies as a matter of course, but it is not general practice in our industry. We tend to be minimalists with regard to management: the fewer the better, and the less time spent managing the better. Does this result in good management?

I don't think so. At the project level, often a technical manager has to do both technical work and management work. But at higher levels of management, it is very important to (1) respect the work of managing, (2) devote resources to training and improving managers' skills as managers, and (3) planning for the future by encouraging people to go into management and supporting them with good leadership and mentoring. How many companies have you worked for that do that? Most of us don't see managing as a professional discipline that has any of the depth that technical work does. This is a big mistake. Management is one of the key strengths of the United States – in spite of the frequent failures we witness. It is also an activity that deserves respect both for its difficulty and also for its value when done well.

### 1.3 Summary

XP is unique in that it has come to us from the bottom up. In other words, we the implementers are teaching the managers how to do this kind of work better. We are doing it from practical principles that we learned the hard way – by trying them out. There is a very active XP community out there trying to make a difference. The software development world is known for a large by its failures, and has a long way to go to earn the respect of others, either technical or managerial.

Management is a soft science and it can learn a lot from experimental techniques. XP principles came as a response to the dismal job software management had been doing. Now, the job is twofold: get management to listen and try out the principles; and move ourselves, who have experience with the principles, into management positions where we can have broader impact. I encourage you to consider moving in that direction.

In many ways, the "spirit" of XP is more important than the particulars of the principles. We are using agile methods because we live in an environment of incomplete specifications and rapid change. It's good to focus on what works, without getting bogged down with heavyweight processes. Being "agile" is a good

mental stance, and we would do well to teach our managers that flexibility and experimentation are valid pathways to better management.