A Project Management Primer[1]

Karl E. Wiegers

Process Impact
www.processimpact.com

A day you've anticipated—and perhaps feared—has finally arrived: You've been "promoted" from the engineering staff to a software project lead or team leader position. This may be your chosen career path, or you may have reluctantly agreed to give it a try. In either case, you have probably received little education in the arts of project and people management and leadership.

There's more to leadership and management (which aren't the same thing) than simply doing the opposite of whatever Dilbert's boss does. As you contemplate your new mission, consider the following list of action items. You can't tackle every promising idea at once, but the recommendations described here will help you focus on actions that will improve both your own effectiveness and that of your team.

Set Your Priorities

Perhaps the most significant step you can take initially is to consciously set your priorities as a manager. While you may be tempted to remain heavily engaged in software development activities (indeed, this may still be part of your job), you also have a new set of responsibilities. Too many new managers cannot resist the lure of staying technically active, which can lead them to neglect the needs of others on the project team who look to the manager for help.

Effective leaders know their top priority is to provide services to the members of the team. These services include coaching and mentoring, resolving problems and conflicts, providing resources, setting project goals and priorities, and providing technical guidance when appropriate. Make it clear to your team members that you are always available to help them. I find it valuable to think of myself as working for the people I supervise, not the reverse. Team members who need something from you should have a non-maskable interrupt priority over most other things you might be doing.

Your second priority is to satisfy your organization's customers. As a manager, you have little direct ability to satisfy customers, since you no longer personally provide the products and services that accomplish this. Instead, you must create an environment that permits the members of your team to most effectively meet the needs of your customers. The manager provides a powerful enabling function that can contribute significantly to customer satisfaction.

Your third priority should be to work on your own projects. These may be technical projects, or activities requested by your own managers, such as serving on steering committees. Be prepared to postpone these activities when they conflict with the two higher priorities.

---

[1] This paper was originally published in *Software Development* magazine in June 1998. It is reprinted here (with modifications) with permission from *Software Development*.

Explicitly taking actions to please your own managers should be your lowest priority. In a congruent (non-Dilbertesque) organization, your managers will be thrilled if you are successful at the three more important items. We aren't all fortunate enough to work in a nicely congruent context, but strive to keep the most important responsibilities of your new job at the top of your list. Focus on helping your team members be as effective - and as happy - as possible, instead of going out of your way to satisfy those above you on the advancement ladder.

Analyze Your Skills Gaps

Unless you have already been preparing yourself for the new position, you may perceive some gaps in your current leadership and management skill sets. Your strong technical background was probably a factor in your being selected for the leadership role, but you'll need some additional skills to be fully effective. Take an honest inventory of your strengths and shortcomings in critical people and project skills and begin closing any gaps.

Software people aren't famous for having superb people skills. You may wish to enhance your adroitness at handling interpersonal relationships, resolving conflicts, persuasion, and selling ideas. You'll have to be able to deal with situations ranging from hiring and firing staff, to negotiating schedules, to having someone crying in your office during a performance discussion session.

I found it valuable to start my management career with a listening skills class. As individual contributors, we often have the luxury of energetically pushing our own technical agendas on the group. Managing effectively demands a more collaborative and receptive interpersonal style. It took me awhile to learn how (and when) to skillfully channel my natural assertiveness. The listening skills class provided a communication mechanism that I have found useful in many situations.

Next, step to the other side of the podium and improve your presentation skills. If you are really uncomfortable at public speaking, a Dale Carnegie course can be helpful. Practice what you learn through such training, and you will find that your enhanced communication ability will serve you well in any job you hold in the future.

As a project leader, you will be responsible for coordinating the work of others, for planning and tracking projects, and for taking corrective actions when necessary to get a project back on track. Take a training course in project management, and begin reading books and articles on project and risk management. Join the Project Management Institute and read their monthly magazine, *PM Network*. The Software Engineering Institute's Software Capability Maturity Model (*The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995) contains much useful advice on software project planning and project tracking. Your ability to set priorities, conduct effective meetings, and communicate clearly will have a substantial impact on your effectiveness as a manager.

Define "Quality"

Almost everyone takes quality seriously and wants to produce high-quality products. However, there's no universal definition of what "quality" means in software. Debates rage about "good enough" software versus more traditional views of software quality. To help steer your group toward success, spend some time working with your team members and your customers to understand what quality means to them.

These two communities often do not have the same definition in mind, so it's easy to work at cross-purposes. A manager focused on delivery schedule may be impatient with an engineer who wants to formally inspect every line of code. A customer to whom reliability is paramount won't be happy with a product containing scads of seldom used features, along with scads of bugs. A spiffy new GUI might turn off a user whose fingers have memorized how to efficiently use the previous version of the product.

To better understand our customers' views of software quality, my group at Kodak once invited our customers (fellow employees) and their managers to an open forum to discuss this topic. This forum was valuable because it showed where our group's ideas of what constituted quality did not match the perceptions of those who used our products. Understanding the differences can help you focus energy where it will yield the greatest customer benefit, not just where it will provide the greatest developer satisfaction.

Traditional interpretations of software quality include conformance to specifications, satisfying customer needs, and the absence of defects in code and documentation. The buzzword of "six-sigma quality" sets a very high bar for defect density or frequency of failure, but it doesn't address such quality dimensions as rapid product delivery, usability, a rich feature set, and delivering value for the price paid. While we all would love to have all of these quality characteristics maximized in the products we both create and purchase, trade-offs are always necessary.

During the requirements phase on one project, we created a list of ten quality attributes (such as efficiency, interoperability, correctness, and ease of learning) we thought would be important to the users. We asked a group of key customer representatives to rate each of these attributes on a scale of 1 to 5 for desirability. Once we had determined which attributes were most significant, we could design the application to achieve those objectives. If you don't go to the trouble of learning what quality means to your customers and then designing to deliver that quality, you're just lucky if the customers get the quality characteristics they expect.

One of the more interesting quality definitions I have heard is "the customer comes back, but the product does not." Work with your customers and developers to define appropriate quality goals for each product. Once determined, make achieving these quality objectives an unambiguous top priority. Lead by example, setting very high personal standards for the quality of your own work. Adopt this motto: "Strive for perfection; settle for excellence."

Recognize Progress

Recognizing and rewarding the achievements of your team members is an important way to keep them motivated. Unless your group already has a recognition program in place, this should be one of your top priorities. Recognition can range from the symbolic (certificates, traveling trophies) to the tangible (movie coupons, restaurant gift certificates, cash bonuses). Presenting recognition of some kind says, "Thanks for what you did to help," or "Congratulations on reaching that milestone." By investing a small amount of thought and money in a recognition and reward program, you can buy a lot of goodwill and future cooperation. Also remember to recognize people outside the development group, including customer representatives and support people who contribute in special ways to the project's success.

Talk to your team members to understand the sorts of recognition and rewards they find meaningful. Make recognition events for accomplishments large and small a standard component of your team culture. Also practice the implicit recognition of showing sincere interest in the work being done by each member of your team and doing all you can to remove obstacles to their effectiveness. Recognition is one way to demonstrate to your team members—and to others

outside your group—that you are aware of and appreciate the contributions they make to the success of the team.

Learn From the Past

It's possible that some of the projects undertaken by your group in the past were not completely successful. Even on successful projects, we can often identify things we would do differently next time. As you embark on your new leadership role, take some time to understand why earlier projects have struggled, and plan to avoid repeating the same mistakes. Software development is too hard for each manager to take the time to make every possible mistake on his or her own. Jump-start your own success by learning from what has worked - and failed - before.

Begin with a non-judgmental assessment of the last few projects undertaken by your group, whether successful or not. Your goal is not to determine blame, but to do a better job on future projects. Conduct a post-project review (sometimes called a postmortem) to learn what went well and what could have been done better. Lead the team in brainstorming sessions or use an impartial facilitator to analyze each current project in the same way at major milestones.

In addition, become well-acquainted with established software industry best practices. A good place to start is with Part III of Steve McConnell's Jolt Award winner, *Rapid Development* (Microsoft Press, 1996), which describes 27 such best practices. Also beware of repeating the 36 classic software development mistakes McConnell describes. Your team members may resist new ways of working, but your role as leader is to ensure that the team consistently applies the best available methods, processes, and tools. Actively facilitate the sharing of information among team members, so local best practices can become a part of every developer's tool kit.

Set Improvement Goals

Once you've conducted a retrospective into previous projects and determined what "quality" means to your group, set some goals for both short- and long-term improvements. Goals should be quantified whenever possible, so you can select a few simple metrics that will indicate whether you are making adequate progress toward the goals.

For example, if you've determined that projects are often late because of volatile requirements, you might set a goal to improve the requirements stability by 50% within six months. Such a goal requires that you actually count the number of requirements changes per week or month, understand their origins, and take actions to control those changes. This will likely require alterations in the way you interact with those who supply the requirements changes.

Your goals and metrics make up part of the software process improvement program you should put into place. It's fashionable to disdain "process" as the last refuge of uncreative bureaucrats. The reality, though, is that *every* group can find ways to improve the work it performs. Indeed, if you continue to work the way you always have, you shouldn't expect to achieve any better results than you have before.

There are two compelling reasons to improve your processes: to correct problems, and to prevent problems. Make sure your improvement efforts align with known or anticipated threats to the success of your projects. Lead your team in an exploration of the strengths and shortcomings of the practices currently being used, and of the risks facing your projects.

My group held a two-session brainstorming exercise to identify barriers to improving our software productivity and quality. In session one, the participants wrote their thoughts about this

topic on sticky notes, one idea per note. A facilitator collected and grouped the ideas as they were generated. At the end, we had a dozen major categories, which we then recorded on large flipchart sheets.

In the second session, the same participants wrote ideas for overcoming these barriers on sticky notes and attached them to the appropriate flipcharts. Further refinement led to a handful of specific action items that could be addressed in our effort to break down barriers and help team members achieve their software quality and productivity objectives.

Setting measurable and attainable goals brings a focus to your improvement efforts. Keep the goals a visible priority, and monitor progress with the group periodically. Remember that your objective is to improve the technical and business success achieved by your projects and company, not to satisfy the detailed expectations found in some process improvement book. Treat improvement efforts as mini-projects, with deliverables, resources, schedules, and accountability. Otherwise, process improvement activities will always get a lower priority than the more enticing technical work.

Start Slowly

This article suggests many activities that can help you, a new software manager, begin steering your team toward greater success. With the day-to-day pressures of the new job, it can be a struggle just to keep your head above water. But you have a critical role to play (and a window of opportunity) in shaping the culture and practices of your software development group for the long term. You can't do all of these things at once, but select a few that seem most appropriate for your situation and get started.

As a software manager, you're responsible for doing more than completing the project on time and on budget. You must also: lead the technical staff into a cohesive team that shares a commitment to quality; foster an environment of collaborative teamwork; promote and reward the application of superior software engineering practices; and balance the needs of your customers, your company, your team members, and yourself.

It's a big job. Good luck!