Managing software projects is difficult under the best circumstances. The project manager must balance competing stakeholder interests against the constraints of limited resources and time, ever-changing technologies, and unachievable demands from unreasonable people. Project management is people management, technology management, business management, risk management, and expectation management. It’s a juggling act, with too many balls in the air at once.

Unfortunately, many new project managers receive little training in how to do the job. Anyone can learn to draw a Gantt chart, but effective project managers also rely on the savvy that comes from painful experience. Coaching and survival tips from people who have already done their tour of duty in the project management trenches can save you from learning such lessons the hard way. Here are 21 such tips for success, which I’ve learned from both well-managed and challenged projects. The tips are organised into five categories:

1. Laying the groundwork for success.
2. Planning the project.
3. Estimating the work.
4. Tracking your progress.
5. Learning for the future.

Together, the practices in these five categories define a project management control system that can help your project deliver on expectations. Keep these suggestions in mind on your next project, recognising that none of them is a silver bullet for your project management problems.

Laying the Groundwork

Tip #1: Define project success criteria

At the beginning of the project, make sure the stakeholders share a common understanding of how they will determine whether this project is successful. Too often, meeting a predetermined schedule is the only apparent success factor, but there are certainly others. Begin by identifying your stakeholders and their interests and expectations. Next, define some clear and measurable business goals. Some examples are:

- Increasing market share by a certain amount by a specified date.
- Reaching a specified sales volume or revenue.
- Achieving certain customer satisfaction measures.
• Saving money by retiring a high-maintenance legacy system.
• Achieving a particular transaction processing volume and correctness.

These business goals should imply specific project success criteria, which should again be measurable and trackable. They could include achieving schedule and budget targets, delivering committed functionality in a form that satisfies customer acceptance tests, complying with industry standards or government regulations, or achieving specific technological milestones.

Also, keep your eye on team member job satisfaction, sometimes indicated by staff turnover rate and the willingness of team members to do what it takes to make the project succeed. The business objectives define the overarching goal, though. It doesn't matter if you deliver to the specification on schedule and budget if those factors don't clearly align with business success.

Remember that not all of these defined success criteria can be your top priority. You'll have to make some thoughtful trade-off decisions to ensure that you satisfy your most important priorities. If you don't define clear priorities for success, team members can wind up working at cross-purposes, leading to frustration, stress, and reduced teamwork effectiveness.

**Tip #2: Identify project drivers, constraints, and degrees of freedom**

Every project must balance its functionality, staffing, cost, schedule, and quality objectives [Wiegers, 1996]. Define each of these five project dimensions as either a constraint within which you must operate, a driver strongly aligned with project success, or a degree of freedom you can adjust within some stated bounds. There's bad news: not all factors can be constraints and not all can be drivers. The project manager must have some flexibility to react to schedule slips, demands for increased functionality, staff turnover, and other realities.

A "flexibility diagram" such as that shown in Figure 1 visually depicts your constraints, drivers, and degrees of freedom. A constraint gives the project manager no flexibility in that dimension, so it is plotted at the zero value on its axis. A driver yields a small amount of flexibility, so its point is plotted a bit higher than zero. Degrees of freedom provide varying degrees of latitude. They represent parameters the project manager can adjust to achieve the project's success drivers within the limits imposed by its constraints. Connecting the five plotted points creates an irregular pentagon. The smaller the area inside the pentagon, the more constrained the project is.

I once heard a senior manager ask a project leader how long it would take to deliver a planned new large software system. The project leader replied, "Two years." The senior manager said, "No, that's too long. I need it in six months." The project leader's response was simply, "Okay," despite the fact that nothing had changed in the few seconds of that conversation to make the six-month target achievable. A better response would have been to negotiate a realistic outcome through a series of questions such as the following:
How critical is the six-month target? Does something drastic happen if we don't deliver in six months [schedule is a constraint], or is that just a desirable target date [schedule is a driver]?

If the six months is a firm limit, what subset of the requested functionality do you absolutely need delivered by then?

Can I get more people to work on it? [staff is a degree of freedom].

Do you care how well it works? [quality is a degree of freedom].

---

**Figure 1.** A flexibility diagram for a project that is staff-constrained and schedule-constrained, with cost being a driver, and quality and features being degrees of freedom.

**Tip #3: Define product release criteria**

Early in the project, decide what criteria will indicate whether the product is ready for release. Possible release criteria might include the following:

- There are no open high-priority defects.
- The number of open defects has decreased for X weeks and the estimated number of residual defects is acceptable.
- Performance goals are achieved on all target platforms.
- Specific required functionality is fully operational.
- Quantitative reliability goals are satisfied.
- X% of system tests have been passed.
- Specified legal, contractual, or regulatory goals are met.
- The optimum marketplace time to release has arrived.
- Customer acceptance criteria are satisfied.

Whatever criteria you choose should be realistic, objectively measurable, documented, and aligned with what "quality" means to your customers. Decide early on how you will tell when you're done, track progress toward your goals, and stick to your guns when confronted with pressure to ship before the product is ready for prime time.

Carefully consider your target market segments when deciding on release criteria.
[Rothman, 1999]. The early adopters and enthusiasts have a higher tolerance for defects than do the pragmatic early majority of customers or the conservative late majority. In contrast, time to market and innovative features or technology usage are most important to the early adopters.

**Tip #4: Negotiate achievable commitments**

Despite pressure to promise the impossible, never make a commitment you know you can't keep. Engage in good-faith negotiations with customers, managers, and team members about goals that are realistically achievable. Negotiation is required whenever there is a gap between the schedule or functionality the key project stakeholders demand and your best prediction of the future as embodied in project estimates. Principled negotiation involves four precepts [Fisher, 1991]:

- Separate the people from the problem.
- Focus on interests, not positions.
- Invent options for mutual gain.
- Insist on using objective criteria.

Any data you have from previous projects will help you make persuasive arguments, although there is no real defence against truly unreasonable people.

I once met with an aggressive and intimidating senior manager to discuss our department's software process improvement plans. Jack was eager to see our department achieve CMM Level 2 by July of 1996. My process improvement group had carefully studied the problem and estimated that the end of 1997 was the earliest date that was even remotely feasible. After some debate, Jack grudgingly agreed to the end of 1996, but I regarded even that goal as pure fantasy. After additional discussion, I finally said, "Jack, I'm not going to commit to the end of 1996." I don't think anyone had ever told Jack he wouldn't make a commitment that Jack demanded. He wasn't sure what to say next. Jack eventually agreed to the target date to which I was willing to commit.

Plan to renegotiate commitments when project realities (such as staff, budget, or deadlines) change, unanticipated problems arise, risks materialise, or new requirements are added. No one likes to have to modify his commitments. However, if the reality is that the initial commitments won't be achieved, let's not pretend that they will up until the moment of unfortunate truth.

**Planning the Project**

**Tip #5: Write a plan**

Some people believe the time spent writing a plan could be better spent writing code, but I don't agree. The hard part isn't writing the plan. The hard part is actually doing the planning-thinking, negotiating, balancing, asking, listening, and thinking some more. Actually writing the plan is mostly transcription at that point. The time you spend analysing what it will take to solve the problem will reduce the number of surprises you have to cope with later in the project. Today's multi-site and cross-
cultural development projects demand even more careful planning and tracking than do traditional projects undertaken by a co-located team.

A useful plan is much more than a schedule or work breakdown structure of tasks to perform. It also includes:

- Staff, budget, and other resource estimates and plans.
- Team roles and responsibilities.
- How you will acquire and train the necessary staff.
- Assumptions, dependencies, and risks.
- Descriptions of, and target dates for, major deliverables.
- Identification of the software development life cycle that you'll follow.
- How you will track and monitor the project.
- Metrics that you'll collect.
- How you will manage any subcontractor relationships.

Your organisation should adopt a standard software project plan template, which can be tailored for various kinds of projects. An excellent starting point is IEEE Std 1058-1998, the "IEEE Standard for Software Project Management Plans" [IEEE, 1998]. This standard describes a comprehensive template, sufficient for the largest projects. Study this template to see what sections would make sense for the types and sizes of projects that you work on.

If you commonly tackle different kinds of projects, such as major new product development as well as small enhancements, adopt a separate project plan template for each. Avoid overburdening small projects with excessive documentation that adds little value. The project plan should be no any longer nor more elaborate than necessary to make sure you can successfully execute the project. But always write a plan.

**Tip #6: Decompose tasks to inch-pebble granularity**

Inch-pebbles are miniature milestones (get it?). Breaking large tasks into multiple small tasks helps you estimate them more accurately, reveals work activities you might not have thought of otherwise, and permits more accurate, fine-grained status tracking. Select inch-pebbles of a size that you feel you can estimate accurately. I feel most comfortable with inch-pebbles that represent tasks of about 5 to 15 labour-hours, or about one to three days in duration. Overlooked tasks are a common contributor to schedule slips, so breaking large problems into small bits reveals more details about the work that must be done and improves your ability to make accurate estimates.

You can track progress based on the number of inch-pebbles that have been completed at any given time, compared to those you planned to complete by that time. Defining the project's work in terms of inch-pebbles is an aid to tracking status through earned value analysis [Lewis, 2000]. The earned value technique compares the investment of effort or dollars that you've made to date with progress as measured by completed inch-pebbles.
Tip #7: Develop planning worksheets for common large tasks

If your team frequently undertakes certain common tasks, such as implementing a new object class, executing a system test cycle, or performing a product build, develop activity checklists and planning worksheets for these tasks. Each checklist should include all of the steps the large task might need. These checklists and worksheets will help each team member identify and estimate the effort associated with each instance of the large task he must tackle. People work in different ways and no single person will think of all the necessary tasks, so engage multiple team members in developing the worksheets. Using standard worksheets will help the team members adopt common processes that they can tune up as they gain experience. Tailor the worksheets to meet the specific needs of individual projects.

Tip #8: Plan to do rework after a quality control activity

I've seen project task lists in which the author assumed that every testing experience will be a success that lets you move into the next development activity. However, almost all quality control activities, such as testing and peer reviews, find defects or other improvement opportunities. Your project schedule or work breakdown structure should include rework as a discrete task after every quality control activity. Base your estimates of rework time on previous experience. For example, you might have historical inspection data indicating that, on average, your developers find 25 defects per thousand lines of code by inspection and that it costs an average of 40 minutes to fully repair each code defect. You can crunch these kinds of numbers to come up with average expected rework effort for various types of work products. If you don't actually have to do any rework after a test, great; you're ahead of schedule on that task. Don't count on it, though.

Tip #9: Manage project risks

If you don't identify and control project risks, they will control you. A risk is a potential problem that could affect the success of your project, a problem that hasn't happened yet, and you'd like to keep it that way [Wiegers, 1998]. Risk management has been identified as one of the most significant best practices for software development [Brown, 1996]. Simply identifying the possible risk factors isn't enough. You also have to evaluate the relative threat each one poses so you can focus your risk management energy where it will do the most good.

Risk exposure is a combination of the probability that a specific risk could materialise into a problem and the negative consequences for the project if it does. To manage each risk, select mitigation actions to reduce either the probability or the impact. You might also identify contingency plans that will kick in if your risk control activities are not effective. Suppose you are concerned that your top developer might move to Australia to be with her new boyfriend. Consider the following actions:

- Pay her more money, offer to hire the boyfriend, or give her more vacation time to fly to Australia periodically (reduces probability).
- Keep her on as a telecommuting employee or contractor, have her document
her work, or have her impart her specialised knowledge to other employees (reduces impact).

- Line up a consultant or contract specialist to replace her if she leaves anyway (contingency plan).

A risk list does not replace a plan for how you will identify, prioritise, control, and track risks. Incorporate risk tracking into your routine project status tracking. Record which risks materialised and which mitigation actions were effective for reference on future projects.

**Tip #10: Plan time for process improvement**

Your team members are already swamped with their current project assignments, but if you want the group to rise to a higher plane of software engineering capability, you'll have to invest some time in process improvement [Wiegers, 1999]. Set aside some time from your project schedule, because software project activities should include making process changes that will help your next project be even more successful. Don't allocate 100 percent of your team members' available time to project tasks and then wonder why they don't make any progress on the improvement initiatives. Some process changes can begin to pay off immediately, whereas you won't reap the full return on your investment in other improvements until the next project. View process improvement as a strategic investment in the sustained effectiveness of your development organisation. I liken process improvement to highway construction: it slows everyone down a little bit for a time, but after the work is done, the road is a lot smoother and the throughput greater.

**Tip #11: Respect the learning curve**

The time and money you spend on training, reading and self-study, consultants, and developing improved processes are part of the investment your organisation makes in project success. Recognise that you'll pay a price in terms of a short-term productivity loss when you first try to apply new processes, tools, or technologies. Don't expect to get fabulous benefits from new software engineering approaches on the first try, no matter what the tool vendor's literature or the methodology consultant's brochure claims. Instead, build extra time into the schedule to account for the inevitable learning curve. Make sure your managers and customers understand the learning curve and accept it as an inescapable consequence of working in a rapidly changing, high-technology field.

**Estimating the Work**

**Tip #12: Estimate based on effort, not calendar time**

People generally provide estimates in units of calendar time. I prefer to estimate the effort (in labour-hours) associated with a task, then translate the effort into a calendar-time estimate. A 20-hour task might take 2.5 calendar days of nominal full-time effort, or 2 exhausting days. However, it could also take a week if you have to wait for critical information from a customer or stay home with a sick child for
two days. The translation of effort into calendar time is based on estimates of how many effective hours I can spend on project tasks per day, any interruptions or emergency bug fix requests I might get, meetings, and all the other places into which time disappears. If you keep track of how you actually spend your time at work, you'll know how many effective weekly project hours you have available on average [Wiegers, 1996]. Typically, this is only about 50 to 60 percent of the nominal time people spend at work, far less than the assumed 100 percent effective time on which so many project schedules are planned.

**Tip #13: Don't schedule multi-tasking people for more than 80 percent of their time**

The task-switching overhead associated with the many activities we are all asked to do reduces our effectiveness significantly. Excessive multi-tasking introduces communication and thought process inefficiencies that reduce individual productivity. I once heard a manager say that someone on his team had spent an average of eight hours per week on a particular activity, so she could do five of them at once. In reality, she'll be lucky if she can handle three such tasks. Some people multi-task more efficiently than others. If some of your team members trash when working on too many tasks at once, set clear priorities and help them succeed by focusing on just one or two objectives at a time.

**Tip #14: Build training time into the schedule**

Estimate how much time your team members spend on training activities annually, and subtract that from the time available for them to work on project tasks. You probably already subtract out average values for vacation time, sick time, and other assignments; treat training time the same way. Recognise that the high-tech field of software development demands that all practitioners devote time to ongoing education, both on their own time and on the company's time. Arrange just-in-time training when you can schedule it, as the half-life of new technical knowledge is short unless the knowledge is put to use promptly. Attending a training seminar can be a team-building experience, as project team members and other stakeholders hear the same story about how to apply improved practices to their common challenges.

**Tip #15: Record estimates and how you derived them**

When you prepare estimates for your work, write down those estimates and document how you arrived at each of them. Understanding the assumptions and approaches used to create an estimate will make them easier to defend and adjust when necessary. It will also help you improve your estimation process. Train the team in estimation methods, rather than assuming that every software developer and project leader is instinctively skilled at predicting the future. Develop estimation procedures and checklists that people throughout your organisation can use.

An effective group estimation technique is the Wideband Delphi method [Wiegers, 2000]. Wideband Delphi builds on the principle that multiple heads are better than one. The Delphi estimation method asks a small team of experts to anonymously
generate individual estimates from a problem description and reach consensus on a final set of estimates through iteration. Figure 2 illustrates the Wideband Delphi process flow. The outputs from the process include a complete list of project and quality-related tasks and an estimate for each task, in whatever units the team chose (such as dollars, weeks, or labour-hours). Participation by multiple estimators and the use of anonymous estimates to prevent one participant from biasing another make the Delphi method more reliable than simply asking a single individual for his best guess.

**Figure 2. The Wideband Delphi process flow.**

**Tip #16: Use estimation tools**

Many commercial tools are available to help you estimate entire projects. Based on large databases of actual project experience, these tools can give you a spectrum of possible schedule and staff allocation options. They'll also help you avoid the "impossible region," combinations of product size, effort, and schedule where no known project has been successful. The tools incorporate a number of "cost drivers" you can adjust to make the tool more accurately model your project, based on the technologies used, the team's experience, and other factors. Over time, you can calibrate the tool with your own project data to make it an even more reliable predictor of the future. A good tool to try is Estimate Pro from the Software Productivity Center (www.spc.ca). Others include KnowledgePlan (www.spr.com), SLIM (www.qsm.com), and Cost Xpert (www.costxpert.com). You can compare the estimates from the tools with the bottom-up estimates generated from a work breakdown structure. Reconcile any major disconnects so you can generate the most realistic overall estimate.

**Tip #17: Plan contingency buffers**

Projects never go precisely as planned. The prudent project manager incorporates budget and schedule contingency buffers (also known as management reserve) at the end of major phases to accommodate the unforeseen. Use your project risk analysis to estimate the possible schedule impact if several of the risks materialise and build that projected risk exposure into your schedule as a contingency buffer. Even more sophisticated is the use of critical chain analysis, a technique that pools the uncertainties in estimates and risks into a rational overall contingency buffer [Zultner, 1999].

Your manager or customer might view these contingency buffers as padding, rather than as the sensible acknowledgement of reality that they are. To help persuade skeptics, point to unpleasant surprises on previous projects as a rationale for your foresight. If a manager elects to discard contingency buffers, he has tacitly absorbed all the risks that fed into the buffer and assumed that all estimates are perfect, no scope growth will occur, and no unexpected events will take place. The
reality on most projects is quite different. I'd rather see us deal with reality, however unattractive, than to live in Fantasyland, which leads to chronic disappointments.

**Tracking Your Progress**

**Tip #18: Record actuals and estimates**

Someone once asked me where to get historical data to improve her ability to estimate future work. My answer was, "If you write down what actually happened today, that becomes historical data tomorrow." Unless you record the actual effort or time spent on each task and compare them to your estimates, you'll never improve your estimating approach. Your estimates will forever remain guesses. Each individual can begin recording estimates and actuals, and the project manager should track these important data items on a project task or milestone basis. In addition to effort and schedule, you could estimate and track the size of the product, in units of lines of code, function points, classes and methods, GUI screens, or other units that make sense for your project.

**Tip #19: Count tasks as complete only when they're 100 percent complete**

We give ourselves a lot of partial credit for tasks we've begun but not fully completed: "I thought about the algorithm for that module in the shower this morning, and the algorithm is the hard part, so I'm probably about 60 percent done." It's difficult to accurately assess what fraction of a sizable task has actually been done at a given moment. One benefit of using inch-pebbles for task planning is that you can break a large activity into a number of small tasks and classify each small task as either done or not done, nothing in between. Your project status tracking is then based on the fraction of the tasks that are completed, not the percent completion of each task. If someone asks you whether a specific task is complete and your reply is, "It's all done except...", it's not done! Don't let people "round up" their task completion status; use explicit criteria to tell whether a step truly is completed.

**Tip #20: Track project status openly and honestly**

An old riddle asks, "How does a software project become six months late?" The answer is, "One day at a time." The painful problems arise when you don't know just how far behind (or, occasionally, ahead) of plan the project really is. Create a climate in which team members feel it is safe for them to report project status accurately. Strive to run the project from a foundation of accurate, data-based facts, rather than from the misleading optimism that sometimes arises from the fear of reporting bad news. Use project status information and metrics data to take corrective actions when necessary and to celebrate when you can. You can only manage a project effectively when you really know what's done and what isn't, what tasks are falling behind their estimates and why, and what problems and issues remain to be tackled. The five major areas of software measurement are size, effort,
time, quality, and status. Remember the cardinal rule of software metrics: metrics data must never be used to punish or reward individuals for their performance.

Learning for the Future

Tip #21: Conduct project retrospectives

Retrospectives (also called post-mortems or post-project reviews) provide an opportunity for the team to reflect on how the last project or the previous phase went and to capture lessons learned that will help enhance your future performance [Kerth, 2001]. During such a review, identify the things that went well, so you can create an environment that enables you to repeat those success contributors. Also look for things that didn't go so well, so you can change your approaches and prevent those problems in the future. In addition, think of events that surprised you. These might be risk factors for which you should be alert on the next project.

Conduct retrospectives in a constructive, honest atmosphere. Don't make them an opportunity to assign blame for previous problems. Capture the lessons learned from each review and share them with the entire team and organisation, so all can benefit from your painfully gained experience. I like to write lessons learned in a neutral way, such that it isn't obvious whether we learned the lesson because we did something right or because we made a mistake.

These 21 project management tips won't guarantee success, but they will help you get a solid handle on your project and ensure that you're doing all you can to make it succeed in an unpredictable world.

Bibliography


Karl Wiegers, Ph.D., is Principal Consultant with Process Impact, a software process consulting and education company in Portland, Oregon. Karl's most recent book is Practical Project Initiation: A Handbook with Tools. Karl is also the author of four other books and 170 articles. He is a frequent speaker at software conferences and professional society meetings. You can reach Karl through www.projectinitiation.com or www.processimpact.com.

Enjoyed this article? Now read 21 Ways to Excel at Project Management.

© 2013 Project Smart. All rights reserved.