Editor in Chief: Steve McConnell ■ Construx Software ■ software@construx.com

The Software Manager's Toolkit

Steve McConnell

The expert developer's intellectual toolkit is filled with coding tips and tricks as well as knowledge of design methodologies, configuration management, and the details of current technologies and development tools. Some developers might have acquired additional intellectual tools related to requirements engineering, maintenance, testing, and quality assurance.

The general, non-software manager's intellectual toolkit contains a completely different set of tools. General managers are trained in how to create budgets, interview prospective employees, conduct performance reviews, and so on. Project managers are trained to manage scope, time, cost, quality, human resources, communications, and risk.[1]

Effective software project managers have some of the same tools expert developers, general managers, and project managers use, but they also need tools the others don't have. The shortage of managers skilled in software-specific competencies causes severe problems. Capers Jones points out that poor software project management is associated with cancelled projects, cost and schedule overruns, low quality, missed market opportunities, low morale, and high turnover.[2]

Tools

The technical manager should have tools for five kinds of software-specific work: estimating, planning, tracking, risk management, and measuring. Let's take a closer look at each of these areas.

Estimating

People often think of estimating as "guesswork" or "expert judgment," but those views are more an indictment of the current state of the practice than a description of effective cost and schedule setting. Skilled technical managers go through three basic steps to create project planning numbers: first, they estimate the scope of the software; then, they compute the effort needed to build a product of that scope; and finally, they compute a schedule based on the effort estimate.

The best estimates involve little guesswork or expert judgment. In the best case, a product attribute is counted rather than estimated to create the scope "estimate." The manager might count function points, requirements, GUI elements, or some other product attribute. The estimator then uses the organization's productivity data to compute effort and schedule. A good estimate is counted and computed rather than guessed or judged.

Developers typically don't learn whole-software-project estimation, and general management training certainly doesn't teach it. To add estimation tools to their toolkits, software project managers must develop skills beyond those acquired in the technical trenches.

IEEE Software

Planning

Many people think of software project planning as creating a list of activities in Microsoft Project and printing a Gantt chart. In reality, that activity is more properly called scheduling and is only one small tool in the technical manager's planning toolkit—and not the most important one either. The effective software project planner must have tools for each of the following activities:

- estimating whole-project effort and schedule;
- determining how many people are needed on the project team—including an appropriate mix of developers, testers, and managers; a good balance of junior and senior staff; how to build up the staff over the course of a project; and so on;
- choosing a lifecycle model appropriate for the project;
- selecting appropriate technical practices to elicit requirements, create designs, manage the intellectual property generated on the project, construct the code, test the software, and capture the project's experience for use on future projects;
- identifying the kinds of quality assurance activities needed to meet the project's cost, schedule, and quality goals—striking a balance between technical reviews and testing, the number of levels of technical reviews, and so on; and
- crafting a set of tracking indicators that will provide status visibility throughout the project.

A software project manager will not learn how to perform any of these activities in a general, non-software management training course, and only the most astute developer will acquire these skills by doing hands-on technical work. Most of these activities are not done on most projects.

Tracking

On a typical project, technical management is almost a black-box function: you create some plans at the beginning, you rarely know what's going on during the project, and you're forced to accept whatever comes out at the end. Capers Jones reports that "software progress monitoring is so poor that several well-known software disasters were not anticipated until the very day of expected deployment."[3]

On a well-run project, you have clear visibility—you know at all times the status of the project's cost, schedule, quality, and functionality. Bill Hetzel has found that strong measurement and tracking of project status characterize industry's best projects; in fact, these were evident in every "best project."[4]

If you don't track a project effectively, you can't manage it. You'll have no way of knowing whether your plans are being carried out or whether you need to modify the current plans. Effective tracking lets you detect problems early, while you still have time to take meaningful corrective action.

Typical general-management tracking controls include task lists, status meetings and reports, milestone reviews, budget reports, and management by walking around. These techniques tend to provide poor status visibility because they tend to track only cost or schedule—if you're not tracking cost, schedule, quality, and functionality, you're not tracking the project. Track any two of these characteristics, and undetected work will shift into the other two areas. If you track only task completion against a schedule (that is, you're tracking only functionality and schedule), unseen work will accumulate in the form of low quality. Developers will do the minimum amount of work necessary to declare a task done. Later in the project, you'll find that work that was reported as "done" is actually incomplete in many major and minor ways, and the project is behind schedule. The visibility you thought you had into functionality and schedule turns out

not to be very good because you weren't tracking quality. The same basic argument holds if you're not tracking any one of cost, schedule, quality, or functionality.

Well-run software projects track status in numerous quantitative ways—code, test, and review metrics; task lists; earned-value analysis; and control charts—in addition to more common techniques such as milestone reviews and management by walking around. Technical workers typically do not acquire the intellectual tools needed to do this work automatically; they require education and training targeted specifically at the software project manager.

Risk Management

Software projects are assaulted by risks arising from shifting user requirements, bleeding-edge technology, unstable tools, unreliable contractors, inexperienced personnel, and many other sources. "Risk" is a fighting word in much of the business world and isn't uttered aloud unless a project is already in trouble. But a software project manager who doesn't say "risk" at least a dozen times a day probably isn't doing his job. If the technical manager isn't actively managing risk, he isn't managing his project.

Measurement

One key to long-term progress in a software organization is collecting historical data to analyze software quality and productivity. Collecting a little historical data for each project can go a long way. If you collect data about effort (staff months), schedule (calendar time), program size in lines of code (or some other measurement), and defect count, you will have a solid basis for planning future projects.

Adding Tools to the Toolbox

Software management skills have at least as much influence on development success as technical skills do. The Software Engineering Institute has repeatedly observed that organizations that try to put software engineering discipline in place before project management discipline are doomed to fail.[5]

The common practice of promoting skilled technical workers into technical management without providing software-specific management training just turns good programmers into mediocre managers. Good technical managers are made, not born, and the software industry needs to make more of them. Fortunately, the current skills gap is in an area that's relatively easy to correct. Books such as Tom Gilb's *Principles of Software Engineering Management* (Addison-Wesley), Tom DeMarco's *Controlling Software Projects* (Yourdon Press), and my *Rapid Development* (Microsoft Press) contain good advice. You can find project management advice at the Project Management Institute's Web site (www.pmi.org) and Construx Software's technical reading list (www.construx.com/ladder).

References

1. *A Guide to The Project Management Body of Knowledge*, PMI Standards Committee, Project Management Inst., Newtown Square, Penn., 1996.
2. C. Jones, *Assessment and Control of Software Risks*, Yourdon Press, Englewood Cliffs, N.J., 1994.
3. C. Jones, "Patterns of Large Software Systems: Failure and Success," *IEEE Software*, Vol. 12, No. 2, Mar. 1995, pp. 86–87.
4. B. Hetzel, *Making Software Measurement Work: Building an Effective Measurement Program*, John Wiley & Sons, New York, 1993.
5. R. Burlton, "Managing a RAD Project: Critical Factors for Success," *Amer. Programmer*, Dec. 1992, pp. 22–29.