Contents

Excerpt from Chapter 3 Finding and Hiring Great Programmers

There are many programmers. However, there are not that many *great* programmers.

> *"Exceptional engineers are more likely than non-exceptional engineers to maintain a 'big picture,' have a bias for action, be driven by a sense of mission, exhibit and articulate strong convictions, play a pro-active role with management, and help other engineers," said an insightful 1993 study of software engineers.*[1]

Frederick Brooks in his classic work *The Mythical Man-Month*[2] cited a study[3] from 25 years earlier that showed, among programmers with two years' experience and similar training, that the best professional programmers are ten times as productive as the poorest of them. The researchers had started out to determine if changing from punch cards to interactive programming would make a productivity difference, only to find their results overwhelmed by the productivity differences among individuals. They found 20:1 differences in initial coding time, 5:1 differences in code size (!), and 25:1 differences in debugging time!

1. Richard Turley and James Bieman, *Competencies of Exceptional and Non-Exceptional Software Engineers* (Colorado State University, 1993).
2. Brooks, *The Mythical Man-Month.*
3. H. Sackman, W. J. Erikson, and E. E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *CACM*, January 1968.

Barry Boehm, 20 years later, reported a 25:1 difference between the most and least productive software developers, and a 10:1 difference in the number of bugs they generated.[4] In 2000, Boehm and coauthors updated their study to examine teams and concluded that teams of experienced top-tier programmers could be expected to be 5.3 times more productive than teams of inexperienced bottom-tier programmers.[5]

Good programmers are up to 30 times better than mediocre programmers, according to "individual differences" research. Given that their pay is never commensurate, they are the biggest bargains in the software field.

—*ROBERT L. GLASS, Software Practitioner, Pioneer, and Author*[6]

While there are some IT organizations that pride themselves on hiring "ordinary" programmers, there are few product companies and professional services organizations where you can be successful managing a software team without the ability to staff some part of your team with "great" ones. It's no wonder, given the kinds of people programmers can be, that finding and identifying exceptional engineers can be a challenge.

The single most important job of a programming manager is to hire the right people.

Hiring is far and away the most difficult-to-undo decision that managers make. Being successful at staffing will ease the rest of your job. The worst of unsuccessful hires can cast a plague upon your team for months, undermine your leadership, incite dissension and strife, delay or derail your deliverables, and in these ways and in every other way demotivate and demoralize your entire organization. Not to mention how hard it is to get rid of underperformers and other bad hires.

4. Barry Boehm, "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering*, October 1988.
5. Barry Boehm et al., *Software Cost Estimation with Cocomo II* (Addison-Wesley, 2000).
6. Barry Boehm et al., *Software Cost Estimation with Cocomo II* (Addison-Wesley, 2000).

If you're hiring not only programmers but also managers of programmers, remember the rule Ron heard at Apple and Mickey heard directly from Steve Jobs:

A's hire A's. B's hire C's.

—STEVE JOBS

Steve's point was to emphasize how essential it is to hire top-notch managers, for the combinatorial effect they have as they make hires.

We've both been fooled. Ron had already been hiring for a decade when he interviewed a manager he was convinced would be a stellar contributor to his organization: "I was certain, given how well he talked the talk, that this was a guy who would really deliver. I called two of his references, and both shared stories and anecdotes that convinced me he'd walked the talk many a time before.[7] My interview team was unanimous in making a 'hire' recommendation. It was a time when I'd inherited a bad apple or two, but I'd never hired one. Until then. I realized it fast and I acted quickly to communicate the change I wanted to see in his behavior. Luckily, when I called him into my office, not even two months on the job, for a change-or-leave meeting, it was he who opened the conversation: He didn't feel like he fit; he was giving notice; he needed to leave. I was lucky."

While it can happen, we've figured out a few principles that have resulted in the vast majority of our hires being good ones.

Determining What Kind of Programmer to Hire

It all starts with knowing whom you want to hire. You're hiring not just a programmer, but also someone to fill a role and a need in your organization.

We outlined in Chapter 2 how to build a job description for the kinds and levels of programmers you need in your organization. But those are generic descriptions.

For individual hires, only by consciously thinking through the skill sets, values, ethics, and orientation you need are you likely to hire the right programmers for the slots you need to fill out your team.

7. "You can't just talk the talk and walk the walk; you've got to walk the talk." This frequent theme of Cecil Williams, renowned pastor of San Francisco's Glide Memorial Church, I realized later, turns out to be the very definition of integrity.

Excerpt from Chapter 5 Becoming an Effective Programming Manager: Managing Down

Everything that has been presented in this book so far has only set the stage for the real heart of the information we hope to convey: how to manage programmers effectively on a day-to-day basis. Since most organizations are structured hierarchically, we have broken this information into four basic topics in relation to your position in the organization:

- Managing down
- Managing up
- Managing out
- Managing yourself

These topics deal with managing your staff, managing your boss(es), managing others inside or outside of your organization, and managing yourself. Chapter 6 covers the last three topics, which are critical to being an effective programming manager.

The most relevant topic here is the first one, and this chapter focuses solely on managing down. As a first-line programming manager, you should probably be spending the bulk of your time doing this. Managing down includes all those things you need to do to effectively manage the staff that reports directly, or indirectly, to you. One aspect of this, that of motivating your staff, is so important that Chapter 7 is devoted to it. The remainder of this chapter deals primarily with the mechanics of being an effective programming manager—what you need to do and how to do it.

Earning Technical Respect

Scott Adams has forever stained the reputation of every programming manager by making an icon of the Pointy-Haired Boss (or PHB, in programmer parlance). The PHB is a buffoon who is clueless at best and malicious or evil at his worst. We need not discuss what the PHB is but rather realize what he isn't—respected by Dilbert and his cohorts.

The PHB isn't respected because he does not understand, or care to understand, what Dilbert and his cohorts do, and he demonstrates that at every turn. Years of working in technical organizations have led us to believe that those who don't intimately understand programmers are going to make a mess if they try to manage, direct, or dictate actions for a programming team or project. It is the rare manager who has this understanding who has not been a programmer.

The single biggest key to successfully managing programmers is to have the technical respect of those you manage and your peers. Without technical respect, every attempt to manage will be thwarted actively or passively. This is why it is so hard for those who do not understand programmers (i.e., have not been programmers at some stage of their career) to manage programmers effectively. This is true of many technical disciplines but seems to be more of a truism in the world of programming.[1] Key aspects of earning technical respect are

- Understanding the art of computer programming
- Having a good track record
- Making some notable technical contribution
- Keeping up with technical trends and technologies
- Being an active member of technical or professional organizations
- Demonstrating strong personal values

To understand programmers, you must have a solid understanding of the tools, the processes, and the art of computer programming.[2] The deeper

1. We are not aware of any other vocation where there is a character such as Dilbert who throws such pointed barbs at his manager except, perhaps, the military.

2. Donald E. Knuth, *The Art of Computer Programming,* Volume 1, *Fundamental Algorithms, Third Edition* (Addison-Wesley, 1997); Volume 2, *Seminumerical Algorithms, Third Edition* (Addison-Wesley, 1997); Volume 3, *Sorting and Searching, Second Edition* (Addison-Wesley, 1998); Volume 4, *Combinatorial Algorithms, Part 1* (Addison-Wesley, 2011); Volume 5, *Syntactic Algorithms* (Addison-Wesley, forthcoming). If you don't have a copy of these books prominently displayed in your office, you should!

your understanding and the stronger your ability to engage in meaningful technical dialog with your staff, the more technical respect you will have from them. A Microsoft program architect once said of Bill Gates, "Gates relishes nothing as much as disassembling the bits and bytes of computer code with his programmers. He easily holds his own in the technological trenches. . . . He gets respect because he can take those guys to the cleaners."[3]

The intangible elements of technical respect explain why it can be difficult to bring programming managers in from outside the company or organization and have them be effective. A key attribute of any candidate you consider to manage a programming team is having a set of "bona fides" (i.e., a proven track record in software) that the team they will be managing can respect.

There are many ways to build a solid set of bona fides, the simplest of which is to be an acknowledged outstanding programmer/technical leader and be promoted to be a programming manager in the same organization. This has its own challenges, but an outstanding programmer will be a known quantity and have the needed technical respect of his peers and those he will then be responsible for managing, and that will help foster a good team culture that is founded on respect. Having a deep understanding of the technical organization and its managers will also be an advantage, since that understanding can be communicated to the team.

Another way to increase your stature, and in turn gain technical respect, is by having developed or managed projects or products that are well known to the programmers you manage.

Mickey's own management career was built by first being recognized as a key contributor at Evans & Sutherland (E&S) after leading the Picture System graphics library project, being promoted as a manager of the team developing the company's next-generation graphics products, and then being recruited by Pixar where his E&S work was well known.[4] Thereafter, having E&S and Pixar on his résumé gained him some technical respect that has carried forward to this very day.

Ron's bona fides for a management role at Apple began with coauthoring the canonical assembly language reference for the emerging

3. Paul Maritz, Microsoft program architect, *Playboy* magazine profile of Bill Gates, 1991.

4. Pixar, and Lucasfilm from which Pixar was spun off, were both heavy users of E&S's 3-D graphics systems. Pixar's groundbreaking short film *Luxo Jr.* was animated by John Lassiter using an E&S Picture System.

65816 microprocessor. When Apple chose the 65816 as the core of what would be a hybrid between the Mac and the Apple II, the Apple IIGS, Ron was recruited to code the animated program by which the IIGS demoed itself to buyers in every Apple store in the world. That led to Ron being recruited for a programming role at Apple in system software, during which time he was repeatedly tapped to manage the groups developing first the Apple II and then the Macintosh UI. That led Berkeley Systems, which had invented the change-the-channel screen saver *After Dark*, to hire him as their Director of Engineering for entertainment products. This led to directing engineering at two other entertainment software companies (not to mention the opportunity for Ron and Mickey to meet).

Pixar and Apple were important bona fides for Mickey and Ron, but any career can be packaged or enhanced with a little effort. This is an important aspect of career management, which every programmer or programming manager should work hard to do. Look for opportunities to make contributions, stand out from the crowd, and create your own legend. It can be as simple as contributing to an open source project, or blogging about your experiences. Find the right thing that works for you.

Joining and participating in relevant technical societies and organizations can contribute to your bona fides. We strongly recommend joining ACM and/or IEEE, either of which will bring anyone a measure of technical credibility regardless of their degree of participation. Long-standing members of either organization command even more technical respect. Attending annual conferences sponsored by these organizations as well as local chapter meetings is a great way to keep in touch with technical advances and do some personal and professional networking.

Other ways to gain technical respect are to get advanced technical degrees; become professionally certified; author technical papers; create open source, commercial, or shareware software; apply for patents; write a book; build your own Web site; start your own company; have your own blog; be a "known" contributor to a technical community (e.g., Slashdot); invent an algorithm (e.g., Page Rank, Warnock's algorithm); postulate a law (e.g., Moore's Law); and so on.

During his tenure managing Apple's Macintosh team developing the Finder—Apple's desktop UI—Ron developed the Macintosh shareware reminders program Birthdays and Such. Exploring coding best practices for one particularly gnarly area of Mac UI programming, Ron realized that Apple's own documentation was wrong and authored two Apple Tech

Excerpts from Rules of Thumb and Nuggets of Wisdom

The Challenges of Managing

Managers must manage.

—*ANDY GROVE, Former Intel Chairman and CEO*[1]

I've used Andy Grove's phrase innumerable times to coach my managers and directors of programming teams. When confronted with a problem, they can't just"raise a red flag."I'm always available when needed, but good software managers find ways to solve problems without my involvement or executive management direction.

Management is about human beings. Its task is to make people capable of joint performance, to make their strengths effective and their weaknesses irrelevant.

—*PETER DRUCKER*[2]

Drucker, born in 1909, is the man who has been called the"father of modern management."He coined the term *knowledge worker* and predicted the rise of today's information society and its need for lifelong learning.

1. Paraphrased from Andrew S. Grove, *High-Output Management* (Vintage Books, 1995).
2. Peter Drucker, "Management as Social Function and Liberal Art," in *The Essential Drucker* (Harper Business, 2001), p. 10.

A manager of years ago gave sage career advice: When you land at a new company, pick a gnarly problem—one that people have been avoiding—and solve it. It gets you up the learning curve, and it gains you credibility and respect, both of which you'll need to be an effective developer and influencer.

—DAVE SMITH, Agile Software Development Coach[3]

People hate change but love progress.

—TERRY PEARCE, Author, Leading Out Loud

It is not enough to respond to change; we must lead change or be left behind.

—POLLYANNA PIXTON, Founder, Agile Leadership Network (ALN)[4]

You miss 100 percent of the shots you never take.

—WAYNE GRETZKY, Hockey Phenom

I have missed more than 9,000 shots in my career. I have lost almost 300 games. On 26 occasions I have been entrusted to take the game-winning shot and I missed. I have failed over and over again in my life. And that's precisely why I succeed.

—MICHAEL JORDAN, Basketball Phenom[5]

Nothing stops a witch hunt faster than owning up to being the culprit.

—TIM SWIHART, Engineering Director, Apple Computer

You know when it's you (or your team) that the angry mob is searching for. Stand up, succinctly explain what went wrong, why

3. http://c2.com/cgi/wiki?WorstThingsFirst.
4. Spoken at BayALN, the Bay Area chapter, January 2007.
5. Quoted in *Guideposts*, August 2002.

Managing People

Trust your feelings, young Luke. The Force will be with you.

—*OBI-WAN KENOBI, Jedi Master in* Star Wars

We counsel many managers and programmers to listen to their intuition carefully; it's usually right. The older I get, the more I regret those times when I don't listen to my intuition. Programmers usually know the right things to do—if they allow themselves to trust their feelings.

Accountability is not micromanagement.

—*MARK HIMELSTEIN*

The author of *100 Questions to Ask Your Software Organization,*[1] speaking to the Best Practices SIG of the East Bay Innovation Group in 2006. We've always hated being micromanaged and thus have avoided being managers who micromanage. The challenge is to realize the line between micromanaging and holding your people accountable. Giving up micromanagement is not giving up expecting accountability.

1. Mark I. Himelstein, *100 Questions to Ask Your Software Organization (Infinity Publishing, 2005).*

Trust but verify.

—*RONALD REAGAN*

President Reagan was referring to the Soviets during the Cold War with his frequent use of this translation of a Russian proverb that had been equally frequently quoted by Soviet founding father Vladimir Lenin. Perhaps it was because of that context that I didn't pay attention until I heard another VP of Engineering use the expression in describing how he managed his team—and realized that it described my goal-state management style: my goal is absolutely no micromanagement, but enough checking to know that the delegation I'd done was appropriate.

One of the surest sources of delay and confusion is to allow any superior to be directly responsible for the control of too many subordinates.

—*V. A. GRAICUNAS*

Graicunas was an early-twentieth-century management consultant and the first to mathematically analyze the complexity of increasing management responsibility. Graicunas showed that as span of control increases, the number of interactions among managers and their reports—and thus the amount of time managers must spend supervising—increases geometrically. His formula takes into account manager-to-report, report-to-report, and manager-to-all-combinations-of-reports interactions, and he posits that supervision time increases proportionately with interactions. He showed that by adding a fifth report, while the potential for accomplishing more work may increase by 20 percent, the number of potential interactions increases from 44 to 100—by 127 percent! Eight reports increases to 1,080 potential interactions and 12 reports to 24,564 to track and manage! Graicunas recommended that managers have a maximum of five reports, ideally four.

Managing Teams to Deliver Successfully

Software is hard.

—*DONALD KNUTH*

Knuth is the acclaimed author of *The Art of Computer Programming.* Here, he was quoted speaking to an audience of 350 people at the Technische Universitat Munchen.[1]

Software isn't released, it's allowed to escape.

—*PROJECT MANAGEMENT LAY WISDOM*

. . . you're not here to write code; you're here to ship products.

—*JAMIE ZAWINSKI*[2]

Hofstadter's Law: It always takes longer than you expect, even when you take Hofstadter's Law into account.

—*DOUGLAS HOFSTADTER*[3]

1. Knuth's comment was captured in the widely shared PDF, *All Questions Answered.*
2. Quoted in Seibel, *Coders at Work*, p. 22.
3. Douglas Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid, 20th Anniversary Edition* (Basic Books, 1999).

Always take redundancy over reliability.

—*VLADIMIR BOGDANOV*

Too much data could be worse than no data at all.

—*VLADIMIR BOGDANOV*

The secret of software development has always been good people and lots of Chinese food.

—*JEFF KENTON, Consulting Developer and Development Manager*[4]

Lots of Chinese food or pizza. Critical projects or milestones often require extraordinary effort to be exerted by the programming team. It's a simple thing that should be obvious: Programmers under schedule pressure will work long hours unless they are interrupted. Bringing in meals—Chinese food, pizza, or even meals from a great restaurant—keeps the programming team going. If you allow them to go out for food, their will to continue will erode and the extra effort will diminish.

Projects should be run like marathons. You have to set a healthy pace that can win the race and expect to sprint for the finish line.

—*ED CATMULL, CTO, Pixar Animation Studios*

At Pixar, Ed Catmull, its cofounder, president, and CTO, encouraged me to manage my projects this way, and I've used this as a rule of thumb ever since. When interviewing new candidates, I make sure I find a way to bring this up so that I set the expectation that there will be times when the programmer will need to work *very* hard and sprint for the finish line and project completion, but also to set the expectation that I don't expect them to "sprint" all the time.

4. www.embedded.com/shared/printableArticle.jhtml?articleID=166400761.

Excerpt from Chapter 8 Establishing a Successful Programming Culture

One of the questions we asked back in Chapter 1 was "What is a great programmer?"

But a question more relevant to you is "What is a great manager of programmers?" An essential and significant element of your role as a great manager is to create and nurture a successful programming culture. For most of us, that's a culture that supports and encourages the delivery of quality software on time and within budget by a team that developers feel proud and gratified to be part of for a long time.

You were hired to manage, right? But even if you follow all of our earlier advice, it's not easy. Your programmers don't always act rationally or predictably. Some have chaotic personal lives. They don't always get along. They can be blunt, reclusive, irritable, manic, silent, impatient, petulant, abrasive . . .

Your organization may not care much about them (unless their irrational behavior spews beyond your department, of course). But your organization cares a lot about your ability to produce and deploy software that meets organizational goals and customer needs.

Almost any group of programmers, no matter how dysfunctional, will care, too. They care about being productive and building successful products and services.

As for you, you care even more. In addition to wanting what your developers want, and wanting to meet your organization's expectations, you want to be a high-performing software development manager who can stretch beyond the ordinary to achieve the remarkable.

You need help. You somehow need to create internal and external expectations for greatness. You need to instill confidence that you and your team(s) can deliver. You need a culture that supports your goals and objectives. And you need to create an environment of excellence that attracts and retains top talent and motivates stellar work.

Powerful cultures drive high-performance work in ways that no amount of personal motivation alone can achieve.

Under the right conditions, the problems of commitment, alignment, motivation, and change largely melt away.

—JIM COLLINS[1]

Defining "Successful"

OK, so it may not be greatness you need to deliver. For some projects it may be functional but frequent delivery. For others your stakeholders may expect their product to be "flawless." Some teams are formed to help visionaries conceptualize products. Other teams are formed to keep products running as the environments they're built within change.

You may find you have organizational goals as well, goals such as developing and retaining quality programmers, perhaps.

It is essential to creating and nurturing a successful programming culture that you understand what "successful" means for your company, your organization, your project, and your team—and how to measure it.

The Programming Culture

Unless you lucked out and inherited it, you have to create your own successful programming culture. To maintain it, even if you inherited it, you need to nurture it. These are truisms whether you and your team are developing packaged software, software as a service, embedded software, B2B software components and services, or internal applications for the firm's employees. They are true whether you're part of a tiny start-up, a large corporation, a nonprofit, or government. Your mission is to deliver value. And that requires managing the people and the culture.

1. Jim Collins, *Good to Great* (HarperCollins, 2001), p. 11.

Creating a powerful programming culture requires establishing

- A work setting that is conducive to developing outstanding quality software and values on-time creation and delivery of on-target, customer-focused software
- An atmosphere of respect and fairness that keeps your staff at their most productive
- An environment in which commitment and motivation are easily nurtured and grown
- Metrics for your products, projects, and deliverables so your team can measure its efforts and improve its results

The challenge is: How do you do that?

Company Culture

All organizations—large and small, companies, governments, and nonprofits alike—have a corporate culture already. It's important to understand your organization's culture in order to create the culture you desire.

If it's a strong, positive culture, it may provide you with a platform you can leverage to create the right environment for your own team. Or it may be one that is so corrosive that you need to wall it off entirely to give your team an insular environment in which it can accomplish good work undistracted.

To figure out the corporate culture, listen to the CEO. Steve Jobs in a video conversation in 2007 (with Bill Gates), for example, said he wanted employees at Apple to feel like they were doing the best work of their careers. Chuck Schwab used those same words in the mid-nineties when the brokerage was moving stock trading online.

In most large companies, it's easy to identify the culture and values the company espouses. You'll find well-phrased statements of vision and values, sometimes referred to as purpose and principles, printed on posters and plaques, T-shirts and coffee cups, the company Web site and laminated wallet cards.

That said, view what you hear and see with some skepticism. What companies espouse is not always how they behave. Look deeper than the words. Enron claimed to stand "on the foundation of its Vision and Values," trumpeting values that included "Respect, Integrity and Communication." Regarding respect, its Web site expounded that "ruthlessness, callousness and arrogance don't belong here." That hardly seems consistent with the directive CEO Jeffrey Skilling is said to have given his management team to "cut jobs ruthlessly by 50 percent."

Even where values have been forsaken, smart development managers recognize that the company's values, with words painted everywhere, can be leveraged. A good development manager at Enron would have forged a programming culture around "respect, integrity, and communication," regardless of their absence in the milieu around them.

For organizations less than 30 or 40 years old, our experience is that the culture almost without fail reflects the personal standards and core values of the company's founder(s). Listen to stories from the earliest employees about how the founder established and grew the company. The stories at Apple of Steve Jobs leading the Mac team—virtually living together, working day and night, printing T-shirts with "Working 90 hours a week and loving it," raising a pirate flag in fierce pride and defiance—without question forged intense esprit de corps. But this environment also created interdepartmental rivalries and frustrated cross-company collaboration. In fact, Apple was long an aggregation of teams more than an integrated company.

Ultimately, culture derives not from the words that are espoused, but from the lessons that are communicated through action. Look for how employees, shareholders, and customers are perceived and treated—and the stories employees tell—regardless of the culture your organization claims to live by.

Leveraging the Complexity of Your Company's Culture

At Charles Schwab, Ron created a small department of 25 to lead a three-year initiative to move all of Schwab's application development to Java. "More than any other company I've seen, Schwab's values were applied equally to employees, customers, and shareholders alike. Daily interactions more often than not mirrored its published values. Schwab's espoused principles—*Fairness, Empathy, Responsiveness, Striving, Teamwork, Trustworthiness*—were modeled by founder Chuck Schwab. Whether seen from inside the company or out, Chuck Schwab is a man who is both extraordinarily entrepreneurial and at the same time one of the most caring and ethical heads of any company anywhere. Where *Teamwork* in most companies refers to *your* team, there was a sense at Schwab that teamwork meant everyone."

But even in the best of companies, the best of values can be a mixed blessing. Ron's Java initiative, at its core, was about building and sharing best practices, about finding and sharing common ways to do things. A slam dunk in an organization with *Teamwork* as a core value, right? Getting teams to share best practices, approaches, patterns, and even code should be easy.

Sample Tools

Sample Tools are available for download from managingtheunmanageable.net.

TITLE:	Programmer 3
DEPARTMENT:	Client Programming
REPORTS TO:	Director, Client Programming
STATUS:	Full-time, exempt
LOCATION:	San Francisco, CA

POSITION SUMMARY: Entry-level position. Responsible for code and/or asset management, conversion, verification, and maintenance. Responsible for writing well-defined portions of source code adhering to established standards of quality for documentation and coding. Works well in a group, and follows direction from manager and senior team members. Expected to work under direct supervision, communicating issues and problems that arise.

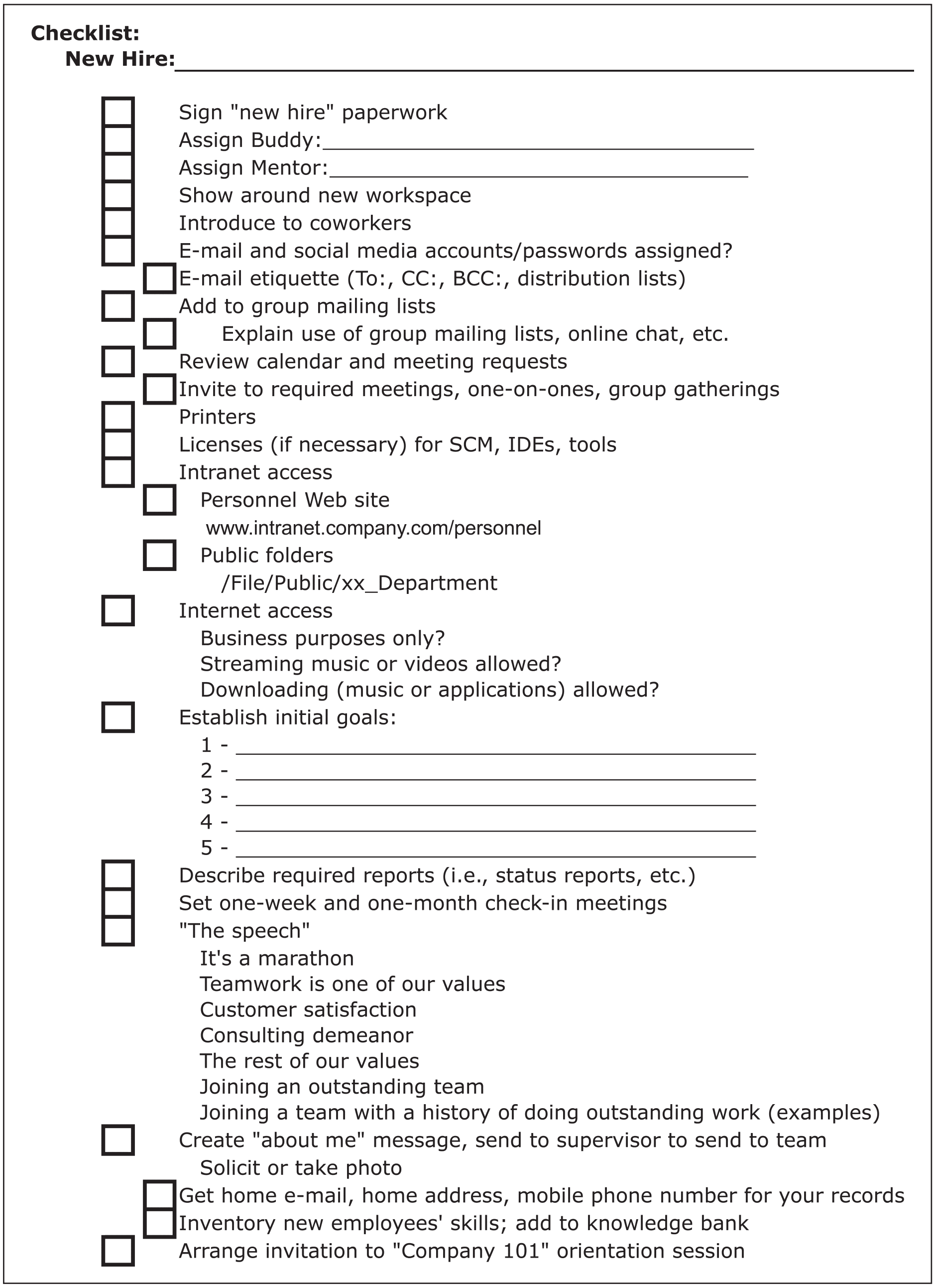
JOB REQUIREMENTS

- Four-year college degree in computer science or equivalent experience.
- Knowledge of Windows, Mac, or Linux/UNIX with more than one platform preferable.
- Knowledge of C/C++ and debugging techniques.
- Basic knowledge of good coding practices and fundamental computer science principles.
- Aware of and interested in Internet technologies, communication protocols, and techniques.
- Aware of and interested in database methodologies and database systems.
- Ability to work in a team and take direction well.
- Self-motivated and responds to supervision. Asks relevant questions.
- Enthusiastic about company and programming company products.
- Can work with supervisor to plan tasks and estimate their completion.
- Can adapt to changing conditions.

Figure 2.1 Sample job description

Table 2.1 Client Programming Level Guidelines

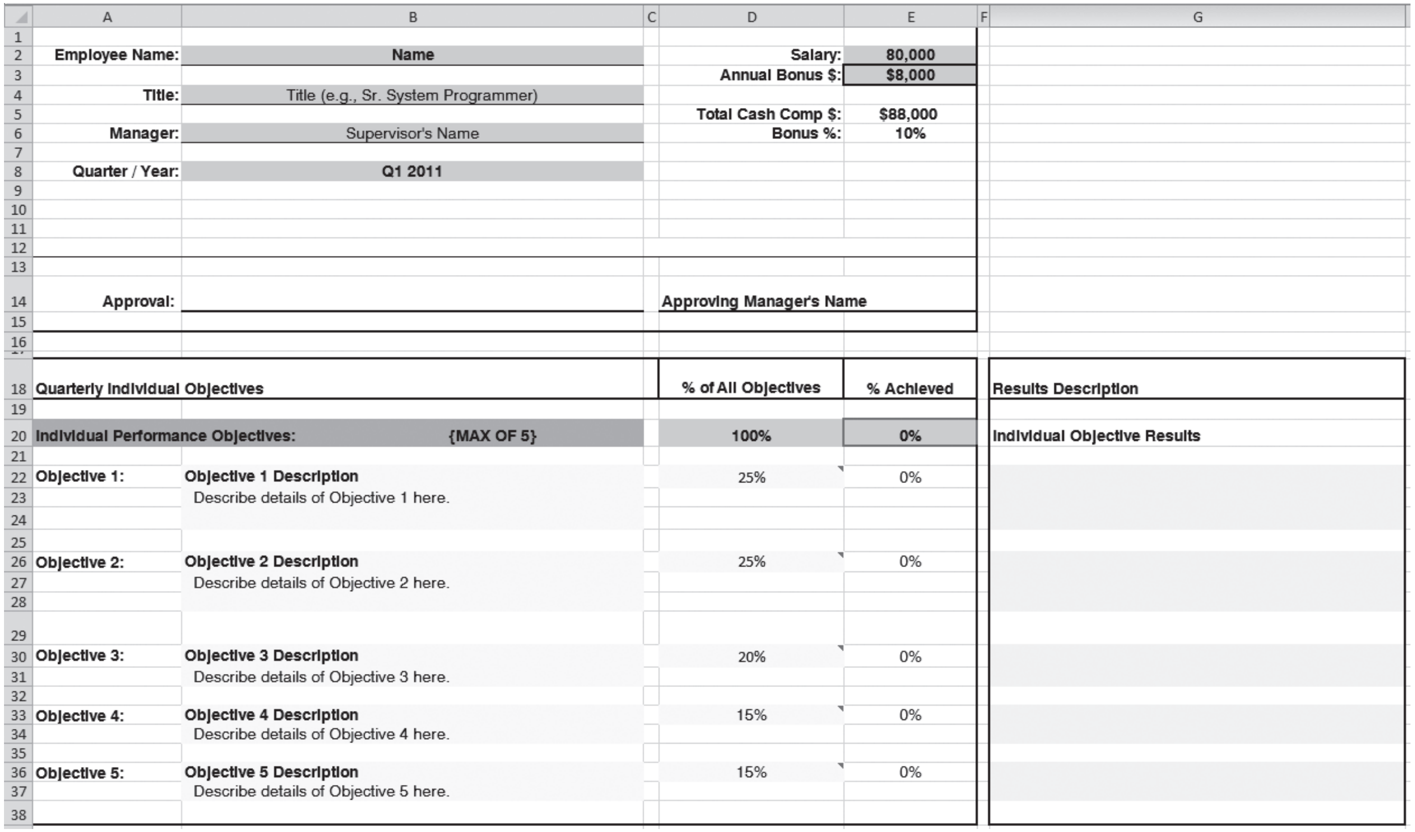
Programming Levels	Client Programmers
Entry level	Programmer 3
1–5 years' experience	Programmer 2
Experienced (5–10 years)	Programmer 1
Experienced (10–20 years)	Senior Programmer 2
Very experienced (12+ years)	Senior Programmer 1/Architect



Checklist:
New Hire:____

- ☐ Sign "new hire" paperwork
- ☐ Assign Buddy:______________________________
- ☐ Assign Mentor:______________________________
- ☐ Show around new workspace
- ☐ Introduce to coworkers
- ☐ E-mail and social media accounts/passwords assigned?
  - ☐ E-mail etiquette (To:, CC:, BCC:, distribution lists)
- ☐ Add to group mailing lists
  - ☐ Explain use of group mailing lists, online chat, etc.
- ☐ Review calendar and meeting requests
  - ☐ Invite to required meetings, one-on-ones, group gatherings
- ☐ Printers
- ☐ Licenses (if necessary) for SCM, IDEs, tools
- ☐ Intranet access
  - ☐ Personnel Web site
    www.intranet.company.com/personnel
  - ☐ Public folders
    /File/Public/xx_Department
- ☐ Internet access
  - Business purposes only?
  - Streaming music or videos allowed?
  - Downloading (music or applications) allowed?
- ☐ Establish initial goals:
  - 1 - ______________________________
  - 2 - ______________________________
  - 3 - ______________________________
  - 4 - ______________________________
  - 5 - ______________________________
- ☐ Describe required reports (i.e., status reports, etc.)
- ☐ Set one-week and one-month check-in meetings
- ☐ "The speech"
  - It's a marathon
  - Teamwork is one of our values
  - Customer satisfaction
  - Consulting demeanor
  - The rest of our values
  - Joining an outstanding team
  - Joining a team with a history of doing outstanding work (examples)
- ☐ Create "about me" message, send to supervisor to send to team
  - Solicit or take photo
  - ☐ Get home e-mail, home address, mobile phone number for your records
  - ☐ Inventory new employees' skills; add to knowledge bank
- ☐ Arrange invitation to "Company 101" orientation session

Figure 4.1 First-day preparation checklist



	A	B	C	D	E	F	G
1							
2	Employee Name:	Name		Salary:	80,000		
3				Annual Bonus $:	$8,000		
4	Title:	Title (e.g., Sr. System Programmer)					
5				Total Cash Comp $:	$88,000		
6	Manager:	Supervisor's Name		Bonus %:	10%		
7							
8	Quarter / Year:	Q1 2011					
9							
10							
11							
12							
13							
14	Approval:			Approving Manager's Name			
15							
16							
18	Quarterly Individual Objectives			% of All Objectives	% Achieved		Results Description
19							
20	Individual Performance Objectives:	{MAX OF 5}		100%	0%		Individual Objective Results
21							
22	Objective 1:	Objective 1 Description		25%	0%		
23		Describe details of Objective 1 here.					
24							
25							
26	Objective 2:	Objective 2 Description		25%	0%		
27		Describe details of Objective 2 here.					
28							
29							
30	Objective 3:	Objective 3 Description		20%	0%		
31		Describe details of Objective 3 here.					
32							
33	Objective 4:	Objective 4 Description		15%	0%		
34		Describe details of Objective 4 here.					
35							
36	Objective 5:	Objective 5 Description		15%	0%		
37		Describe details of Objective 5 here.					
38							

Figure 5.2 Quarterly objectives form